CSE 559A: Computer Vision

Fall 2017: T-R: 11:30-1pm @ Lopata 101

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Staff: Abby Stylianou (abby@wustl.edu), Jarett Gross (jarett@wustl.edu)

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

Nov 7, 2017
• PSET 4 due next Tuesday.

• Recitation this Friday November 10.
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 \( P_{XY}(x, y) \):

\[
f = \text{arg min}_{f \in \mathcal{H}} \int_{\mathcal{X}} \int_{\mathcal{Y}} L(y, f(x)) \ P_{XY(x,y)} \ dy \ dx
\]

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 \((x_t, y_t)\),

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

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

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

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!
MACHINE LEARNING

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

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

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

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

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

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

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

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!

\[
\begin{align*}
    w &= \left( \sum_{t} \tilde{x}_t \tilde{x}_t^T \right)^{-1} \left( \sum_{t} \tilde{x}_t y_t \right)
\end{align*}
\]

- 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 \( L2 \) loss, and where the hypothesis space is linear in the parameters (not necessarily in the input).
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?
Why not just fit the more complex model?

Linear Fit

Quadratic Fit

12th Order Polynomial

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?

Too simple

Just Right

Too complex

Quadratic Fit
MACHINE LEARNING

Why not just fit the more complex model?

Too complex

Can be fixed if we had a lot more data.
While we train on empirical loss,

\[ \frac{1}{T} \sum_{t} L(y_t, f(x_t)) \]

we care about the actual expected loss:

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

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.
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.

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

- **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".\(^2\)

- **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".
Error = Bayes Error + Approximation Error + Estimation Error

**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!
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).
Overfitting: Good ML "Hygiene"

Remember, you can overfit not just the parameters, but your design choices!

For any given task:

- Have train, dev, val, and test set.
- Train your estimators on the train set.
- Choose hyperparameters based on dev set.
  - Function class
  - Regularization weight
  - Number of iterations to train, etc.
- Keep periodically checking to see if it generalizes to val set.
- Look at the test set only at the "end" of the project.
Consider the case when $y$ is binary, i.e., $\mathcal{Y} = \{0, 1\}$.

How do you define the loss function then?

- Ideally, $L(y, \hat{y})$ is 0 if they are equal, 1 otherwise.

But don't know how to solve that. What if we solved by regression?

$$w = \arg \min_w \frac{1}{T} \sum_t (y_t - w^T \tilde{x}_t)^2$$

And at test time, we can output $y = 1$ if $w^T \bar{x} > 0.5$ and 0 otherwise.

The problem is the loss function will penalize $w^T \bar{x}_t > 1$ when $y_t = 1$. While at test time, this would give us exactly the right answer!
Logistic regression

- Learn a function $f(x) = P(y == 1)$ which regresses to the probability $y$ is 1.
- We have to choose $f$ such that the domain of $f(x)$ lies between $[0, 1]$.

$$f(x; w) = \sigma (w^T \tilde{x}) , \quad \sigma(p) = \frac{\exp(p)}{1 + \exp(p)}$$

- This ensures that the output of $f$ is between $[0, 1]$
- $w^T \tilde{x}$ can be interpreted as the log of the odds, or log of ratio between $P(y == 1)$ to $P(y == 0)$
- Again, can choose any way of constructing augmented vector $\tilde{x}$ from $x$. This means that then the log-odds will be linear / polynomial / etc. function of $x$.

What about the loss?
Logistic regression

\[ P(y == 1) = f(x) = \sigma (w^T \tilde{x}) \]

Cross-Entropy Loss

If true \( y \) is 1, we want \( f(x) \) to be high, and if it is 0, we want it to be low.

\[ L(y, f(x)) = -y \log f(x) - (1 - y) \log(1 - f(x)) \]

There's a minus because this is the loss.

Minimizing \( \sum_t L(y_t, f(x_t)) \) can be viewed as maximizing the product of the probabilities of the labels \( y_t \).

But now, how do we minimize this function in terms of \( w \)? No longer least-squares.
NEXT TIME

- Gradient Descent & Stochastic Gradient Descent
- Back-propagation & Neural Networks