GENERAL

- Problem Set 3 Due tonight
  - Please DO NOT忽略 to add solution.pdf to your repo, commit, AND PUSH.
- Problem Set 4 posted and ready to clone!

GROUPING & SEGMENTATION

Formally, let’s say our smoothness cost $S_{n,n'}(l, l') = w_{n,n'} \delta[l \neq l']$, for $w_{n,n'} \geq 0$.

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

- 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 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

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 \( C \rightarrow A \), everything else
    - Set \( C[n, 0] = \infty \) for all pixels \( n \) where the current label is already \( A \)
    - Set \( C[n, 1] = \) cost of its current assigned label for every other pixel
    - Do a min-cut. Replace all pixels labeled \( A \) with \( \beta \).
  - \( \alpha - \beta \) 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 AND SEGMENTATION

Graph Based Approaches

- Define a set of vertices \( V \), edges \( E \) with weights \( w(v_1, v_2) \)
- \( \text{Cut} = \text{partition of vertices into two sets of nodes } A \) and \( B \)
  - \( A \cup B = V \)
  - \( A \cap B = \emptyset \)
- \( \text{Cut}(A, B) = \sum_{u \in A, v \in B} w(u, v) \)

Can lead to isolated points

\[
NCut(A, B) = \frac{\text{Cut}(A, B)}{\text{Assoc}(A, V)} + \frac{\text{Cut}(A, B)}{\text{Assoc}(B, V)}
\]

\[
= 2 - \left( \frac{\text{Assoc}(A, A)}{\text{Assoc}(A, V)} + \frac{\text{Assoc}(B, B)}{\text{Assoc}(B, V)} \right) = 2 - N\text{Assoc}(A, B)
\]

GROUPING & SEGMENTATION

References

- Rother et al., GrabCut -Interactive Foreground Extraction using Iterated Graph Cuts, SIGGRAPH 2004.
GROUPING AND SEGMENTATION

Graph Based Approaches

Instead of minimizing cut, minimize normalized cut, or maximize normalized association.

\[
NCut(A, B) = \frac{\text{Cut}(A, B)}{\text{Assoc}(A, V)} + \frac{\text{Cut}(A, B)}{\text{Assoc}(B, V)}
\]

\[
= 2 - \left( \frac{\text{Assoc}(A, A)}{\text{Assoc}(A, V)} + \frac{\text{Assoc}(B, B)}{\text{Assoc}(B, V)} \right) = 2 - N\text{Assoc}(A, B)
\]

- Instead of minimizing cut, minimize normalized cut, or maximize normalized association.
- Note that for this case, you might not even need a "unary" cost.
  - The solution of assigning all nodes to A or B is sub-optimal, because now the denominator goes to 0.
  - So this can be viewed as a form of clustering
- NP-hard even for the binary case. But there are approximate solvers based on "real-valued" relaxations, that correspond to solving eigen-decomposition problems on the graph's adjacency matrix.
- For details

MACHINE LEARNING

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.

MACHINE LEARNING

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")

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,

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

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

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, …)
  - 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
MACHINE LEARNING

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 \( P_{XY}(x, y) \):

\[ f = \arg \min_{f \in \mathcal{H}} \int \int L(y, f(x)) \, p_{XY}(x, y) \, dy \, dx \]

- 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 \( (x_i, y_i) \),

\[ f = \arg \min_{f \in \mathcal{H}} \frac{1}{T} \sum_i L(y_i, f(x_i)) \]

You’re given data. Choose a loss function that matches your task, a hypothesis space \( \mathcal{H} \), and minimize.
MACHINE LEARNING

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!

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

\[
w, b = \arg \min_{w \in \mathbb{R}^{d+1}} \frac{1}{T} \sum_t (y_t - w^T \tilde{x}_t - b)^2
\]

So we know how to solve this: take derivative and set to 0.

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.

Define \( \tilde{x}_t = [x_t^T, 1]^T \)

\[
w = \arg \min_{w \in \mathbb{R}^{d+1}} \frac{1}{T} \sum_t (y_t - w^T \tilde{x}_t)^2
\]

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

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 \( L(y, \hat{y}) = (y - \hat{y})^2 \).

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

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

And you get exactly the same equation!

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

- 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?

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.

Too simple

Can be fixed if we had a lot more data.
While we train on empirical loss, we care about the actual expected loss:

\[ \frac{1}{T} \sum_{i} L(y_i, f(x_i)) \]

we care about the actual expected loss:

\[ \int_{x} \int_{y} L(y, f(x)) P_{XY}(x, y) \, dy \, dx \]

Why? Because we don’t want to explain the training set. We want to do well on new inputs. We want to "generalize" from the training set.

This is why a more complex function that exactly fits the training set is bad, when it "generalizes" poorly.

**Bias-Variance Tradeoff**

Choosing a simple function class: higher approximation error, lower estimation error.

Choosing a complex function class: lower approximation error, higher estimation error.

How do I decrease Bayes Error? By getting better inputs!

**Error = Bayes Error + Approximation Error + Estimation Error**

- **Bayes Error**: This is the error due to the uncertainty in \( p(y|x) \). This is the error you would have even if you knew the exact distribution and could craft a function \( f \) with infinite complexity.

  \[ \text{Bayes Error} = \int \left( \int \min \, L(y, \hat{y}) P_{XY}(x, y) \, dy \right) \, dx \]

- **Approximation Error**: This is the error due to the limited capacity of our hypothesis space \( \mathcal{H} \). It is the error of the true best function \( f \in \mathcal{H} \), minus the Bayes error, assuming we had perfect knowledge of \( P_{XY} \).

  Also called the "Bias".

- **Estimation Error**: This is the remaining component of error, caused by the fact that we don’t know the true \( P_{XY} \), but only have a limited number of samples.

  This depends on the size of our training set (and also, how well we are able to do the minimization). Called "variance".

**Overfitting**

- Definitions of complexity of a function or hypothesis space \( \mathcal{H} \): VC-dimension, Rademacher complexity
- Try to capture that one function or function space provides a "simpler" explanation than another
- Useful as intuition. But often don’t "work" for very complex functions and tasks.
- But the idea is:
  - Given two functions with the same error, you want to choose one that’s simpler.
  - You never want to consider a class of functions that can fit ‘random’ \( T \) pairs of \((x, y)\), where \( T \) is the size of your training set.
- Choose hypothesis space based on size of training set.
- Add a "regularizer" (for example, a squared penalty on higher order polynomial coefficients).

**Public Service Announcement**: Any regularizer is biased by what you think is "simple". There is no universal definition of simple.