1 Mutual Information Algorithm

In this article I propose an algorithm to pick the top k most impactful features for a discrete dataset. Let X be the probability space on which we are working (our discrete set of samples), for some element $x \in X$ let us consider the random variable $Z(x)$ to be the feature we are aiming to predict (e.g the closing price of a stock), and let the random variables $Y_i(x)$ $1 \leq i \leq n$ be the rest of the features we have access to (e.g the closing price of that stock i days ago). Furthermore we assume that all of our random variables are also discrete. Any continuous features can be made discrete by putting them into buckets. As an example we can discretize prices by rounding to the closest dollar value.

We can then define a probability measure on our random variables with $|X|$ being the number of samples:

$$P(W = w) = \sum_{\substack{x \in X \\ W(x) = w}} 1 \cdot \frac{1}{|X|}$$

In other words the probability of some W (either Z or Y_i) being equal to w is just the total number of samples for which that feature has a value of w divided by the total number of samples. We can also define a joint probability distribution

$$P(W_1 = w_1, W_2 = w_2) = \sum_{\substack{x \in X \\ W_1(x) = w_1 \\ W_2(x) = w_2}} 1 \cdot \frac{1}{|X|}$$

So now we count the total number of samples for which both features match and again divide by the total number of samples. To make our life easier let us define $w \in W$ to represent all the possible values w that a feature W can take. Remember that we are in the discrete case and there are only finitely many options. We can now define the mutual information between two random variables W_1, W_2 using the formula:

$$I(W_1, W_2) = \sum_{\substack{w_1 \in W_1 \\ w_2 \in W_2}} P(W_1 = w_1, W_2 = w_2) \cdot \log\left(\frac{P(W_1 = w_1, W_2 = w_2)}{P(W_1 = w_1)P(W_2 = w_2)}\right)$$

Note that the numerator is equal to 0 whenever the denominator is equal to 0 and we exclude such terms from the summation. Notice that if two variables are independent then $P(X, Y) = P(X)P(Y)$ and as a result their mutual information is 0. Otherwise the mutual information is some positive number. If $X = Y$ then $P(X, Y) = P(X)$ where $x = y$ and 0 otherwise. in that case the mutual information

$$I(W, W) = \sum_{w \in W} P(W = w) \cdot \log\left(\frac{1}{P(W = w)}\right)$$

is the amount of information encoded by W. This means that the mutual information between two variables is less than or equal to the minimum of their information. With everything defined our goal becomes to find a subset of features $i_1, i_2 \dots i_k$ such that $Y_{i_1}, Y_{i_2} \dots Y_{i_k}$ have the maximum amount of mutual information for Z. Finally in order to run the algorithm we have to compute the

mutual information for a random variable represented by a tuple of random variables. We simply extend the formula by summing over more indicies. As an example:

$$I((W_1, W_2), W_3) = \sum_{\substack{w_1 \in W_1 \\ w_2 \in W_2 \\ w_3 \in W_3}} P(W_1, W_2, W_3) \cdot \log \left(\frac{P(W_1, W_2, W_3)}{P(W_1, W_2) P(W_3)} \right)$$

Algorithm 1 MUTUAL_INFORMATION

```
Require: n: INT                                          ▷ Total number of features
Require: k: INT                                          ▷ Number of top features to find
Require: Z: RV                                           ▷ Random Variable representing the target feature
  Declare Y: array[1, n] of RV                           ▷ array of random variables to consider
  Declare S: array[1, n] of [INT, FLOAT]                 ▷ feature index and mutual information pair
  Declare B: array[1, k] of [INT, FLOAT]                 ▷ Store best features here
  for 1 ≤ j ≤ n do
    S[j] ← [j, I(Z, Y[j])]                               ▷ compute mutual information here
  end for
  S ← sort S by S[*][2] descending
  B[1] ← S pop head
  for 2 ≤ j ≤ k do:
    bestIndex ← −1
    bestInformation ← 0
    for 1 ≤ i ≤ k − j + 1 do
      if S[i][2] < bestInformation then:
        break                                            ▷ information is only decreasing from here
      end if
      S[i][2] ← I((Z, B), Y[S[i][1]])]   ▷ Mutual information for Z with top j − 1 features and
      if S[i][2] > bestInformation then
        bestIndex ← S[i][1]
        bestInformation ← S[i][2]
      end if
    end for
    S ← sort S by S[*][2] descending
    B[j] ← S pop head
  end forOutput B
```