Lecture 3

Note Title 1/18/2011

In the first lecture we discussed the task of labeling pixels – either as edge or non-edge
or as sky, road, vegetation, etc.

We made a restrictive assumption, that we could model each pixel independently.

Suppose $\underline{w} = \{w_i : i \in \Lambda\}$ is the set of labels for each image pixel (Λ = set of all pixels).

Then $P(\underline{w}|\underline{I}) = \prod_i P(w_i|\underline{I})$

factorization
conditional independence

This assumption is clearly incorrect for real images – but it is a useful approximation because it simplifies inference and learning.

Inference is simplified because to find
$\hat{\underline{w}} = \arg\max_{\underline{w}} P(\underline{w}|\underline{I})$
we simply need to solve $\hat{w}_i = \arg\max_{w_i} P(w_i|\underline{I})$ for each i

Note: suppose that each pixel w_i can take k labels. Then the total number of labels for the entire image is $k^{|\Lambda|}$ → exponentially big. (set of all possible $\underline{w}$)

Searching through this space of all possible labels is very hard in general.

There are exceptional cases where it is possible: depending on the form of $P(\underline{w}|\underline{I})$
→ we may be able to use dynamic programming, steepest descent, or graph cuts. (discussed later)

If we can factorize $P(\underline{w}|\underline{I}) = \prod_i P(w_i|\underline{I})$
then we can determine $\hat{\underline{w}}$ in at most $|\Lambda| k$ operations

Learning will also be simplified in this case. The number of training examples will be small.

(2) Now let's consider the more general case.

We can express $P(w|I,\lambda) = \frac{1}{Z(\lambda,I)} e^{\lambda \cdot \phi(w,I)}$

This is an exponential distribution.

The ϕ's are feature functions of w & I (examples later) and the λ's are parameters of the model.

$$\lambda \cdot \phi(w,I) = \sum_{\mu} \lambda_\mu \phi_\mu(w,I)$$

Assume that we are given a set of images with the value of w specified $\{(w_a, I_a) : a \in A\}$

Learning can be formulated as finding the values of the parameters λ that maximize:

$$\prod_{a \in A} P(w_a | I_a, \lambda)$$

Note: this is supervised learning. We assume that the potentials $\phi_\mu(w,I)$ are known, but their parameters λ_μ are not. We will see other forms of learning in this course.

Note: this is a form of statistical regression.

Reformulate the problem in terms of minimization

$$\hat{\lambda} = \arg\min_{\lambda} \left\{ -\sum_{a \in A} \log P(w_a | I_a, \lambda) \right\}.$$

Properties: (i) this is a convex optimization problem, so there is a unique solution and algorithms which are guaranteed to find it.

(ii) the solution $\hat{\lambda}$ obeys.

$$\sum_{a \in A} \sum_{w} P(w | I_a, \hat{\lambda}) \phi(w, I_a) = \sum_{a \in A} \phi(w_a, I_a)$$

expected statistics of model = observed statistics of data.

But there is a problem. The learning algorithm may require an exponential number of operations.

(3) Justify these two properties.

$$P(\underline{w}|I,\underline{\lambda}) = \frac{1}{Z[\underline{\lambda},I]} e^{\underline{\lambda}\cdot\underline{\varphi}(\underline{w},I)}$$

$$-\sum_{a\in A} \log P(w_a|I_a,\underline{\lambda}) = -\sum_{a\in A} \underline{\lambda}\cdot\underline{\varphi}(w_a,I_a) + \sum_a \log Z[\underline{\lambda},I_a]$$

$$Z[\underline{\lambda},I_a] = \sum_{\underline{w}} e^{\underline{\lambda}\cdot\underline{\varphi}(\underline{w},I_a)}$$

(to ensure that $\sum_{\underline{w}} P(\underline{w}|I,\underline{\lambda}) = 1$)

The function $\log Z[\underline{\lambda},I]$ has two important properties:

(a) $\frac{\partial}{\partial \underline{\lambda}} \log Z[\underline{\lambda},I] = \sum_{\underline{w}} \underline{\varphi}(\underline{w},I)\, e^{\underline{\lambda}\cdot\underline{\varphi}(\underline{w},I)} = \sum_{\underline{w}} \underline{\varphi}(\underline{w},I)\, P(\underline{w}|I,\underline{\lambda})$

(b) $\frac{\partial^2}{\partial \underline{\lambda}\partial\underline{\lambda}} \log Z[\underline{\lambda},I]$ is positive semi-definite (proof by Cauchy-Schwarz inequality)

Property (b) implies that $-\sum_{a\in A} \log P(w_a|I_a,\underline{\lambda})$ is convex and hence has, at most, one minimum

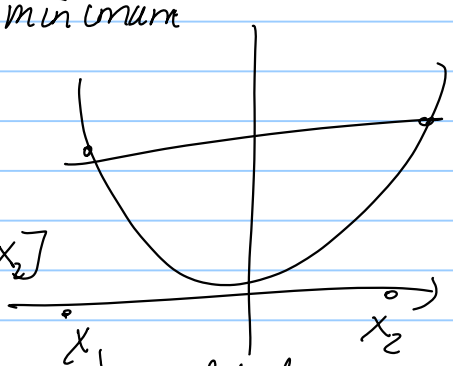
Convexity

$\forall\ X_1, X_2$

$\forall\ \alpha\in[0,1]$

$$\alpha E(X_1) + (1-\alpha) E(X_2) \geq E[\alpha X_1 + (1-\alpha) X_2]$$

equivalent to the Hessian $\frac{\partial^2 E}{\partial x \partial x}$ being positive semi-definite.



The function is bounded below by 0, so there is always one minimum.

Property (a) shows that the solution $\hat{\underline{\lambda}}$

$$\frac{\partial}{\partial \underline{\lambda}} \left\{ -\sum_{a\in A} \log P(w_a|I_a,\underline{\lambda}) \right\} = 0$$

obeys

$$\sum_{a\in A} \underline{\phi}(w_a,I_a) = \sum_{a\in A} \underline{\phi}(\underline{w},I_a)\, P(\underline{w}|I_a,\hat{\underline{\lambda}})$$

So far, everything looks good. But now we have to consider the computational complexity.

(4) Learning Algorithms:

There are a variety of learning algorithms that can converge to the solution $\hat{\lambda}$.

(1) Steepest Descent in $-\sum_{a\in A} \log P(w_a|I_a,\lambda)$

$$\lambda^{t+1} = \lambda^t - \epsilon \frac{\partial}{\partial \lambda}\left\{-\sum_{a\in A}\log P(w_a|I_a,\lambda)\right\}_{\lambda^t}$$

the derivative evaluated at λ^t.

i.e. $$\lambda^{t+1} = \lambda^t - \epsilon \sum_{a\in A} \phi(w_a,I_a) + \epsilon \sum_{a\in A}\sum_{w} \phi(w,I_a) P(w|I_a,\lambda^t)$$

This converges when $\sum_{a\in A} \phi(w_a,I_a) = \sum_{a\in A}\sum_{w} \phi(w,I_a) P(w|I_a,\lambda)$.
(ϵ is time step)

(2) Generalized Iterative Scaling.

$$\lambda^{t+1} = \lambda^t - \log \sum_{a\in A} \phi(w_a,I_a) + \log \sum_{a\in A}\sum_{w} \phi(w,I_a) P(w|I_a,\lambda^t)$$

Discrete Iterative Algorithm — same fixed point → note, no need of a time step ϵ.

Important → both methods require computing

$\sum_{a\in A} \phi(w_a,I_a)$ → which is practical.

$\sum_{a\in A}\sum_{w} \phi(w,I_a) P(w|I_a,\lambda)$ → which is often impossible

(summation over $k^{|\Lambda|}$ possible states)

Two Problems: (i) normalizing to compute $Z[\lambda] = \sum_{w} e^{\lambda\cdot\phi(w)}$

(ii) computing the expectation $\sum_{w} \phi(w,I) P(w|I,\lambda)$.

Again → for some exceptional distributions these quantities can be computed (e.g. by Dynamic Programming)

But in general they cannot be.

What can we do? One possibility is Markov Chain Monte Carlo (MCMC). If we can sample efficiently from $P(w|I,\lambda)$ then we compute the expectations (sampling does not require that we know $Z[\lambda]$.)

(5.) In standard machine learning (e.g. Stat231) we have to deal with data complexity – i.e., limited amount of data available to learn complex models.

Now, we are also faced with computational complexity – we need inference algorithms to perform tasks like:

estimate $\hat{\underline{w}} = \arg\max_{\underline{w}} P(\underline{w} \mid \underline{I}, \underline{\lambda})$

compute $\sum_{\underline{w}} \underline{\phi}(\underline{w}, \underline{I})\, P(\underline{w} \mid \underline{I}, \underline{\lambda})$

Both are possible if $P(\underline{w} \mid \underline{I}, \underline{\lambda})$ takes restricted forms → e.g. distributions on graphical models without closed loops (then dynamic programming can be used).

Otherwise, we will need to use stochastic methods (MCMC) or approximate algorithms.

Note: stochastic methods are very general purpose, but they are a strategy not an algorithm. Usually a lot of problem specific knowledge is required to design an MCMC to make it computationally efficient.

How do these algorithms simplify if $\underline{w} = \{w_i : i \in \Lambda\}$, and $P(\underline{w} \mid \underline{I}) = \prod_i P(w_i \mid \underline{I})$?

In this case, we need only learn

$$P(w_i \mid \underline{I}, \underline{\lambda}) = \frac{1}{Z(\underline{\lambda}, w_i)} e^{\underline{\lambda} \cdot \underline{\phi}(w_i, \underline{I})}$$

when each w_i takes only k-values

Inference is easy → $\hat{w}_i$ is computed by exhaustive search over k-possible values of w_i.

$\sum_{w_i} \underline{\phi}(w_i, \underline{I})\, P(w_i \mid \underline{I}, \underline{\lambda})$ requires k computations only.

(6) Example: suppose $w \in \{-1, 1\}$

Define potential function $\phi_\mu(w, I) = w \phi_\mu(I)$

$$P(w|I,\lambda) = \frac{1}{Z[I,\lambda]} e^{w \sum_\mu \lambda_\mu \phi_\mu(I)}$$

Learning algorithms as before. Maximize

$$\prod_\mu P(w^\mu | I^\mu, \underline{\lambda}) \quad \text{w.r.t. } \underline{\lambda}$$

The only difference is that the computations

$$\hat{w} = \text{ARG MAX}_w P(w|I,\lambda)$$

and $\sum_w w \phi_\mu(I) P(w|I,\lambda)$ can be performed simply, because w takes only two possible values

Relationship to AdaBoost.

AdaBoost seeks to learn a strong classifier

$$H(I) = \text{sign}(w \sum_\mu \lambda_\mu \phi_\mu(I))$$

by selecting and combining weak classifiers

$$\{\phi_\mu(I) : \mu \in \Lambda\}$$

with condition $\phi_\mu(I) \in \{\pm 1\}$

AdaBoost can be formalized mathematically as "coordinate descent" on a function

$$F(\lambda) = \sum_{a \in A} e^{-w_a \sum_\mu \lambda_\mu \phi_\mu(I)}$$

(note sign change in exponential)

Initialize: $\underline{\lambda} = 0$

At iteration step t

current state $\{\lambda_\mu^t\}$

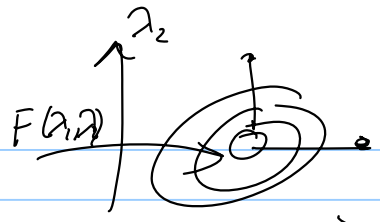
For each μ, solve for Δ_μ s.t. $\frac{\partial F}{\partial \lambda_\mu}(\lambda_\mu^t + \Delta_\mu) = 0$ (1)

(i.e. minimize w.r.t. λ_μ keeping the other λ's fixed).

calculate $F_\mu^t = F(\lambda_1^t, \ldots, \lambda_\mu^t + \Delta_\mu, \ldots \lambda_\nu^t, \ldots)$ (2)

Calculate $\hat{\mu} = \text{ARG MIN}_\mu F_\mu^t$

set $\lambda_\mu^{t+1} = \lambda_\mu^t, \ \mu \neq \hat{\mu}, \quad \lambda_{\hat{\mu}}^{t+1} = \lambda_{\hat{\mu}}^t + \Delta_{\hat{\mu}}$



Geometrically,

At time t, select the direction $\lambda_{\hat{\mu}}$ that gives maximum decrease in $F(\lambda)$. (while not changing the other λ_μ, $\mu \neq \hat{\mu}$)

Selection & Weighting:

Recall that we initialized the process so that $\lambda_\mu^{t=0} = 0$, $\forall \mu$.

Suppose we select $\lambda_{\hat{\mu}}$ at time $t+1$, then there are two cases to consider:

(i) if $\lambda_{\hat{\mu}}^t = 0$, then the weak classifier $\phi_{\hat{\mu}}(I)$ has not been used and is now selected $\lambda_{\hat{\mu}}^{t+1} \neq 0$. and we are assigning it a weight $\Delta_{\hat{\mu}}$.

(ii) if $\lambda_{\hat{\mu}}^t \neq 0$, then we have already selected the weak classifier $\phi_{\hat{\mu}}(I)$ and so we are simply changing its weight. $\lambda_{\hat{\mu}}^t \rightarrow \lambda_{\hat{\mu}}^t + \Delta_{\hat{\mu}}$.

Hence selection and weighting are performed by the same process — coordinate descent on $F[\lambda]$

An important point to realize is that because of the form of $F[\lambda]$ it is possible to perform all the learning calculations ①②③ analytically.

This reduces to the standard equations of AdaBoost.

Note: AdaBoost minimizes $F[\lambda]$, while conventional methods maximize $\prod_{a \in A} P(W_a(I_a,\lambda))$ — which is computationally harder.

But. it can be shown that both methods are asymptotically equivalent (when there is infinite data). and that $F[\lambda]$ & $\prod_{a \in A} P(W_a|I_a,\lambda)$ are related by bounds.

Note: bounds will be discussed later in the course.