CIS6930/4930 – Probability for Computer Systems and Machine Learning

Introduction

Jan. 06, 2022

Prof. Ye Xia

Objectives

- Introduce probability specifically to CS students in terms of
  - selection of material
  - assumption of students' background
- Some distinctive features:
  - emphasize probability calculations so that it is usable
  - will do some measure theory: to be elaborated
- Expected outcomes: students can
  - do non-obvious probability calculations when thinking through system designs
  - do discrete event simulation
  - understand probability aspects in many areas of CS
  - read advanced literature

What the Course is and is not

- It is mainly a course on probability.
- Will draw examples/motivations from computer systems and machine learning (ML) areas.
- It should be broadly useful to many areas of computer science.
- There is much interest in ML. This is not a course on ML. But, it will be useful to study/understand (statistical) ML properly.
- I will try to draw more examples from ML.
- There will be no neural networks.

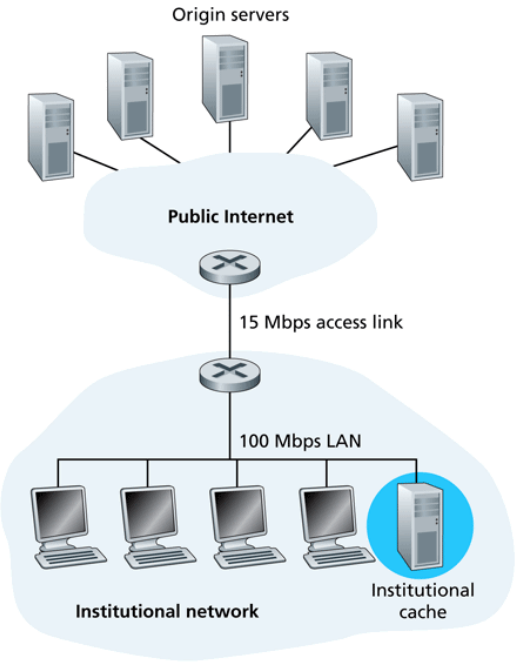
Related Courses

- CAP6610 - Machine Learning: covers many machine learning methods, not just probabilistic ones. I'd think this course complements it.
- CAP6617 - Advanced Machine Learning: more specialized; requires very good understanding of probability and other mathematics.
- CAP4621 - Artificial Intelligence: slightly related
- COT5615 - Math for Intelligent Systems: probability is one of the several components. The coverage will be different from this course.
- Probability courses in other departments: They do overlap with this course. The courses differ in material selection and assumptions on background.

Logistics

- Lectures will be recorded and posted on Canvas.
- Prerequisites: calculus, linear algebra and statistics at the level of MAC2313, MAS3114 and STA3032.
- Homework (75%): around five assignments
- Project (25%): simple discrete event simulation.
  I will provide the C code for priority queue. If you use another language, you can search for similar code or implement your own priority queue.
- No exams
- Follow the assignments, due dates and other schedules on Canvas.

Example from Computer Systems: Cache



- There are N files (e.g., web page objects), originally residing in far-away Internet (origin) servers. N is large.
- Users make requests for the files.

- There is a cache server nearby the users. A user request goes to the cache first.
- If the cache does not have the file, the request goes to the origin server. The requested file goes to the cache first and is cached, and then to the user.
- File i has a size S_i and a popularity rating p_i.
- p_i is the probability that a request is for i.
- Suppose the resource limitation is at the access link (in-bound direction). Its bandwidth is denoted C_a, a constant.
- The institution network bandwidth is C_i, a constant.
- Assume the propagation time within the institution network is 0.
- The one-way propagation time between the institution network and the origin server(s) is D, a constant (alternatively, random).

Project: Evaluate Cache Replacement Policies

- The cache storage capacity is S_c, with $S_c < \sum_{i=1}^{N} S_i$.
- Replacement Policies: Oldest First; Least-Popular First; others?
- Performance metric: average response time experienced by the users
- What are random: The file requests follow a Poisson process with rate λ.
- Each file size S_i is a sample drawn from a Pareto distribution (heavy tail).
- The popularity p_i is drawn from another Pareto distribution.
- It may be possible to analyze this system, but it won't be easy.
- Will study this by **discrete event simulation – your project**.

Example from Machine Learning

- Much of ML can be brought under statistical decision theory.
- $X \in \mathbb{R}^p$: random input vector
- $Y \in \mathbb{R}$: random output scalar
- $P(X, Y)$: joint distribution, which we don't really know
- We observes realizations of (X, Y) or the training data: $(x_1, y_1), \ldots, (x_N, y_N)$. We wish to learn:
  - $P(X, Y)$. Example: clustering. If so, we know everything except that there is uncertainty in the knowledge due to limited data.
  - More often, we like to think X and Y are functionally related by $Y = f(X)$, which isn't necessarily true, but can be useful for prediction or classification. In this case, we wish to learn about f.

- If we have f, given another input vector x_0, we can predict the output $\hat{y}_0 = f(x_0)$.
- $\hat{y}_0$ is usually not the same as the true output y_0. We need to worry about how good the prediction is.

  Note: If X and Y are truly related through $P(X, Y)$, then given $X = x_0$, Y is a random variable fully described by the conditional probability $P(Y|X = x_0)$. The prediction using $\hat{y}_0 = f(x_0)$ is a drastic simplification.
- The first problem: Oftentimes, there is $P(X, Y)$, but not necessarily f.
  - Sometimes, X and Y may be related by $Y = f(X) + \varepsilon$, where ε is random; but in general, not even that.
  - Even when $Y = f(X) + \varepsilon$, it is incorrect to say $Y = f(X)$.
  - We shall write $\hat{Y} = f(X)$, where $\hat{Y}$ is the **predicted output**.

Square Error Loss

- If we insist of predicting Y based on input X, i.e., $\hat{Y} = f(X)$, what will be f? We need a **loss function** $L(Y, f(X))$.
- It is common to use the **square error loss** $L(Y, f(X)) = (Y - f(X))^2$.
- The expected loss (or expected prediction error EPE) is $EPE(f) = E[L(Y, f(X))] = E[(Y - f(X))^2]$.
- The expectation is taken with respect to $P(X, Y)$, which is assumed known at this point. To be clear, we can write $E_{X,Y}$.

What f minimizes $EPE(f)$ above?

Answer: the conditional expectation $E[Y|X]$, which is a function of X.

$$EPE(f) = E_{X,Y}[(Y - f(X))^2] = E_X E_{Y|X}[(Y - f(X))^2|X]$$

- Here, when computing the expectation $(Y - f(X))^2$ with respect to the joint distribution of (X, Y), we do the usual trick of '**conditioning on something first**'. Here, we first condition on X and compute the expectation $(Y - f(X))^2$ with respect to the conditional distribution of Y given X. We are left with a function of X, $h(X) \triangleq E[(Y - f(X))^2|X]$; $h(X)$ is random.
- We then compute the expectation of $h(X)$ with respect to the distribution of X.

What f minimizes $EPE(f)$ above?

At each $X = x$,

$$E_{Y|X}[(Y - f(X))^2|X = x] = E_{Y|X}[(Y - f(x))^2|X = x],$$

We'd like to know what $f(x)$ (which will be a number for the fixed x) minimizes the above expectation.

Suppose the minimum is denoted by $f^*(x)$. Then, the function f^* must minimize $E_{X,Y}[(Y - f(X))^2]$ since

$$\begin{aligned} E_{X,Y}[(Y - f^*(X))^2] &= E_X E_{Y|X}[(Y - f^*(X))^2|X] \\ &\leq E_X E_{Y|X}[(Y - f(X))^2|X] \\ &= E_{X,Y}[(Y - f(X))^2]. \end{aligned}$$

Why the inequality: For two functions h and g with $h \leq g$, we have

$$E[h(X)] \leq E[g(X)],$$

because expectation is a weighted sum, or in general, an integral.

Now, at $X = x$, $f(x)$ is a number. To minimize $E_{Y|X}[(Y - f(x))^2|X = x]$ over f at $X = x$ is the same as saying $\min_c E_{Y|X}[(Y - c)^2|X = x]$.

$$E_{Y|X}[(Y - c)^2|X = x] = E_{Y|X}[Y^2 - 2cY + c^2|X = x]$$
$$= E_{Y|X}[Y^2|X = x] - 2cE_{Y|X}[Y|X = x] + c^2.$$

By taking the derivative with respect to c, we see that the minimum c is equal to $E_{Y|X}[Y|X = x]$, the conditional expectation of Y given $X = x$. We then have $f^*(x) = E_{Y|X}[Y|X = x]$.

Conclusion: We see that the function $f(X)$ that minimizes $EPE(f)$ is equal to $E[Y|X]$, the conditional mean or expectation.

$E[Y|X]$ is the 'weighted' average of Y for each given X, where the weight is the conditional distribution $P(Y|X)$.

Recall that here we assume $P(X, Y)$ is given.

Side Note

- Formula: $E_{X,Y}[g(X,Y)] = E_X E[g(X,Y)|X]$.
- Proof for discrete random variables: Suppose $p_{X,Y}$ is the joint probability mass function and p_X is the marginal probability mass function for X.

$$\begin{aligned} E_{X,Y}[g(X,Y)] &= \sum_{x,y} g(x,y) p_{X,Y}(x,y) \\ &= \sum_x \sum_y g(x,y) p_{Y|X}(y|x) p_X(x) \\ &= \sum_x E_{Y|X}[g(x,Y)|X = x] p_X(x) \\ &= E_X E_{Y|X}[g(X,Y)|X]. \end{aligned}$$

Random Sample

In reality, we do not know $P(X, Y)$. We observe the training data $(x_1, y_1), \ldots, (x_N, y_N)$.

Each (x_i, y_i) is drawn from the distribution $P(X, Y)$. Different (x_i, y_i)'s are drawn independently from each other.

That is, $(x_1, y_1), \ldots, (x_N, y_N)$ is realization of the underlying IID random variables $(X_1, Y_1), \ldots, (X_N, Y_N)$, each distributed as $P(X, Y)$

We call $(X_1, Y_1), \ldots, (X_N, Y_N)$ a **random sample**, or simply a **sample**.

We often abuse the notation and use $(x_1, y_1), \ldots, (x_N, y_N)$ to represent the random sample.

Key: Whatever we will learn or compute from the sample $(x_1, y_1), \ldots, (x_N, y_N)$ is random. We can talk about the mean and variance of the learned result.

Supervised Learning

Now, we insist on predicting Y based on X, i.e., $\hat{Y} = f(X)$, and we know that in principle the best prediction function is $f(X) = E[Y|X]$ (assuming the goal is to minimize the square error loss).

Question: How do we figure out such f based on a sample?

Answer: We will do 'hackish' things (many learning algorithms), and maybe later, try to improve or say something more definitive.

Much of statistical learning is about improvement and/or saying something more definitive.

The 'supervised' part: We observe a set of input-output pairs, the training data, based on which we learn f. That is, we learn based on known examples.

One Route: Parametric Models

We consider a class of functions indexed by a set of parameters, β (in general a vector), $\{g_\beta\}_{\beta \in \Theta}$, where Θ is the set of allowed parameters.

We assume $f = g_\beta$ for some β and will find the best β based on the training data.

Such β must be computed based on the training data $(x_1, y_1), \ldots, (x_N, y_N)$. Therefore, β can be viewed as being random.

Now, suppose the original X is a p-dimensional vector and we write $X = (X_1, \ldots, X_p)^T$. Each X_j is a component of X instead of a sample.

Therefore, each x_i is a p-dimensional vector. Y and y_i are scalars.

Example - Linear (Affine) Functions

Suppose we have scalars $z_1, \ldots, z_p$ and $\beta_0, \ldots, \beta_p$.

For convenience, we define $z = (z_1, \ldots, z_p)^T$, $\bar{z} = (1, z_1, \ldots, z_p)^T$ and $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^T$.

A function of the form $\bar{z}^T\beta = \beta_0 + \sum_{j=1}^{p} z_j\beta_j$ is affine in z and linear in β. We will see the linearity in β is more important.

The family of functions are $g_\beta(z) = \bar{z}^T\beta$, where $\beta \in \Theta = \mathbb{R}^{p+1}$.

But, if $E[Y|X = z]$ indeed has the form $\bar{z}^T\beta$, then, we need to find β to minimize $E_{X,Y}[(Y - \bar{X}^T\beta)^2]$, where $\bar{X} = (1, X_1, \ldots, X_p)^T$.

We can approximate $E[(Y - \bar{X}^T\beta)^2]$ using the training data and then minimize over β. That is,

$$\min_{\beta} \sum_{i=1}^{N} (y_i - (\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j))^2. \tag{1}$$

This is the method of **least squares**.

Consider the special case where X and Y are related by $Y = h(X) + \varepsilon$ for some function h, and the random error ε is independent of X and has $E[\varepsilon] = 0$. Then, $f(X) \triangleq E[Y|X] = h(X)$.

If h is not a linear (or affine) function, then $\bar{z}^T \beta$ may not be close to $h(z)$ for any β. Then, there seems to be little reason that the method of least square will give us good prediction.

What do we even mean by that?

Linear Regression

We will solve (1). Let $\mathbf{y} = (y_1, \ldots, y_N)^T$.

Let $\mathbf{X}$ be the $N \times (p+1)$ matrix based on the input vectors:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Np} \end{bmatrix}$$

The function to be minimize in (1) can be written as

$$RSS(\beta) \triangleq (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta),$$

where RSS stands for residual sum of squares.

Then, the gradient and Hessian of RSS are:

$$\frac{\partial RSS}{\partial \beta} = \nabla RSS(\beta) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)$$

$$\frac{\partial^2 RSS}{\partial \beta \partial \beta^T} = \nabla^2 RSS(\beta) = 2\mathbf{X}^T\mathbf{X}.$$

Let us assume $\mathbf{X}$ has full column rank and therefore $\mathbf{X}^T\mathbf{X}$ is positive definite. Then, $RSS(\beta)$ is convex. Its minimum is obtained by solving $\nabla RSS(\beta) = 0$. We get the unique solution:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2}$$

We now can define the prediction function $\hat{f}$. At any input vector x_0,

$$\hat{f}(x_0) = (1, x_0^T)\hat{\beta}.$$

where $(1, x_0^T)$ is the row vector by concatenating 1 with the x_0^T.

The predicted output is denoted by $\hat{y}_0$, where $\hat{y}_0 = \hat{f}(x_0)$.

Interpretations and Discussions

- $\hat{\beta}$ is (2) depends entirely on the training data $(x_1, y_1), \ldots, (x_N, y_N)$.
- When the training data are viewed as a random sample, we can say $\hat{\beta}$ is a random vector. The distribution of the random sample is the joint distribution $P((X_1, Y_1), \ldots, (X_N, Y_N))$, where (X_i, Y_i)'s are IID, each distributed as (X, Y).
- For simplicity, let $\mathcal{T}$ denote the distribution of the training sample.
- At each input vector x_0, the predicted output $\hat{y}_0 = \hat{f}(x_0)$ depends entirely on the training data and $\hat{y}_0$ is random.
- We can ask the mean (vector) and covariance (matrix) of $\hat{\beta}$, and the mean and variance of $\hat{y}_0$.

- The true output Y_0 is also random, with the distribution $P(Y|X = x_0)$, which can be computed from the joint distribution $P(X, Y)$.

- We can ask about the bias in the prediction (aka estimate): $E[Y_0 - \hat{f}(x_0)|X = x_0]$, where the expectation is over $\mathcal{T}$ and $P(Y|X = x_0)$.

  This is the difference between the mean of the estimate and the true mean.

  The estimator/prediction is said **unbiased** if $E[Y_0 - \hat{f}(x_0)|X = x_0] = 0$.

- We can talk about the prediction error at x_0

$$Err(x_0) = E[(Y_0 - \hat{f}(x_0))^2|X = x_0].$$

Bias and Variance of Estimator

- Special Case: $Y = f(X) + \varepsilon$, where ε is an independent noise with 0 mean and variance σ^2.
- f is not necessarily linear. The estimator $\hat{f}$ is not necessarily from the method of least squares.
- The bias is: $E[Y_0 - \hat{f}(x_0)|X = x_0] = f(x_0) - E_{\mathcal{T}}[\hat{f}(x_0)]$.
- Will show: The prediction error is the sum of an irreducible component σ^2, the squared bias and the variance of the estimate.
- There is often a trade-off between the bias and the variance in different estimates/models/prediction methods.

$$
\begin{aligned}
Err(x_0) =& E[(Y_0 - \hat{f}(x_0))^2 | X = x_0] \\
=& E[(Y_0 - f(x_0) + f(x_0) - \hat{f}(x_0))^2 | X = x_0] \\
=& E[(\varepsilon + f(x_0) - \hat{f}(x_0))^2 | X = x_0] \\
=& E[\varepsilon^2 + 2\varepsilon(f(x_0) - \hat{f}(x_0)) + (f(x_0) - \hat{f}(x_0))^2 | X = x_0] \\
=& \sigma^2 + E[(f(x_0) - \hat{f}(x_0))^2] \qquad (3) \\
=& \sigma^2 + E[(f(x_0) - E\hat{f}(x_0) + E\hat{f}(x_0) - \hat{f}(x_0))^2] \\
=& \sigma^2 + (f(x_0) - E\hat{f}(x_0))^2 + E(E\hat{f}(x_0) - \hat{f}(x_0))^2 \\
& - 2(f(x_0) - E\hat{f}(x_0))E[E\hat{f}(x_0) - \hat{f}(x_0)] \\
=& \sigma^2 + \text{Bias}^2(\hat{f}(x_0)) + \text{Var}(\hat{f}(x_0)).
\end{aligned}
$$

Starting from (3), the expectation is taken with respect to $\mathcal{T}$.

Example - k-Nearest Neighbor

Again, consider the special case $Y = f(X) + \varepsilon$.

Recall the least-squares method. Instead of the optimal predictor $f(X) = E[Y|X]$, we use a function of the form $\hat{f}(X) = \bar{X}^T\beta$ as a predictor, with an appropriately chosen β.

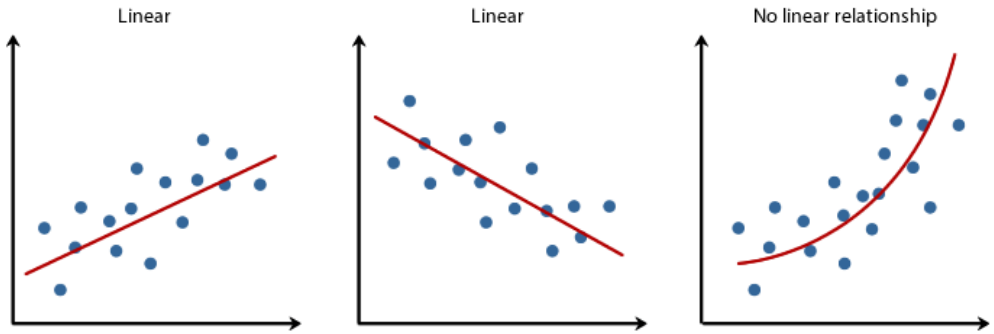
β is chosen by minimizing an estimate of the expected prediction error $EPE(\hat{f}) = E_{X,Y}(Y - \hat{f}(X))^2$.

That is, we use the sum of squares from the training data as an estimate of $EPE(\hat{f})$ and minimize over β.

$$\min_{\beta} \sum_{i=1}^{N} (y_i - (1, x_i^T)\beta)^2.$$

The optimal $\hat{\beta}$ uses information from all the training data.

The bias at $X = x_0$ is $E[Y_0 - (1, x_0^T)\hat{\beta}|X = x_0] = f(x_0) - E[(1, x_0^T)\hat{\beta}]$.



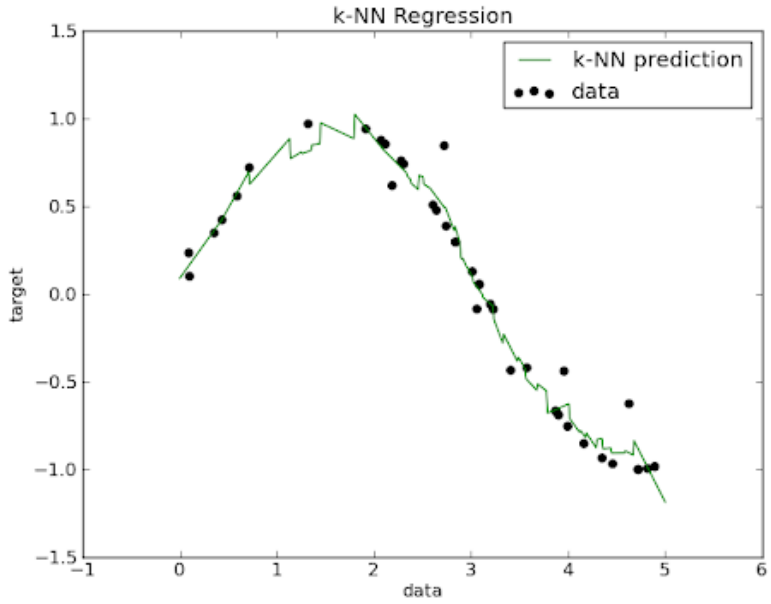
If f is significantly different from any linear function, the least-squares predictor will have significant bias at different input vectors, and hence, possibly significant prediction error.

We will consider the k-**nearest-neighbor** predictor/estimator.

$$\hat{f}(x_0) = \text{Ave}(y_i | x_i \in N_k(x_0)).$$

$N_k(x_0)$: the set containing k input vectors (in the sample) closest to x_0;

Ave: the average.



$\hat{f}(x_0)$ is used to approximate $E[Y|X = x_0]$:

- expectation is approximated by averaging over the training data
- conditioning on a point x_0 is relaxed to conditioning on some region 'close' to x_0.

If all the input vectors in $N_k(x_0)$ are very close to x_0, taking the average of the corresponding outputs should approximate $E[Y|X = x_0]$ well.

If the input has high dimensional, the input vectors in $N_k(x_0)$ are usually not close to x_0 at all $\rightarrow$ the 'curse of dimensionality'.

Linear Models vs. k-Nearest Neighbor

- For simplicity of discussion, suppose $Y = f(X) + \varepsilon$.
- We wish to approximate f by constructing some $\hat{f}$ based on the training data, which can be viewed as a noisy representation of f.
- In the linear models, $\hat{f}(z)$ is found in the class $(1, z^T)\beta$ by the method least squares.
- If $f(z)$ is far from linear, none of the of linear functions $(1, z^T)\beta$ may approximate f well.
- Least squares and linear model use the training data in 'global' fashion.
- When $f(z)$ is complex looking, the k-nearest-neighbor method may adapt to $f(z)$ better.
- It uses the training data in a local fashion.

- But, it has problems in dealing with high-dimensional input; may require a large amount of training data; the resulting $\hat{f}$ lacks smoothness.

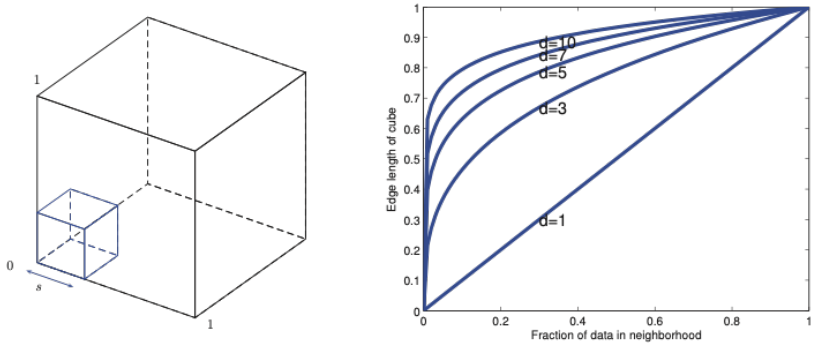


Figure 1: Cure of Dimensionality

- Many popular learning methods are variants of the above two methods. They often address the issues in either method.
- Examples: kernel methods, local regression, basis expansion.