A Quick Write-up on Mutual Information Estimation

Alan Yang

October 6, 2018

1 Intro and Preliminaries

Many quantities in the real world, for example the temperature outside tomorrow, the value of a dice roll, and a person's height, can be modeled as a **random variable**. We use random variables to describe things that have uncertainty around them. Loosely, a random variable X has an **alphabet** $\mathcal{X}$ and a probability distribution $p(x)$ that associates a non-negative probability to each $x \in \mathcal{X}$.

For example, let X be the result of a fair coin flip. In this case it is associated with alphabet

$$\mathcal{X} = \{\text{heads}, \text{tails}\}$$

and probability mass function

$$p(x) = \begin{cases} \frac{1}{2} & x = \text{heads} \\ \frac{1}{2} & x = \text{tails} \end{cases}$$

In this case, X was discrete. For continuous X, its associated alphabet will be infinite and have a continuous probability density function. Integrals can be interchanged with summations for discrete variables. Some useful quantities you might be familiar with:

1. The **expected value** of X:

$$\mu_X = \mathbb{E}[X] = \int x \cdot p(x) dx$$

2. The **variance** of X:

$$\sigma_X^2 = \text{Var}(X) = \mathbb{E}\left[(X - \mu_X)^2\right]$$

3. The **covariance** between X and another random variable Y:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)]$$

4. The **entropy** of X describes how uncertain we are about its value, and is measured in bits (for log base 2).

$$H(X) = -\int p(x) \log p(x) dx$$

2 Mutual Information

Let's say we have two random variables X and Y, and want to figure out how strongly they are related. One measure is the Pearson correlation coefficient

$$\rho_{X,Y} = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}\left[(X - \mu_X)(Y - \mu_Y)\right]}{\sigma_X \sigma_Y}$$

which is able to measure linear associations between X and Y. For example, a positive correlation means that a linear increase in X leads to a linear increase in Y. However, zero correlation does not imply independence; X and Y are independent if and only if their joint probability distribution factorizes as

$$p(x,y) = p(x)p(y)$$

It would be nice if there were some generalized way to measure how far X and Y are to being independent. Their **mutual information** fills this role:

$$I(X;Y) = \int \int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy$$

Notice that if X and Y are independent, the log term becomes one, and $I(X;Y) = 0$. In fact, this is an if and only if condition. Mutual information is non-negative, and is measured in bits for log base 2.

3 Estimating Mutual Information of Continuous Variables from Data

3.1 The Main Idea

If we knew the distributions of X and Y, we could compute the integral to find their mutual information

$$I(X;Y) = \int \int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy$$

However, in real life we almost never don't know the joint distribution $p(x,y)$ or the marginal distributions $p(x)$, $p(y)$; we can only estimate them from data. Let's say that X and Y take on scalar real values, and we are given a dataset of N pairs (x_i, y_i) drawn i.i.d. from their joint distribution $p(x,y)$:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

A plot of a dataset might look something like this:

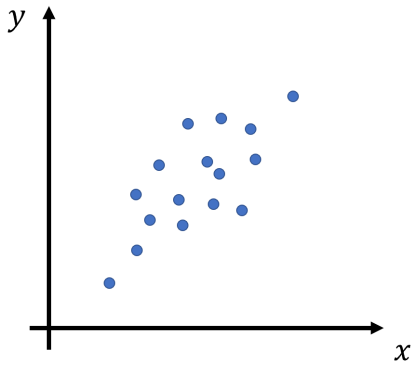

Figure 1: Plot of dataset $\mathcal{D}$

One way to estimate $p(x, y)$ would be to use a discrete approximation of X and Y. For example, you could bin the 2D space into a bunch of little rectangles, and consider each rectangle as a single event. the empirical probability of each event rectangle would then be the number of points inside the rectangle divided by the total number of points.

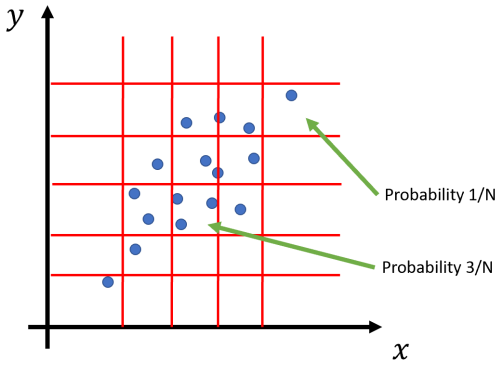

Figure 2: Discretized dataset

This binning approximation can be used to estimate the mutual information. Each bin can be represented by a coordinate pair i, j, with associated joint probability:

$$p(i, j) = \frac{\#\text{ points in } i, j \text{ bin}}{N}$$

And marginal probabilities

$$p(i) = \frac{n_x = \#\text{ points in bin } i, \cdot \text{ for any } j}{N}$$

$$p(j) = \frac{n_y = \#\text{ points in bin } \cdot, j \text{ for any } i}{N}$$

This leads to an estimator for mutual information:

$$\hat{I}_{binned}(X;Y) = \sum_{i,j} p(i,j) \log \frac{p(i,j)}{p(i)p(j)}$$

In the figure, there are 5 bins for X values (index i) and 6 bins for Y values (index j), meaning that there are a total of 30 bins. It is true that as the dataset size N grows to infinity and the size of each rectangle tends to zero, $\hat{I}_{binned} \to I$, the true value. However, there are many errors that can show up in this approximation for finite samples and finite discretization. The KSG estimator (named after the inventors) was developed to reduce these errors as much as possible.

3.2 The KSG Nearest Neighbor estimator

In short, the KSG estimator tries to fix the box sizing issue by not specifying the box size for each discretized event beforehand. Instead, around each datapoint (x_i, y_i) the algorithm considers the k nearest neighboring points as part of the same box, or event, and estimates the joint probability of that box as k/N. The marginal probabilities of each datapoint can be estimated as $n_{x,i}/N$ and $n_{y,i}/N$, following the procedure as above. Replacing the log (base 2) with the *digamma function* (to reduce the error induced by estimating the log of a probability with the log of a fractional count), this leads to the estimator

$$\begin{aligned}\hat{I}_{KSG}(X;Y) &= \frac{1}{N}\sum_{i=1}^{N} \psi\left(\frac{\frac{k}{N}}{\frac{n_{x,i}}{N} \cdot \frac{n_{y,i}}{N}}\right) \\ &= \psi(k) + \psi(N) - \frac{1}{N}\sum_{i=1}^{N} \left(\psi(n_{x,i}) + \psi(n_{y,i})\right)\end{aligned}$$

The algorithm is written explicitly below:

Algorithm 1 Estimator for $I(X;Y)$. $\psi(\cdot)$ denotes the digamma function.

procedure KSG($\{X_i, Y_i\}_{i=1}^N$, k)
  for $i \in \{1, ..., N\}$ **do**
    $\epsilon_i =$ The k-th smallest distance among
      $\left\{ d_{i,j} = \max\left\{ \|X_i - X_j\|, \|Y_i - Y_j\| \right\} : j \neq i \right\}$
    $n_{x,i} = \#$ samples with $\max\left\{ \|X_i - X_j\| \right\} \leq \epsilon_i$
    $n_{y,i} = \#$ samples with $\max\left\{ \|Y_i - Y_j\| \right\} \leq \epsilon_i$
    $\xi_i = \psi(k) - \psi(n_{x,i}) - \psi(n_{y,i}) + \psi(N)$
  end for
  return $\hat{I}(X;Y) = \frac{1}{N} \sum_{i=1}^N \xi_i$
end procedure

Let's take a look at this algorithm. Calculating ϵ_i, $n_{x,i}$, and $n_{y,i}$ by brute force leads to a runtime $O(N^2)$, which can be too slow for large datasets. One way to improve this would be to first put the dataset into a k-d tree, and query the k-nearest neighbors for each datapoint, leading to an improved $O(N \log N)$ runtime. However, this problem is essentially an all nearest-neighbor problem; we want to know the k nearest neighbors of every datapoint. Is there a way to implement a solution faster than $O(N \log N)$?