A Recursive Algorithm for Exponential Power Sum Combinations

Christopher Thompson
cjt5144@gmail.com

Published : June 6, 2016

Abstract

Data science exploration sometimes requires brute force generation of data set features before narrowing focus on relevant features. Polynomial feature generation of all exponential power sum (EPS) combinations less than and equal to the given power are generated using the algorithm below. If given features $\{x_1\ x_2\ x_3\}$, the EPS is the sum of exponents of the multiplied features. $x_1^2x_2x_3^3$ can be represented as an EPS of the 6th order $(2+1+3=6)$, while $x_1^2x_2x_3$ can be represented as an EPS of the 4th order $(2+1+1=4)$.

1 Algorithm Basis

A table is created with a column for each feature. Each feature has what we will call a *feature set*. A feature set is a set of combinations where the feature the set belongs to (the *feature set owner*) is the furthest left non-zero feature in the table. Feature sets are organized from smallest to largest *power set*. A power set contains all combinations within the feature set with the same EPS. For the first N-1 feature sets, the combinations within a power set begin with 1 in the feature owner slot and (power set EPS - 1) in the furthest right feature slot. The remaining power set combinations are filled by subtracting 1 from the furthest right non-zero (FRNZ) feature and adding 1 to the feature one column to the left of the FRNZ feature until the feature set owner equals the power set EPS. The Nth feature set is just Q power sets of one item each from 1 to Q, where Q is the max EPS defined as argument to the *powerTable* procedure.

An example of a *powerTable* generated table:

x_1	x_2	x_3	
1	0	0	x_1 power set 1
1	0	1	x_1 power set 2
1	1	0	
2	0	0	
0	1	0	x_2 power set 1
0	1	1	x_2 power set 2
0	2	0	
0	0	1	x_3 power set 1
0	0	2	x_3 power set 2

Table 1: An example of the Power Table created for features $\{x_1\ x_2\ x_3\}$ with a max EPS of 2.

2 Algorithm Pseudocode

The *powerTable* procedure generates a table of all possible EPS combinations.

Arguments:

1. N : The number of features
2. Q : The max EPS

Procedure 1 QUEUE powerTable(INT N, INT Q)

```
1: VECTOR v := <new VECTOR of N INTs := 0>
2: QUEUE pt := <new QUEUE>
3: for INT i := 1 to N − 1 do
4:    for INT p := 1 to Q do
5:       v[i] := 1
6:       v[N] := p − 1
7:       pt.push(v)
8:       recur(N, i, v, pt)
9:       v := <new VECTOR of N INTs := 0>
10:   end for
11: end for
12: for INT p := 1 to Q do
13:   v[N] += 1
14:   pt.push(v)
15: end for
16: return pt // Table of exponent combinations
```

The Recursive procedure adds exponent combinations to table.

Arguments:

1. a : The current FRNZ feature number
2. s : The current feature set owner number
3. v : The current combination being worked on
4. pt : The power table queue

Procedure 2 VOID recur(INT a, INT s, VECTOR v, QUEUE pt)

```
1: if a == s then
2:    return
3: end if
4: if v[a] == 0 then
5:    a -= 1
6:    recur(a, s, v, pt)
7: else
8:    v[a] -= 1
9:    v[a − 1] += 1
10:   pt.push(v)
11:   recur(a, s, v, pt)
12: end if
```

3 Algorithm Analysis

The top two nested levels of the algorithm run in O(N-1) and O(Q) runtimes, where N and Q are defined as arguments to *powerTable*. The recursive procedure must be called N - 1 times, for all but the first column. Within the N - 1 recursive calls, the recursive procedure needs to be called a maximum of Q times to move each column from p - 1 to 0. The recursive procedure is therefore called $(N-1)*Q$ times. This gives a runtime of $O((N-1)*Q*((N-1)*Q)) = O(N^2Q^2 - 2NQ + Q^2)$. If N is much larger than Q, the runtime simplifies to $O(N^2Q^2)$.