EFFICIENT CALCULATION OF POLYNOMIAL FEATURES ON SPARSE MATRICES

Andrew Nystrom

Savvysherpa Inc. 6200 Shingle Creek Pkwy Suite 400 Minneapolis, MN 55430, USA awnystrom@gmail.com John F. Hughes

Department of Computer Science Brown University Providence, RI jfh@cs.brown.edu

ABSTRACT

We provide an algorithm for polynomial feature expansion that operates directly on a sparse matrix. For a vector of dimension D and density d, the algorithm has time and space complexity $O(d^kD^k)$ where k is the polynomial order.

1 Introduction

Polynomial feature expansion has long been used in statistics to approximate nonlinear functions Gergonne (1974); Smith (1918). Despite this, we are unaware of any efforts to optimize calculating them. Here we provide an algorithm for calculating polynomial features for a vector of dimension D and density d with time and space complexity $O(d^kD^k)$ where k is the polynomial order, and $0 \le d \le 1$ is the fraction of elements that are nonzero. The standard algorithm has time and space complexity $O(D^k)$, so the added factor of d^k represents a significant complexity reduction. The algorithm avoids densification of the vector, i.e. the vector remains in compressed sparse row form, so the space complexity is also $O(d^kD^k)$ as opposed to $O(D^k)$.

2 ALGORITHM

In the naive computation of polynomial features for a vector \vec{x} , we create a new feature for each product (with repetition) of k features in \vec{x} (or without repetition, for "interaction features"). This ignores data sparsity and will yield a product of zero any time one of the features involved in the product is zero. In a sparse matrix, such zero-products are common. If we store vectors in a sparse matrix format, these zero-products need not be computed or stored.

The main idea behind our algorithm is to leverage sparsity by only computing products that do not involve zeros. In a compressed sparse row matrix, the columns containing nonzero data are the only columns that are stored. We can therefore iterate over products of combinations with repetition of order k of *only these columns* for each row to calculate k-degree polynomial features.

While the idea is straightforward, there is yet an unaddressed challenge: Given a multiset of column indices whose corresponding nonzero components were multiplied to produce a polynomial feature, where in the augmented polynomial vector does the result of the product belong? To address this, we give a bijective mapping from the set of possible column index combinations-with-repetition of order k onto the column index space of the polynomial feature matrix. Thus the map has the form

$$(i_0, i_1, \dots, i_{k-1}) \rightarrowtail p_{i_0 i_1 \dots i_{k-1}} \in \{0, 1, \dots, \binom{D}{k}\}$$
 (1)

such that $0 \le i_0 \le i_1 \le \cdots \le i_{k-1} < D$ where $(i_0, i_1, \dots, i_{k-1})$ are column indicies of a row vector \vec{x} of an $N \times D$ input matrix, and $p_{i_0 i_1 \dots i_{k-1}}$ is a column index into the polynomial expansion vector for \vec{x} where the product of elements corresponding to indices i_0, i_1, \dots, i_{k-1} will be stored.

2.1 Construction of Mappings

For the second degree case, we seek a map from matrix indices (i,j) (with $0 \le i < j < D$) to numbers f(i,j) with $0 \le f(i,j) < \frac{D(D-1)}{2}$, one that follows the pattern indicated by

$$\begin{bmatrix} x & 0 & 1 & 3 \\ x & x & 2 & 4 \\ x & x & x & 5 \\ x & x & x & x \end{bmatrix}$$
 (2)

where the entry in row i, column j, displays the value f(i,j). We let $T_2(n) = \frac{1}{2}n(n+1)$ be the nth triangular number; then in Equation 2, column j (for j>0) contains entries with $T_2(j-1) \le e < T_2(j)$; the entry in the ith row is just $i+T_2(j-1)$. Thus we have $f(i,j)=i+T_2(j-1)=\frac{1}{2}(2i+j^2-j)$. For instance, in column j=2 in our example (the third column), the entry in row i=1 is $i+T_2(j-1)=1+1=2$.

With one-based indexing in both the domain and codomain, the formula above becomes $f_1(i,j) = \frac{1}{2}(2i+j^2-3j+2)$.

For polynomial features, we seek a similar map g, one that also handles the case i=j. In this case, a similar analysis yields $g(i,j)=i+T_2(j)=\frac{1}{2}(2i+j^2+j+1)$.

To handle *three-way interactions*, we need to map triples of indices in a 3-index array to a flat list, and similarly for higher-order interactions. For this, we'll need the tetrahedral numbers $T_3(n) = \sum_{i=1}^n T_2(n) = \frac{1}{6}(n^3 + 3n^2 + 2n)$.

For three indices, i, j, k, with $0 \le i < j < k < D$, we have a similar recurrence. Calling the mapping h, we have

$$h(i,j,k) = i + T_2(j-1) + T_3(k-2); (3)$$

if we define $T_1(i) = i$, then this has the very regular form

$$h(i,j,k) = T_1(i) + T_2(j-1) + T_3(k-2); (4)$$

and from this the generalization to higher dimensions is straightforward. The formulas for "higher triangular numbers", i.e., those defined by

$$T_k(n) = \sum_{i=1}^n T_{k-1}(n)$$
 (5)

for k > 1 can be determined inductively.

The explicit formula for 3-way interactions, with zero-based indexing, is

$$h(i,j,k) = 1 + (i-1) + \frac{(j-1)j}{2} + \tag{6}$$

$$\frac{(k-2)^3 + 3(k-2)^2 + 2(k-2)}{6}. (7)$$

3 COMPLEXITY ANALYSIS

Calculating k-degree polynomial features via our method for a vector of dimensionality D and density d requires $\binom{dD}{k}$ (with repetition) products. The complexity of the algorithm, for fixed $k \ll dD$, is therefore

$$O\left(\binom{dD+k-1}{k}\right) = O\left(\frac{(dD+k-1)!}{k!(dD-1)!}\right)$$
(8)

$$=O\left(\frac{(dD+k-1)(dD+k-2)\dots(dD)}{k!}\right) \tag{9}$$

$$=O\left((dD+k-1)(dD+k-2)\dots(dD)\right) \text{ for } k\ll dD \tag{10}$$

$$=O\left(d^kD^k\right) \tag{11}$$

REFERENCES

JD Gergonne. The application of the method of least squares to the interpolation of sequences. *Historia Mathematica*, 1(4):439–447, 1974.

Kirstine Smith. On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of observations. *Biometrika*, 12(1/2):1–85, 1918.