CSE 559A: Computer Vision

Fall 2020: T-R: 11:30-12:50pm @ Zoom

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Adith Boloor, Patrick Williams

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Nov 12, 2020
GROUPING & SEGMENTATION

Graph-based Methods

\[ L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n,n') \in \mathcal{E}} S_{n,n'}(L[n], L[n']) \]
Graph-based Methods

\[ L = \arg \min_{L[n] \in \{0,1\}} \ C[n, L[n]] + \sum_{(n,n') \in E} S_{n,n'}(L[n], L[n']) \]
Graph-based Methods

$L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n, n') \in E} S_{n,n'}(L[n], L[n'])$

Break set of vertices into two disconnected graphs by removing edges.
Cost = sum of weight on edges you removed
Formally, let’s say our smoothness cost $S_{n,n'}(l, l') = w_{n,n'} \delta[l! = l']$, for $w_{n,n'} \geq 0$.

$$L = \arg \min_{L[n] \in \{0,1\}} \sum_n C[n, L[n]] + \sum_{(n,n') \in E} w_{n,n'} \delta[L[n]! = L[n']]$$

- Build a graph with vertices $V = \{n\} \cup \{0, 1\}$.
- Place an edge between every $(n, n') \in E$ with weight $w_{n,n'}$.
- Place an edge between $(n, 0) \forall n$ with weight $C[n, 1]$ (assuming costs are positive).
- Place an edge between $(n, 1) \forall n$ with weight $C[n, 0]$ (assuming costs are positive).
- Partition the vertices into sets $A, B$ such that $0 \in A, 1 \in B$, to minimize $\text{Cut}(A, B)$.
  - The cut is defined as the sum of the weights of the edges going between vertices in $A$ to vertices in $B$.
- Can be solved in polynomial time (e.g., Stoer-Wagner)
- Assign all pixels in $A$ label 0, and all pixels in $B$ label 1.
GROUPING & SEGMENTATION

Graph-based Methods

\[
L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n, n') \in \mathcal{E}} S_{n, n'}(L[n], L[n'])
\]
GROUPING & SEGMENTATION

Graph-based Methods

\[
L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n,n') \in E} S_{n,n'}(L[n], L[n'])
\]

- Initialize unary costs \( C \) with user labels. Do a segmentation.
GROUPING & SEGMENTATION

Graph-based Methods

\[ L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n, n') \in E} S_{n,n'}(L[n], L[n']) \]

- Initialize unary costs \( C \) with user labels. Do a segmentation.
- Now look at the foreground and background pixels. Fit a probability distribution to each (mixture of Gaussians, histogram)
GROUPING & SEGMENTATION

Graph-based Methods

\[ L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n,n') \in E} S_{n,n'}(L[n], L[n']) \]

- Initialize unary costs \( C \) with user labels. Do a segmentation.
- Now look at the foreground and background pixels. Fit a probability distribution to each (mixture of Gaussians, histogram)
- Update \( C[n,L] \) based on how well the intensity at location \( n \) fits with the foreground/background distributions (for \( L=1/0 \))
Formally, let’s say our smoothness cost $S_{n,n'}(l, l') = w_{n,n'} \delta[l! = l']$, for $w_{n,n'} \geq 0$.

$$L = \operatorname{arg\ min}_{L[n] \in \{0,1\}} \sum_{n} C[n, L[n]] + \sum_{(n,n') \in E} w_{n,n'} \delta[L[n]! = L[n']]$$

- Build a graph with vertices $V = \{n\} \cup \{0, 1\}$.
- Place an edge between every $(n, n') \in E$ with weight $w_{n,n'}$.
- Place an edge between $(n, 0) \forall n$ with weight $C[n, 1]$ (assuming costs are positive).
- Place an edge between $(n, 1) \forall n$ with weight $C[n, 0]$ (assuming costs are positive).
- Partition the vertices into sets $A, B$ such that $0 \in A, 1 \in B$, to minimize $\operatorname{Cut}(A, B)$.
  - The cut is defined as the sum of the weights of the edges going between vertices in $A$ to vertices in $B$.
- Can be solved in polynomial time (e.g., Stoer-Wagner)
- Assign all pixels in $A$ label 0, and all pixels in $B$ label 1.
GROUPING & SEGMENTATION

- Polynomial Time for Binary Segmentation
- NP-hard for multi-label cases. \( L[n] \in \{A, B, C, \ldots \} \)
  - Remember, this is the same as our stereo case.
- But approximate algorithms available
  - Typically different algorithms work well here than for stereo
GROUPING & SEGMENTATION

Multi-label Case: $L[n] = \{A, B, C, \ldots \}$

- Begin with some initial assignment of $L[n]$ (perhaps the pixel-wise minimizer of $C$)
- Then update $L$ by making one of two kinds of moves in each iteration
- $\alpha$-Expansion
  - Choose one of the labels (say $A$)
  - Build a binary segmentation problem where $1 = A$, $0 = \text{everything else}$
  - Set $C[n, 0] = \infty$ for all pixels $n$ where the current label is already $A$
  - Set $C[n, 0] = \text{cost of its current assigned label}$ for every other pixel
  - Set $C[n, 1] = \text{cost of } A \text{ for every other pixel}$
  - Do a min-cut. Replace all pixels labeled 1 with $A$.
- $\alpha - \beta$ Swap
  - Choose a pair of labels (say $A$ and $B$)
  - Now define a new graph, containing only pixels that currently have label $A$ or $B$.
  - Solve the binary segmentation problem
- Iterate through these different kinds of moves for different choices of labels.
GROUPING & SEGMENTATION

References

- Rother et al., GrabCut - Interactive Foreground Extraction using Iterated Graph Cuts, SIGGRAPH 2004.
So far, given an input $X$ and desired output $Y$ we have

- Tried to explain the relationship of how $X$ results from $Y$
  - $X =$ observed image(s) / $Y =$ clean image, sharp image, surface normal, depth
  - Noise, photometry, geometry, …
- Often put a hand-crafted “regularization” cost to compute the inverse
  - Depth maps are smooth
  - Image gradients are small
- But sometimes, there is no way to write-down a relationship between $X$ and $Y$?
  - $X =$ Image, $Y =$ Does the image contain a dog?
- Even if there is, the hand-crafted regularization cost is often arbitrary.
  - Real images contain far more complex and subtle regularity.
Instead, we are going to assume that there is some underlying joint probability distribution $P_{XY}(x, y)$

- And our goal is to compute:
  - The best estimate of $y$ conditioned on a specific value of $x$,

(using small letters for actual values, capitals for “random variables”)
Instead, we are going to assume that there is some underlying joint probability distribution $P_{XY}(x, y)$.

- And our goal is to compute:
  - The best estimate of $y$ conditioned on a specific value of $x$,
  - To minimize some notion of “risk” or “loss”

Define a loss function $L(y, \hat{y})$, which measures how much we dislike $\hat{y}$ as our estimate, when $y$ is the right answer.

**Examples**

- $L(y, \hat{y}) = \|y - \hat{y}\|^2$
- $L(y, \hat{y}) = \|y - \hat{y}\|$
- $L(y, \hat{y}) = 0$ if $y = \hat{y}$, and some $C$ otherwise.
Instead, we are going to assume that there is some underlying joint probability distribution $P_{XY}(x, y)$

- And our goal is to compute:
  - The best estimate of $y$ conditioned on a specific value of $x$,
  - To minimize some notion of “risk” or “loss”

Ideally,

$$\hat{y}(x) = \arg \min_{\hat{y}} \int L(y, \hat{y})P(y|x) \ dy$$

What is $P(y|x)$ in terms of $P_{XY}$?
Instead, we are going to assume that there is some underlying joint probability distribution \( P_{XY}(x, y) \)

- And our goal is to compute:
  - The best estimate of \( y \) conditioned on a specific value of \( x \),
  - To minimize some notion of “risk” or “loss”

Ideally,

\[
\hat{y}(x) = \arg \min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \, dy
\]

\[
P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y') dy'}
\]
\[ \hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y})P(y|x) \, dy \]

\[ P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'} \]

- So we have a loss (depends on the application)
- We can compute \( P(y|x) \) from \( P_{XY} \).
- But we don’t know \( P_{XY} \)!

Assume we are given data!
\[
\hat{y}(x) = \arg \min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \ dy
\]

\[
P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y') dy'}
\]

- So we have a loss (depends on the application)
- We can compute \(P(y|x)\) from \(P_{XY}\).
- But we don’t know \(P_{XY}\)!

Assume we are given as training examples, samples \((x, y) \sim P_{XY}\) from the true joint distribution.
\[ \hat{y}(x) = \arg \min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \, dy \]

\[ P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'} \]

Given \( \{(x_i, y_i)\} \) as samples from \( P_{XY} \), we could:

- Estimate \( P_{XY} \)
\[
\hat{y}(x) = \arg \min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \ dy
\]

\[
P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y') dy'}
\]

Given \{ (x_i, y_i) \} as samples from \( P_{XY} \), we could:

- **Estimate** \( P_{XY} \)
  - Choose parametric form for the joint distribution (Gaussian, Mixture of Gaussians, Bernoulli, \ldots)
  - Estimate the parameters of that parametric form to “best fit” the data.
  - Depending again on some notion of fit (often likelihood)

\[
P_{XY}(x, y) = f(x, y; \theta)
\]

\[
\theta = \arg \max_{\theta} \prod_{i} f(x_i, y_i; \theta) = \arg \max_{\theta} \sum_{i} \log f(x_i, y_i; \theta)
\]

Maximum Likelihood Estimation
\[
\hat{y}(x) = \arg \min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \, dy
\]

\[
P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y') dy'}
\]

\[
P_{XY}(x, y) = f(x, y; \theta)
\]

\[
\theta = \arg \max_{\theta} \sum_i \log f(x_i, y_i; \theta)
\]

So that’s one way of doing things …

- You’re doing a minimization for learning \(P_{XY}\), but then also a minimization at “test time” for every input \(x\).
- You’re approximating \(P_{XY}\) with some choice of the parametric form \(f\).
- And it’s possible that the best \(\theta\) that maximizes likelihood, may not be the best \(\theta\) that minimizes loss.
Given a bunch of samples \( \{(x_i, y_i)\} \) from \( P_{XY} \),
we want to learn a function \( y = f(x) \), such that

given a typical \( x, y \) from \( P_{XY} \), the expected loss \( L(y, f(x)) \) is low.

\[
f = \arg \min_f \left( \int L(y, f(x)) p(y|x) dy \right)
\]
Given a bunch of samples \( \{(x_i, y_i)\} \) from \( P_{XY} \),

we want to learn a function \( y = f(x) \), such that

given a typical \( x, y \) from \( P_{XY} \), the expected loss \( L(y, f(x)) \) is low.

\[
f = \arg\min_f \int \left( \int L(y, f(x)) p(y|x) dy \right) p(x) dx
\]
Given a bunch of samples \( \{(x_i, y_i)\} \) from \( P_{XY} \),

we want to learn a function \( y = f(x) \), such that

given a typical \( x, y \) from \( P_{XY} \), the expected loss \( L(y, f(x)) \) is low.

\[
 f = \arg \min_f \int \int L(y, f(x)) \ p_{XY}(x, y) dx dy
\]

What we’re going to is to replace the double integration with a summation over samples!
Given a bunch of samples \( \{(x_i, y_i)\} \) from \( P_{XY} \),

we want to learn a **function** \( y = f(x) \), such that

given a typical \( x, y \) from \( P_{XY} \), the **expected** loss \( L(y, f(x)) \) is low.

\[
f = \arg \min_{f} \sum_{i} L(y_i, f(x_i))
\]

What we’re going to is to replace the double integration with a summation over samples!

**Empirical Risk** Minimization

- So instead of first fitting the probability distribution from training data, and then
given a new input, minimizing the loss under that distribution …
- We are going to do a search over possible functions that “do well” on the training data,
and assume that a function that minimizes “empirical risk” also minimizes “expected risk”.
Formally

- Given inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we want to learn a function $y = f(x)$, i.e., $f : \mathcal{X} \to \mathcal{Y}$
- Function should be a “good” predictor of $y$, as measured in terms of a risk or loss function: $L(y, \hat{y})$.
- Ideally, we want to find the best $f \in \mathcal{H}$, among some class or space of functions $\mathcal{H}$ (called the hypothesis space), which minimizes the expected loss under the joint distribution $P_{XY}(x, y)$:

$$f = \arg\min_{f \in \mathcal{H}} \int_{\mathcal{X}} \int_{\mathcal{Y}} L(y, f(x)) \ P_{XY}(x, y) \ dy \ dx$$

- But we don’t know this joint distribution, but we have a training set $(x_1, y_1), (x_2, y_2), \ldots (x_T, y_T)$, which (we hope!) are samples from $P_{XY}$.
- So we approximate the integral over the $P_{XY}$ with an average over the training set $(x_t, y_t)$,

$$f = \arg\min_{f \in \mathcal{H}} \frac{1}{T} \sum_{t} L(y_t, f(x_t))$$

You’re given data. Choose a loss function that matches your task, a hypothesis space $\mathcal{H}$, and minimize.
Consider:

- \( x \in \mathcal{X} = \mathbb{R}^d \)
- \( y \in \mathcal{Y} = \mathbb{R} \)
- \( \mathcal{H} \) is the space of all “linear functions” of \( \mathcal{X} \).
  - \( f(x; w, b) = w^T x + b, \quad w \in \mathbb{R}^d, \; b \in \mathbb{R} \)
  - Minimization of \( f \in \mathcal{H} \) becomes a minimization of \( w, b \)
- Consider \( L(y, \hat{y}) = (y - \hat{y})^2 \)

And then we have our familiar least-squares regression!
So we know how to solve this: take derivative and set to 0.

\[ f = \arg \min_{f \in \mathcal{H}} \frac{1}{T} \sum_{t} L(y_t, f(x_t)) \]

\[ w, b = \arg \min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{T} \sum_{t} (y_t - w^T x_t - b)^2 \]

Not just for fitting “lines”. Imagine \( x \) is a vector corresponding to a patch of intensities in a noisy image. \( y \) is corresponding clean intensity of the center pixel. You could use this to “learn” a linear “denoising filter”, by fitting to many examples of pairs of noisy and noise-free intensities.
Now, let’s say we wanted to fit a polynomial instead of a linear function.

For \( x \in \mathbb{R} \),

\[
f(x; w_0, w_1, w_2, \ldots, w_n) = w_0 + w_1 x + w_2 x^2 + \ldots + w_n x^n.
\]

This is our hypothesis space. Same loss function \( L(y, \hat{y}) = (y - \hat{y})^2 \).
Try to work out an expression for $w$. 

$$w = \arg \min_{w \in \mathbb{R}^{n+1}} \frac{1}{T} \sum_{t} (y_t - w_0 - w_1 x_t - w_2 x_t^2 \ldots - w_n x_t^n)^2$$
\[ w = \arg \min_{w \in \mathbb{R}^{n+1}} \frac{1}{T} \sum_{t} (y_t - w_0 - w_1 x_t - w_2 x_t^2 \ldots - w_n x_t^n)^2 \]

Set \( \tilde{x}_t = [1, x_t, x_t^2, x_t^3, \ldots x_t^n]^T \).

And you get exactly the same equation!

\[ w = \left( \sum_t \tilde{x}_t \tilde{x}_t^T \right)^{-1} \left( \sum_t \tilde{x}_t y_t \right) \]

- But now, inverting a larger matrix.
- Can apply the same idea to polynomials of multi-dimensional \( x \).
- Can apply least-squares fitting to any task with an \( L^2 \) loss, and where the hypothesis space is linear in the parameters (not necessarily in the input).

E.g: \( f(x; w_0, w_1, w_2, \ldots, w_n) = w_0 + w_1 x + w_2 x^2 + \ldots w_n x^n \).
Why not just fit the more complex model?
Why not just fit the more complex model?
Why not just fit the more complex model?
Why not just fit the more complex model?
Why not just fit the more complex model?
Why not just fit the more complex model?

nth Order Polynomials can fit all orders up to n.
Why not just fit the more complex model?

nth Order Polynomials can fit all orders up to n.

Why not just choose the most complex inclusive hypothesis space?
Why not just fit the more complex model?

nth Order Polynomials can fit all orders up to n.

Why not just choose the most complex inclusive hypothesis space?
Because, y may have noise (P(y|x) is not deterministic).
MACHINE LEARNING

Why not just fit the more complex model?

nth Order Polynomials can fit all orders upto n.

Why not just choose the most complex inclusive hypothesis space?
Because, y may have noise (P(y|x) is not deterministic).

Given enough capacity, f will hallucinate structure in the noise.
Why not just fit the more complex model?

nth Order Polynomials can fit all orders upto n.

Why not just choose the most complex inclusive hypothesis space?
Because, $y$ may have noise ($P(y|x)$ is not deterministic).
MACHINE LEARNING

Why not just fit the more complex model?

- Linear Fit
- Quadratic Fit

[Image: Flat Earth Society map]

[Image: UFO protest]

[Image: "UFO's are real & the government knows it!"]
Why not just fit the more complex model?

Too simple

Just Right

Too complex

Quadratic Fit
Why not just fit the more complex model?

Too complex

Can be fixed if we had a lot more data.
Why not just fit the more complex model?

Too complex

Can be fixed if we had a lot more data.