Quantization

Alex Sverdlov
alex@theparticle.com

1 Introduction

Often there's a need to discretize features. For example, converting a real number into a range of $0 \dots 9$.

A quantizer is a function: $q(x)$ that takes a real number and produces an integer output between 0 and $L-1$, where L is the number of quantization levels for a given feature.

The function $q(x)$ can be trained from data samples: often setup such that each quantization level has an equal probability of occuring (this is not required, but is often very useful).

The feature vector can be quantized as well. For a measurement tuple $X = (x_1, \dots, x_N)$, we can define/train a quantizer q_i for each feature x_i, with L_i levels. The quantization $Q(X)$ is then $(q_1(x_1), \dots, q_N(x_N))$.

The function $Q(X)$ can be trained from data samples: perhaps by allocating appropriate number of levels per feature based on each feature's entropy in relation to other features.

The quantized feature vector is a list of integers from 0 to $L_i - 1$. This itself can be turned into a single number in range 0 to $\prod_i L_i$, essentially quantizatizing the measurement tuple into a single number, which can be used as an index into a lookup table, etc.

2 Implementation Notes

Given a measurement tuple, $X = (x_1, \dots, x_N)$, and our desire to quantize it into a number in $0 \dots Z$.

We first need to estimate the relative entropy of each x_i: take a sample of the data (or the entire dataset if not big), and for each feature i calcualte the entropy:

$$H_i = -\sum p_{ij} \log(p_{ij})$$

Where p_{ij} is the probability of jth value of feature i.

This gets us: $(H_1, \dots, H_N)$, which we normalize (ensure they sum to 1) to get $(w_1, \dots, w_N)$. The w_i indicates the proportion of entropy that feature i takes up.

Our original goal was to quantize into range $0 \dots Z$, so each column gets quantized into range $0 \dots w_i * \log(Z)$, lets call this number L_i, or number of levels for feature i.

We can now take a sample (or entire dataset) of feature i, sort it, and cut the sorted list into L_i equally sized slices.

At each of the slice indexes, we take the floor of the value at that index, which for feature i gets us a list of L_i numbers. This is our quatizer data, d_i.

To build the actual quantizer q_i for feature x_i we often use the "lower bound" function (this is binary search to locate the lower bound; google for it). The quatizer is essentially:

$$\text{lower\_bound}(d_i, x_i)$$

The above will return an integer in $0 \ldots L_i$ range.

Since we know each L_i, the tuple $(q_1(x_1), \ldots, q_N(x_N))$ can be viewed as an "index" into an N dimensional array, which is essentially a single offset from the start of the array.

This technique of turning the measurement tuple into an integer can be very powerful, especially when you consider that we can also subsample the dimensions (we don't have to pick all meaurement dimensions, but a small sample: build a lookup table, estimate results from that table, then take another sample of dimensions, etc.)