Training MRFs, CRFs, and SSVMs

Mark Schmidt

University of British Columbia

August, 2015

Summary of Week 1 and Framing Week 2

- We used structured prediction to motivate studying UGMs:

Input: 

Output: "Paris"

Summary of Week 1 and Framing Week 2

- We used structured prediction to motivate studying UGMs:

Input: 

Output: "Paris"

- We talked about defining an energy function $E(Y|X)$:
  - Want low eneryg for correct labels.
  - Energy will depend on features $F(Y, X)$.
  - Parts Y that occur in same feature define the graph.

Summary of Week 1 and Framing Week 2

- We used structured prediction to motivate studying UGMs:

Input: 

Output: "Paris"

- We talked about defining an energy function $E(Y|X)$:
  - Want low eneryg for correct labels.
  - Energy will depend on features $F(Y, X)$.
  - Parts Y that occur in same feature define the graph.
- But last week we got side-tracked by inference problems:
  - We considered decoding, inference, and sampling.
  - We considered exact methods to do these tasks.

Summary of Week 1 and Framing Week 2

- We used structured prediction to motivate studying UGMs:

Input: 

Output: "Paris"

- We talked about defining an energy function $E(Y|X)$:
  - Want low eneryg for correct labels.
  - Energy will depend on features $F(Y, X)$.
  - Parts Y that occur in same feature define the graph.
- But last week we got side-tracked by inference problems:
  - We considered decoding, inference, and sampling.
  - We considered exact methods to do these tasks.
- This week:
  - Learning parameters of $E(Y|X)$.
  - Approximate inference methods.

Learning in Unconditional Models

- We will first consider the unconditional case:

  (AKA Markov random field)

  - Input is a sequence of samples $X_i = (x_1, x_2, x_3, \dots, x_d)$.
  - Assume we have a parameterization of our potentials.
  - Assume we are given the graph structure (until Friday).
  - Output is the 'best' parameters (e.g., maximum likelihood).

Learning in Unconditional Models

- We will first consider the unconditional case:

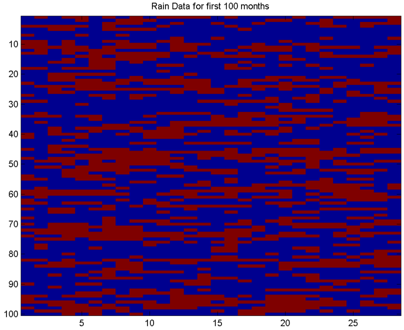
  (AKA Markov random field)
  - Input is a sequence of samples $X_i = (x_1, x_2, x_3, \ldots, x_d)$.
  - Assume we have a parameterization of our potentials.
  - Assume we are given the graph structure (until Friday).
  - Output is the 'best' parameters (e.g., maximum likelihood).
- Typically leads to better model than hand-tuned parameters.
- Usually, decoding/inference/sampling is a sub-routine in learning.

Example: Vancouver Rain Data

- Vancouver Rain dataset:
  - Binary X_i is whether it rained or not on first 28 days of month i.
  - Dataset contains 1059 months from 1896-2004.

Example: Vancouver Rain Data

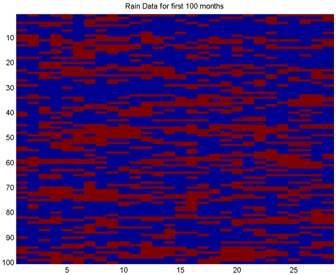
- Vancouver Rain dataset:
  - Binary X_i is whether it rained or not on first 28 days of month i.
  - Dataset contains 1059 months from 1896-2004.
  - First 100 months (red means red):

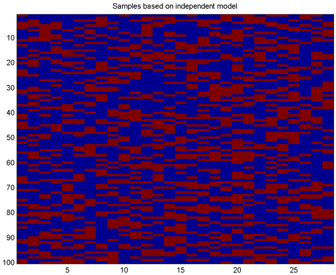


  - Sadly, $p(x_i = r) = 0.41$.

Example: Vancouver Rain Data

Real data vs. sampling day indepenedently with probability 0.41:

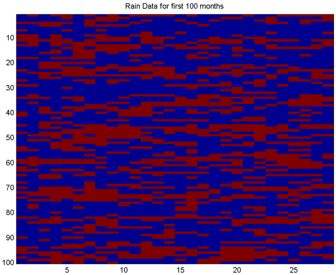


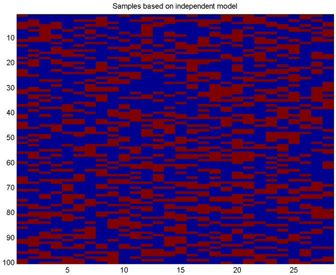


- Independent model misses correlations between days.

Example: Vancouver Rain Data

Real data vs. sampling day indepenedently with probability 0.41:





- Independent model misses correlations between days.
- We can do better with a UGM:
    - But we're not going to make up potentials.
    - Use the data to find the best potentials!

Maximum Likelihood Formulation

- Let's fit the parameters using maximum likelihood of data:

  (assuming the X_i are independent)

$$w = \underset{w}{\text{argmax}} \prod_{i=1}^{n} p(X_i|w),$$

Maximum Likelihood Formulation

- Let's fit the parameters using maximum likelihood of data:

  (assuming the X_i are independent)

$$w = \underset{w}{\text{argmax}} \prod_{i=1}^{n} p(X_i|w),$$

or equivalently minimize negative log-likelihood (NLL),

$$w = \underset{w}{\text{argmin}} -\frac{1}{n} \sum_{i=1}^{n} \log(p(X_i|w)),$$

Maximum Likelihood Formulation

- Let's fit the parameters using maximum likelihood of data:

  (assuming the X_i are independent)

$$w = \underset{w}{\text{argmax}} \prod_{i=1}^{n} p(X_i|w),$$

or equivalently minimize negative log-likelihood (NLL),

$$w = \underset{w}{\text{argmin}} -\frac{1}{n}\sum_{i=1}^{n} \log(p(X_i|w)),$$

and you could/should also use a regularizer,

$$w = \underset{w}{\text{argmin}} -\frac{1}{n}\sum_{i=1}^{n} \log(p(X_i|w)) + \frac{\lambda}{2}\|w\|^2.$$

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}).$$

where m maps exponentiated 'parameters' to potentials.

(m called 'nodeMap' and 'edgeMap' in UGM code)

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}).$$

  where m maps exponentiated 'parameters' to potentials.

  (m called 'nodeMap' and 'edgeMap' in UGM code)

- Parameter tieing can be done with choice of m:

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}).$$

  where m maps exponentiated 'parameters' to potentials.

  (m called 'nodeMap' and 'edgeMap' in UGM code)

- Parameter tieing can be done with choice of m:
  - If $m(i, x_i) = x_i$ for all i, each day has same potentials.

    (parameters are tied)

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}).$$

  where m maps exponentiated 'parameters' to potentials.

  (m called 'nodeMap' and 'edgeMap' in UGM code)

- Parameter tieing can be done with choice of m:
  - If $m(i, x_i) = x_i$ for all i, each day has same potentials.

    (parameters are tied)
  - If $m(i, x_i) = x_i(n-1) + i$ for all i, each day has different potentials.

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}).$$

  where m maps exponentiated 'parameters' to potentials.

  (m called 'nodeMap' and 'edgeMap' in UGM code)

- Parameter tieing can be done with choice of m:
  - If $m(i, x_i) = x_i$ for all i, each day has same potentials.

    (parameters are tied)
  - If $m(i, x_i) = x_i(n-1) + i$ for all i, each day has different potentials.
  - We could have groups: E.g., weekdays vs. weekends, or boundary.
  - We'll use the convention that $m(i, x_i) = 0$ means that $\phi_i(x_i) = 1$.

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}).$$

  where m maps exponentiated 'parameters' to potentials.

  (m called 'nodeMap' and 'edgeMap' in UGM code)

- Parameter tieing can be done with choice of m:
  - If $m(i, x_i) = x_i$ for all i, each day has same potentials.

    (parameters are tied)
  - If $m(i, x_i) = x_i(n-1) + i$ for all i, each day has different potentials.
  - We could have groups: E.g., weekdays vs. weekends, or boundary.
  - We'll use the convention that $m(i, x_i) = 0$ means that $\phi_i(x_i) = 1$.
  - Similar logic holds for edge potentials.

Example: Ising Model of Rain Data

- E.g., we could parameterize our node potentials using

$$\log(\phi_i(x_i)) = \begin{cases} w_1 & \text{rain} \\ 0 & \text{no rain} \end{cases},$$

  and one parameter is enough since scale of ϕ_i is arbitrary.

  (though might want two parameters if using regularization)

Example: Ising Model of Rain Data

- E.g., we could parameterize our node potentials using

$$\log(\phi_i(x_i)) = \begin{cases} w_1 & \text{rain} \\ 0 & \text{no rain} \end{cases},$$

  and one parameter is enough since scale of ϕ_i is arbitrary.

  (though might want two parameters if using regularization)

- Use chain-structure, Ising parameterization of edge potentials,

$$\log(\phi_{ij}(x_i, x_j)) = \begin{cases} w_2 & x_i = x_j \\ 0 & x_i \neq x_j \end{cases}.$$

Example: Ising Model of Rain Data

- E.g., we could parameterize our node potentials using

$$\log(\phi_i(x_i)) = \begin{cases} w_1 & \text{rain} \\ 0 & \text{no rain} \end{cases},$$

  and one parameter is enough since scale of ϕ_i is arbitrary.

  (though might want two parameters if using regularization)

- Use chain-structure, Ising parameterization of edge potentials,

$$\log(\phi_{ij}(x_i, x_j)) = \begin{cases} w_2 & x_i = x_j \\ 0 & x_i \neq x_j \end{cases}.$$

- The maximum likelihood solution is

$$w = \begin{bmatrix} 0.16 \\ 0.85 \end{bmatrix}, \quad \phi_i = \begin{bmatrix} \exp(w_1) \\ \exp(0) \end{bmatrix} = \begin{bmatrix} 1.17 \\ 1 \end{bmatrix}, \quad \phi_{ij} = \begin{bmatrix} 2.34 & 1 \\ 1 & 2.34 \end{bmatrix},$$

  preference towards no rain, and adjacent days being the same.

- Average NLL of 16.8 vs. 19.0 for independent model.

Full Model of Rain Data

- We could alternately use fully expressive edge potentials

$$\log(\phi_{ij}(x_i, x_j)) = \begin{bmatrix} w_2 & w_3 \\ w_4 & 0 \end{bmatrix},$$

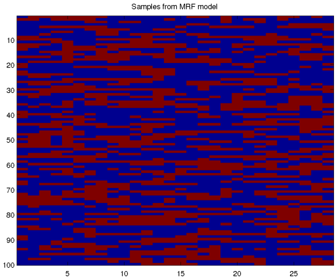
  but these don't improve the likelihood much.
- We could also have special potentials for the boundaries.

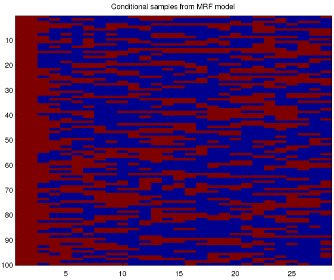
Full Model of Rain Data

- We could alternately use fully expressive edge potentials

$$\log(\phi_{ij}(x_i, x_j)) = \begin{bmatrix} w_2 & w_3 \\ w_4 & 0 \end{bmatrix},$$

  but these don't improve the likelihood much.
- We could also have special potentials for the boundaries.
- Samples from model and conditional samples if rain on first day:





Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}),$$

- We've excluded $\phi_i = 0$, but otherwise this is not restrictive.

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}),$$

- We've excluded $\phi_i = 0$, but otherwise this is not restrictive.
- Energy function $E(X_i)$ will be linear,

$$\begin{aligned}
E(X) &= \log\left(\prod_i \phi_i(x_i) \prod_{(i,j)\in E} \phi_{ij}(x_i, x_j)\right) \\
&= \log\left(\exp\left(\sum_i w_{m(i,x_i)} + \sum_{(i,j)\in E} w_{m(i,j,x_i,x_j)}\right)\right) \\
&= \sum_i w_{m(i,x_i)} + \sum_{(i,j)\in E} w_{m(i,j,x_i,x_j)}.
\end{aligned}$$

Log-Linear Parameterization of MRFs

- We'll use a log-linear parameterization:

$$\phi_i(x_i) = \exp(w_{m(i,x_i)}), \quad \phi_{ij}(x_i, x_j) = \exp(w_{m(i,j,x_i,x_j)}),$$

- We've excluded $\phi_i = 0$, but otherwise this is not restrictive.
- Energy function $E(X_i)$ will be linear,

$$\begin{aligned} E(X) &= \log\left(\prod_i \phi_i(x_i) \prod_{(i,j)\in E} \phi_{ij}(x_i, x_j)\right) \\ &= \log\left(\exp\left(\sum_i w_{m(i,x_i)} + \sum_{(i,j)\in E} w_{m(i,j,x_i,x_j)}\right)\right) \\ &= \sum_i w_{m(i,x_i)} + \sum_{(i,j)\in E} w_{m(i,j,x_i,x_j)}. \end{aligned}$$

- To make notation simpler, consider this identity

$$w_{m(i,x_i)} = \sum_f w_f \mathcal{I}[m(i, x_i) = f],$$

Feature Vector Representation

Use this identity to write any log-linear energy in a simple form

$$
\begin{aligned}
E(X) &= \sum_i w_{m(i,x_i)} + \sum_{(i,j)\in E} w_{m(i,j,x_i,x_j)} \\
&= \sum_i \sum_f w_f \mathcal{I}[m(i,x_i) = f] + \sum_{(i,j)\in E} \sum_f w_f \mathcal{I}[m(i,j,x_i,x_j) = f] \\
&= \sum_f w_f \left(\sum_i \mathcal{I}[m(i,x_i) = f] + \sum_{(i,j)\in E} \mathcal{I}[m(i,j,x_i,x_j) = f] \right) \\
&= w^T F(X),
\end{aligned}
$$

where $F_f(X) \triangleq \sum_i \mathcal{I}[m(i,x_i) = f] + \sum_{(i,j)\in E} \mathcal{I}[m(i,j,x_i,x_j) = f]$ are sufficient statistics of the example.

Feature Vector Representation

Use this identity to write any log-linear energy in a simple form

$$
\begin{aligned}
E(X) &= \sum_i w_{m(i,x_i)} + \sum_{(i,j)\in E} w_{m(i,j,x_i,x_j)} \\
&= \sum_i \sum_f w_f \mathcal{I}[m(i,x_i) = f] + \sum_{(i,j)\in E} \sum_f w_f \mathcal{I}[m(i,j,x_i,x_j) = f] \\
&= \sum_f w_f \left(\sum_i \mathcal{I}[m(i,x_i) = f] + \sum_{(i,j)\in E} \mathcal{I}[m(i,j,x_i,x_j) = f] \right) \\
&= w^T F(X),
\end{aligned}
$$

where $F_f(X) \triangleq \sum_i \mathcal{I}[m(i,x_i) = f] + \sum_{(i,j)\in E} \mathcal{I}[m(i,j,x_i,x_j) = f]$ are sufficient statistics of the example.

E.g., in Ising model $F_1(X)$ is number of times it rained in X and $F_2(X)$ is number adjacent days that have the same value.

MRF Training Objective Function

- With log-linear parameterization, NLL takes the form

$$
\begin{aligned}
f(w) &= -\frac{1}{n}\sum_{i=1}^{n}\log p(X_i|w) = -\frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{\exp(w^T F(X_i))}{Z(w)}\right)\\
&= -\frac{1}{n}\sum_{i=1}^{n} w^T F(X_i) + \frac{1}{n}\sum_{i=1}^{n}\log Z(w)\\
&= -w^T F(D) + \log Z(w).
\end{aligned}
$$

where $F(D) = \frac{1}{n}\sum_i F(X_i)$ is sufficient statistics of data.

MRF Training Objective Function

- With log-linear parameterization, NLL takes the form

$$
\begin{aligned}
f(w) &= -\frac{1}{n}\sum_{i=1}^{n}\log p(X_i|w) = -\frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{\exp(w^T F(X_i))}{Z(w)}\right) \\
&= -\frac{1}{n}\sum_{i=1}^{n} w^T F(X_i) + \frac{1}{n}\sum_{i=1}^{n}\log Z(w) \\
&= -w^T F(D) + \log Z(w).
\end{aligned}
$$

  where $F(D) = \frac{1}{n}\sum_i F(X_i)$ is sufficient statistics of data.

- Given sufficient statistics $F(D)$, can throw out data X_i.

  (only go through data once)

- Function $f(w)$ is convex.
- With $\|w\|^2$ regularizer, unique solution is guaranteed to exist.

Optimization with MRFs

- With log-linear parameterization, NLL takes the form

$$f(w) = -w^T F(D) + \log Z(w).$$

Optimization with MRFs

- With log-linear parameterization, NLL takes the form

$$f(w) = -w^T F(D) + \log Z(w).$$

- Gradient with respect to parameter f is given by

$$\begin{aligned} -\nabla_f f(w) &= F_f(D) - \sum_X \frac{\exp(w^T F(X))}{Z(w)} F_f(X) \\ &= F_f(D) - \sum_X p(X) F_f(X) \\ &= F_f(D) - \mathbb{E}_X[F_f(X)]. \end{aligned}$$

Optimization with MRFs

- With log-linear parameterization, NLL takes the form

$$f(w) = -w^T F(D) + \log Z(w).$$

- Gradient with respect to parameter f is given by

$$\begin{aligned}
-\nabla_f f(w) &= F_f(D) - \sum_X \frac{\exp(w^T F(X))}{Z(w)} F_f(X) \\
&= F_f(D) - \sum_X p(X) F_f(X) \\
&= F_f(D) - \mathbb{E}_X[F_f(X)].
\end{aligned}$$

- Derivative of $\log(Z)$ is marginal of feature.

(inference required for learning)

Optimization with MRFs

- With log-linear parameterization, NLL takes the form

$$f(w) = -w^T F(D) + \log Z(w).$$

- Gradient with respect to parameter f is given by

$$\begin{aligned} -\nabla_f f(w) &= F_f(D) - \sum_X \frac{\exp(w^T F(X))}{Z(w)} F_f(X) \\ &= F_f(D) - \sum_X p(X) F_f(X) \\ &= F_f(D) - \mathbb{E}_X[F_f(X)]. \end{aligned}$$

- Derivative of $\log(Z)$ is marginal of feature.

(inference required for learning)

- $\nabla_f f(w) = 0$ means sufficient statistics match in model and data.

Optimization with MRFs

- With log-linear parameterization, NLL takes the form

$$f(w) = -w^T F(D) + \log Z(w).$$

- Gradient with respect to parameter f is given by

$$\begin{aligned} -\nabla_f f(w) &= F_f(D) - \sum_X \frac{\exp(w^T F(X))}{Z(w)} F_f(X) \\ &= F_f(D) - \sum_X p(X) F_f(X) \\ &= F_f(D) - \mathbb{E}_X[F_f(X)]. \end{aligned}$$

- Derivative of $\log(Z)$ is marginal of feature.

(inference required for learning)

- $\nabla_f f(w) = 0$ means sufficient statistics match in model and data.
- ML is maximum entropy distribution with these statistics.

Optimization with MRFs

- With log-linear parameterization, NLL takes the form

$$f(w) = -w^T F(D) + \log Z(w).$$

- Gradient with respect to parameter f is given by

$$\begin{aligned}
-\nabla_f f(w) &= F_f(D) - \sum_X \frac{\exp(w^T F(X))}{Z(w)} F_f(X) \\
&= F_f(D) - \sum_X p(X) F_f(X) \\
&= F_f(D) - \mathbb{E}_X[F_f(X)].
\end{aligned}$$

- Derivative of $\log(Z)$ is marginal of feature.

  (inference required for learning)

- $\nabla_f f(w) = 0$ means sufficient statistics match in model and data.
- ML is maximum entropy distribution with these statistics.
- Typical solvers: L-BFGS, IPF (coordinate descent), closed form (decomposable), proximal Newton (constraints/non-smooth).

Learning for Structured Prediction

3 types of classifiers discussed in CPSC 540:

Setting	Generative Model	Discriminative Model	Discriminant Function
“Classic ML”	Naive Bayes, GDA	Logistic Regression	SVM

Learning for Structured Prediction

3 types of classifiers discussed in CPSC 540:

Setting	Generative Model	Discriminative Model	Discriminant Function
"Classic ML"	Naive Bayes, GDA	Logistic Regression	SVM
Struct. Pred.	MRF	CRF	SSVM

Learning for Structured Prediction

3 types of classifiers discussed in CPSC 540:

Setting	Generative Model	Discriminative Model	Discriminant Function
"Classic ML"	Naive Bayes, GDA	Logistic Regression	SVM
Struct. Pred.	MRF	CRF	SSVM

We'll discuss MRFs and CRFs today, SSVMs in week 3.

Review of Generative Models for Classification

First let's consider generative models for classification:

- To model $p(y|X)$, generative models use Bayes rule:

$$
\begin{aligned}
p(y|X) &\propto p(y, X) \\
&= p(X|y)p(y).
\end{aligned}
$$

Review of Generative Models for Classification

First let's consider generative models for classification:

- To model $p(y|X)$, generative models use Bayes rule:

$$\begin{aligned} p(y|X) &\propto p(y,X) \\ &= p(X|y)p(y). \end{aligned}$$

- Estimating $p(y)$ is easy: count the number of times $y_i = y$.

Review of Generative Models for Classification

First let's consider generative models for classification:

- To model $p(y|X)$, generative models use Bayes rule:

$$\begin{aligned} p(y|X) &\propto p(y,X) \\ &= p(X|y)p(y). \end{aligned}$$

- Estimating $p(y)$ is easy: count the number of times $y_i = y$.
- Estimating $p(X|y)$ not easy: features might be really complicated.
- Typical solutions:
    - Naive Bayes: $p(X|y) \approx \prod_{i=1}^{n} p(x_i|y)$.
    - Gaussian discriminant analysis: $p(X|y) \sim \mathcal{N}(\mu_y, \Sigma_y)$.

Review of Generative Models for Classification

First let's consider generative models for classification:

- To model $p(y|X)$, generative models use Bayes rule:

$$\begin{aligned} p(y|X) &\propto p(y, X) \\ &= p(X|y)p(y). \end{aligned}$$

- Estimating $p(y)$ is easy: count the number of times $y_i = y$.
- Estimating $p(X|y)$ not easy: features might be really complicated.
- Typical solutions:
  - Naive Bayes: $p(X|y) \approx \prod_{i=1}^{n} p(x_i|y)$.
  - Gaussian discriminant analysis: $p(X|y) \sim \mathcal{N}(\mu_y, \Sigma_y)$.
- More exotic:
  - Bayesian network classifiers.
  - Mixture models.
  - Kernel density estimation.
  - Fit an MRF.

20 Newsgroups Example

20 newsgroups data:

features X	class y
files, mac, pc	computer
hockey, league, win	sports
car, drive	automotive
drive, files, mac	computer
files, pc, win	computer

20 Newsgroups Example

20 newsgroups data:

features X	class y
files, mac, pc	computer
hockey, league, win	sports
car, drive	automotive
drive, files, mac	computer
files, pc, win	computer

To find p(computer | files, pc, win), compute

- p(computer),
- p(files, pc, win | computer).

Generative Models for Structured Prediction

Now let's consider generative models for structured prediction:

- To model $p(Y|X)$, generative models use Bayes rule:

$$\begin{aligned} p(Y|X) &\propto p(Y,X) \\ &= p(X|Y)p(Y). \end{aligned}$$

Generative Models for Structured Prediction

Now let's consider generative models for structured prediction:

- To model $p(Y|X)$, generative models use Bayes rule:

$$
\begin{aligned}
p(Y|X) &\propto p(Y, X) \\
&= p(X|Y)p(Y).
\end{aligned}
$$

- Estimating $p(Y)$ is harder: fit an MRF.

Generative Models for Structured Prediction

Now let's consider generative models for structured prediction:

- To model $p(Y|X)$, generative models use Bayes rule:

$$\begin{aligned} p(Y|X) &\propto p(Y,X) \\ &= p(X|Y)p(Y). \end{aligned}$$

- Estimating $p(Y)$ is harder: fit an MRF.
- Estimating $p(X|Y)$ is much harder:
  - Need a model of features for each possible output object.

Generative Models for Structured Prediction

Now let's consider generative models for structured prediction:

- To model $p(Y|X)$, generative models use Bayes rule:

$$\begin{aligned} p(Y|X) &\propto p(Y,X) \\ &= p(X|Y)p(Y). \end{aligned}$$

- Estimating $p(Y)$ is harder: fit an MRF.
- Estimating $p(X|Y)$ is much harder:
  - Need a model of features for each possible output object.
- Typical solution:
  - Doubly-naive Bayes: $p(X|Y) \approx \prod_{i=1}^n p(x_i|Y) \approx \prod_{i=1}^n p(x_i|y_i)$.
    - maybe with assumption on $p(x_i|y_i)$ (naive Bayes, Gaussian, etc.).
  - Assume features x_i generated independently from part y_i.

Generative Models for Structured Prediction

Now let's consider generative models for structured prediction:

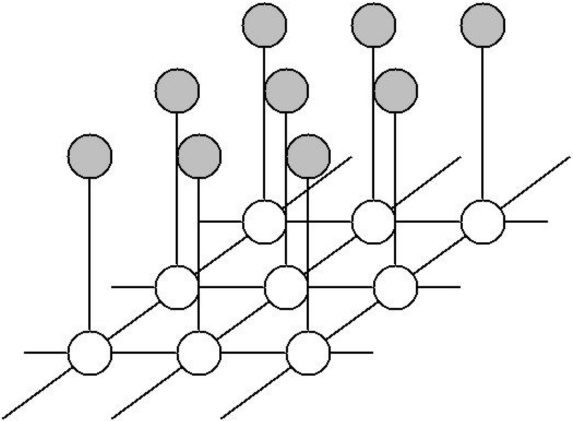
- To model $p(Y|X)$, generative models use Bayes rule:

$$\begin{aligned} p(Y|X) &\propto p(Y, X) \\ &= p(X|Y)p(Y). \end{aligned}$$

- Estimating $p(Y)$ is harder: fit an MRF.
- Estimating $p(X|Y)$ is much harder:
  - Need a model of features for each possible output object.
- Typical solution:
  - Doubly-naive Bayes: $p(X|Y) \approx \prod_{i=1}^n p(x_i|Y) \approx \prod_{i=1}^n p(x_i|y_i)$.
    - maybe with assumption on $p(x_i|y_i)$ (naive Bayes, Gaussian, etc.).
  - Assume features x_i generated independently from part y_i.
- Alternatives:
  - directly model $p(Y, X)$ as an MRF.
  - treat $p(X|Y)$ as a structured prediction problem.

Image Segmentation Example

Naive Bayes across space:



Given labels, features generated independently across space.
(possible naive Bayes assumption about features at same location)

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0809/ORCHARD

Review of Discriminative Models for Classification

- Conditional models for classification directly model $p(y|X)$.
  - No need to model features X given each y.
- Canonical example is logistic regression:

$$p(y=+1|X) = \frac{1}{1+\exp(-yw^TX)} = \frac{\phi(+1)}{\phi(+1)+\phi(-1)}.$$

Review of Discriminative Models for Classification

- Conditional models for classification directly model $p(y|X)$.
  - No need to model features X given each y.
- Canonical example is logistic regression:

$$p(y=+1|X) = \frac{1}{1+\exp(-yw^TX)} = \frac{\phi(+1)}{\phi(+1)+\phi(-1)}.$$

$$\begin{aligned} p(y=-1|X) &= 1 - p(y=+1|X) = 1 - \frac{1}{1+\exp(-yw^TX)} \\ &= \frac{\exp(-yw^TX)}{1+\exp(-yw^TX)} = \frac{\phi(-1)}{\phi(+1)+\phi(-1)}. \end{aligned}$$

Review of Discriminative Models for Classification

- Conditional models for classification directly model $p(y|X)$.
  - No need to model features X given each y.
- Canonical example is logistic regression:

$$p(y=+1|X) = \frac{1}{1+\exp(-yw^TX)} = \frac{\phi(+1)}{\phi(+1)+\phi(-1)}.$$

$$\begin{aligned} p(y=-1|X) = 1 - p(y=+1|X) &= 1 - \frac{1}{1+\exp(-yw^TX)} \\ &= \frac{\exp(-yw^TX)}{1+\exp(-yw^TX)} = \frac{\phi(-1)}{\phi(+1)+\phi(-1)}. \end{aligned}$$

- This is a conditional UGM with:

$$m(1, X_{ij}, y_i=+1) = 0, \quad m(1, X_{ij}, y_i=-1) = j.$$

- Generalization of this is conditional random fields (CRFs).

Log-Linear CRF Parameterization

- The log-linear generalization for CRFs is given by

$$\phi_i(y_i) = \exp\left(\sum_f w_{m(i,y_i,f)} x_{i,f}\right),$$

and similarly for edges ($E(Y|X)$ is linear, NLL is convex).

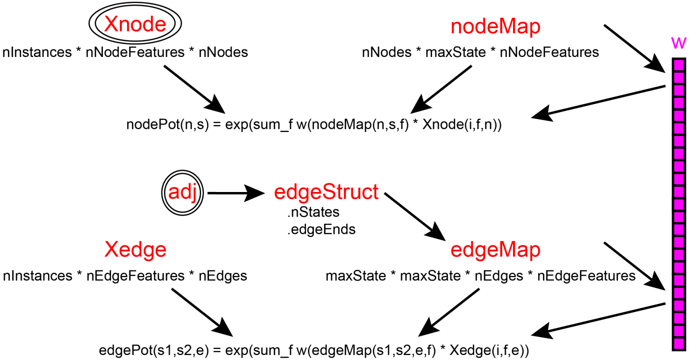
Log-Linear CRF Parameterization

- The log-linear generalization for CRFs is given by

$$\phi_i(y_i) = \exp\left(\sum_f w_{m(i,y_i,f)} x_{i,f}\right),$$

  and similarly for edges ($E(Y|X)$ is linear, NLL is convex).

- How this works in UGM software:



Conditional Random Fields (CRFs)

- CRFs directly model $p(Y|X)$ for structured prediction

$$p(Y|X) = \frac{\exp(w^T F(Y,X))}{Z(w,X)}.$$

- Much simpler than generative approach:

Conditional Random Fields (CRFs)

- CRFs directly model $p(Y|X)$ for structured prediction

$$p(Y|X) = \frac{\exp(w^T F(Y,X))}{Z(w,X)}.$$

- Much simpler than generative approach:
  - No need to model features x for each possible object y.
  - > 8000 citations since 2001.
- For pairwise UGMs, features have form $F(y_i, X)$ or $F(y_i, y_j, X)$.

Conditional Random Fields (CRFs)

- CRFs directly model $p(Y|X)$ for structured prediction

$$p(Y|X) = \frac{\exp(w^T F(Y,X))}{Z(w,X)}.$$

- Much simpler than generative approach:
  - No need to model features x for each possible object y.
  - > 8000 citations since 2001.
- For pairwise UGMs, features have form $F(y_i, X)$ or $F(y_i, y_j, X)$.
- NLL and its gradient have similar form to MRF

$$f(w) = -\frac{1}{n}\sum_{i=1}^{n} -w^T F(Y_i, X_i) + \log(Z(w, X_i)),$$

$$\nabla_f f(w) = -\frac{1}{n}\sum_{i=1}^{n} F(Y_i, X_i) + \mathbb{E}_{Y|X}[F_f(Y_i, X_i)],$$

  but partition function and marginals for each example i.
- Maintains maximum entropy interpretation.

Fitting CRFs

Solvers for fitting parameters of CRFs:

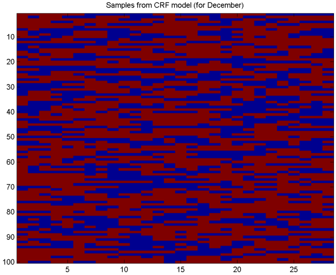
- L-BFGS as in MRFs.
- Stochastic gradient (only 1 partition function per iteration).
- Non-uniform SAG: same cost as stochastic gradient, faster convergence rate, but requires storing marginals.
- Non-uniform SVRG? (similar to SAG, but without the memory)

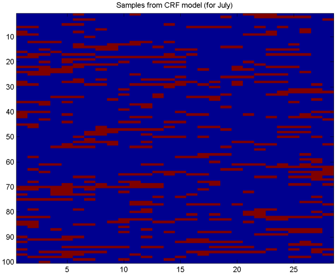
Rain Demo with Month Data

- Let's add a month variable to rain data:
  - Fit a CRF of p(rain | month).
  - Use 12 binary indicator features giving month.
  - NLL goes from 16.8 to 16.2.

Rain Demo with Month Data

- Let's add a month variable to rain data:
  - Fit a CRF of p(rain | month).
  - Use 12 binary indicator features giving month.
  - NLL goes from 16.8 to 16.2.
- Samples of rain data conditioned on December and July:





Approximate Learning

- Inference is a sub-routine of learning:
  - We can only learn when inference is tractable.

Approximate Learning

- Inference is a sub-routine of learning:
  - We can only learn when inference is tractable.
- Strategies when inference is not tractable:
  - Change the objective function:
    - Pseudo-likelihood (fast, convex, and crude):

$$\log p(Y|X) \approx \sum_i \log p(y_i|y_{-i}, X),$$

      transforms learning into logistic regression on each part.
    - SSVMs have decoding as a sub-routine (week 3).

Approximate Learning

- Inference is a sub-routine of learning:
  - We can only learn when inference is tractable.
- Strategies when inference is not tractable:
  - Change the objective function:
    - Pseudo-likelihood (fast, convex, and crude):

      $$\log p(Y|X) \approx \sum_i \log p(y_i|y_{-i}, X),$$

      transforms learning into logistic regression on each part.
    - SSVMs have decoding as a sub-routine (week 3).
  - Use approximate inference (weeks 2-3):
    - Local search.
    - Variational methods.
    - Monte Carlo methods.
    - Convex relaxations.

Homework: TrainMRF, TrainCRF, and ICM

Homework: TrainCRF, ICM, and Block ICM part of the Block.

TrainMRF UGM Demo

Up to this point, we have focused on the tasks of decoding/inference/sampling, given known potential functions. That is, we have assumed that the model is known, and/or made up a story to justify our choice of potential functions. We now turn to the task of *parameter estimation*. In parameter estimation, we are given data (and a graph structure), and we want to find the 'best' potential functions for modeling the data. For example, we might want to find the potential functions that maximize the likelihood of the data. Once we have estimated a good set of potential functions, we can then use these potentials with the techniques discussed in the previous demos for decoding/inference/sampling.

Typically, using data to estimate good potential functions will lead to a much better model than if we tuned the potential functions manually. This is true for several reasons, but one of the main reasons is simply that in most models it is difficult to describe what the values of the potential functions mean in probabilistic terms (of course, there are some exceptions like Markov chains). However, parameter estimation will generally be harder than decoding/inference/sampling with known parameters. Indeed, most parameter estimation methods will need to use decoding/inference/sampling as a sub-routine. Further, we also must address the modeling issue of tieing/parameterizing the potential functions.

Vancouver Rain Data

Our first example of parameter estimation we will consider a simple model of rainfall in Vancouver. I extraced the "daily precipitation" amount for the Steveston weather station (which is close to the Vancouver airport) from the Canadian daily climate archive (CDCA) from Canada's National Climate Archive. This archive contained data for this weather station from 1896-2004, and I made a simple binary data set out of the available data as follows: I treated each month as a sample, and concentrated on the first 28 days of the month so that all samples would be the same length. I also removed the months with missing or accumulated values, and binarized the data set by giving it the state '0' if there was no daily precipitation (or trace amounts), and '1' if there was non-zero daily precipitation. After removing the missing months, we are left with a data set containing 1059 months (samples) with 28 days (variables).

We can load this data set into a variable *y* using:

```
load rain.mat
y = int32(X+1); % Convert from {0,1} double to {1,2} int32 representation
```

TrainCRF UGM Demo

The MRF model we used in the last demo was fairly naive, and it can be improved in various ways. As one example, in this demo we will take into account the month of the year. One way to do this would be add a 12-state 'month' variable to the model that is connected to every node. We could then do parameter estimation in the model by cutset conditioning (i.e. we condition on the month variable so that the remaining variables form a chain). We could subsequently do conditional decoding/inference/sampling in the model, where we condition on the month.

If we are only interested in conditional queries, then a conditional random field (CRF) might be a better option. In CRFs, we have two types of variables: (i) the 'features' (also known as 'covariates') *X* are treated as fixed non-random variables, and (ii) the 'labels' *y* are treated as random variables in a UGM, where the parameters of the UGM depend on the features. In our case, the 'labels' will be the 28 variables representing days of the month, and the features will consist of a 'bias' feature and a set of indicator variables that will represent the month. That is, each possible value that the features can take leads to a different UGM model of the labels.

Vancouver Rain Data, with Months

In addition to the matrix of binarized daily precipitation values, the file 'rain.mat' also contains a vector *months* that contains a number in the range 1-12, representing the month of the year. We might hope to get a better model by using this data, since common sense would indicate that it is more likely to rain in December than July. Of course, since we have the relevant data we don't need to just assume that our common sense is correct; we can use the data to find a good way to incorporate the month information. In particular, we are going to use a CRF where the months are used to define the features, and the binarized daily precipitation values represent the labels. As before, the first step will be loading the data and making the edgeStruct:

```
load rain.mat
y = int32(X+1);
[nInstances,nNodes] = size(y);
nStates = max(y);
adj = zeros(nNodes);
for i = 1:nNodes-1
    adj(i,i+1) = 1;
end
adj = adj+adj';
edgeStruct = UGM_makeEdgeStruct(adj,nStates);
maxState = max(nStates);
```

Note that we have used *y* to denote the samples of the random variables in this demo, since *X* is typically used to denote the features in a CRF model.

Representing the Features

To train CRFs, we need to make the arrays *Xnode* and *Xedge*. *Xnode* represents features that affect the node potentials, while *Xedge* represents features that

ICM UGM Demo

The last demo ends the first series of demos coverings exact methods for decoding/inference/sampling. We now turn to the case of approximate methods. These methods can be applied to models with general graph structures and edge potentials, but don't necessarily perform these operations exactly. This first demo discusses approximate decoding with local search, generalizing the iterated conditional mode (ICM) algorithm mentioned in the previous demo.

Iterated Conditional Modes

The ICM algorithm is one of the simplest methods for optimal decoding. In the ICM algorithm, we initialize the nodes to some starting state values (by default, UGM uses the states that maximize the node potentials), and we then start cycling through the nodes in order. When we get to node *i*, we consider all states that node *i* could take, and replace its current state with the state that maximizes the joint potential. We keep cycling through the nodes in order until we complete a full cycle without changing any nodes. At this point, we have reached a local optima of the joint potential that can not be improved by changing the state of any single node. The ICM algorithm is described in Besag's paper on analyzing "dirty pictures":

- J.E. Besag. **On the Statistical Analysis of Dirty Pictures**. Journal of the Royal Statistical Society Series B, 1986.

In UGM, we can apply ICM using:

```
getNoisyX
ICMDecoding = UGM_Decode_ICM(nodePot,edgePot,edgeStruct);
figure;
imagesc(reshape(ICMDecoding,nRows,nCols));
colormap gray
```

This function UGM_Decode_ICM can also take an optional fourth argument that gives the intial configuration.

Greedy Local Search Decoding

The ICM algorithm is a specific instance of a *local search* discrete optimization algorithm, sometimes described as a *first improvement* greedy algorithm. Instead of this particular local search method, we could instead consider other local search methods.

As one example, we could consider a *best improvement* greedy algorithm, where we search for the single state change that improves the joint potential by the largest amount. This is slightly more expensive than ICM since we only make a single state change after cycling through the nodes, but it ensures that we always take the 'best' move at each iteration. In UGM we can apply this greedy algorithm using:

```
greedyDecoding = UGM_Decode_Greedy(nodePot,edgePot,edgeStruct);
figure;
imagesc(reshape(greedyDecoding,nRows,nCols));
```

Review of Topics

- Can use maximum likelihood to fit potentials given data.
- Log-linear parameterization has nice properties (e.g., convexity).
- Parameter tieing allows sharing of statistical strength.
- Fitting MRFs requires sufficient statistics and inference.
- Generalization of logistic regression is CRFs, which are more expensive but allow conditioning on arbitrary features.