Exact max function Shapley values

August 3, 2017

1 Compute the exact Shapley values of the max function in $O(M^2)$ complexity

While the exact Shapley values take $O(2^M)$ time each to compute in general (where M is the number of input features), here we show that for the max function they can be found much more quickly in $O(M^2)$ time to compute all M values. We assume a single reference input, not a whole dataset, so repeating for a set of samples from the dataset would be necessary for computing expectations.

Below is the algorithm (in Julia code) with input vector x and a reference vector r. It treats the max function as a decision tree and weights each possible binary outcome with the number of permutations that match it. Note that once a value is fixed that is greater than all other values then the decision tree stops branching. By sorting the inputs by the maximum of their input and reference values we can ensure that one branch of the decision tree will stop branching at each step. Once at a leaf in the tree we compute the effects for all the features encountered so far.

```
>>> function shapley_max(x, r)
...
...     # sort so that at each step we know either the input value
...     # or the reference value of the next feature will be the next largest value
...     perm = sortperm(collect(zip(x,r)), by=maximum, rev=true)
...     xsorted = x[perm]
...     rsorted = r[perm]
...
...     M = length(x)
...     path = zeros(M)
...     weight = 1.0
...     num_ones = 0
...     phi = zeros(M)
...     last_val = -Inf
...     weight_scale = 1.0
...
...     for i in 1:M
...         largest_remaining = i == M ? -Inf : max(xsorted[i+1], rsorted[i+1])
...
...         if xsorted[i] >= largest_remaining
...             path[i] = 1
...             for j in 1:i
...                 if path[j] == 1
...                     phi[perm[j]] += max(last_val, xsorted[i])*weight*((num_ones+1)/i)/(num_ones+1)
...                 else
...                     phi[perm[j]] -= max(last_val, xsorted[i])*weight*((num_ones+1)/i)/(i-num_ones-1)
...                 end
...             end
...             path[i] = 0
...             weight_scale = (i-num_ones)/i
...         end
...
...         if rsorted[i] >= largest_remaining
```

```
...             path[i] = 0
...             for j in 1:i
...                 if path[j] == 1
...                     phi[perm[j]] += max(last_val, rsorted[i])*weight*((i-num_ones)/i)/num_ones
...                 else
...                     phi[perm[j]] -= max(last_val, rsorted[i])*weight*((i-num_ones)/i)/(i-num_ones)
...                 end
...             end
...             path[i] = 1
...             num_ones += 1
...             weight_scale = num_ones/i
...         end
...
...         if xsorted[i] >= largest_remaining && rsorted[i] >= largest_remaining
...             break
...         end
...
...         last_val = max(min(xsorted[i], rsorted[i]), last_val)
...         weight *= weight_scale
...     end
...
...     phi
... end
```

```
shapley_max (generic function with 1 method)
```

1.1 Validation using a brute force Shapley method

```
>>> using Iterators
...
... function shapley_weight(M, s)
...     factorial(s)*factorial(M-s-1)/factorial(M)
... end
... function brute_force_shapley(f, x, missing_values, i)
...     phi = 0
...     xtmp = zeros(length(missing_values))
...     for subset in subsets(setdiff(1:length(x), [i]))
...         xtmp[:] = missing_values
...         xtmp[subset] = x[subset]
...         val2 = f(xtmp)
...         xtmp[i] = x[i]
...         val1 = f(xtmp)
...         w = shapley_weight(length(x), length(subset))
...         phi += w*(val1-val2)
...     end
...     phi
... end
```

```
brute_force_shapley (generic function with 1 method)
```

```
>>> # test 150 random instances of varying sizes
... for i in 1:15, j in 1:10
...     x = rand(i)
...     r = rand(i)
...     diff = norm(shapley_max(x, r) .- [brute_force_shapley(maximum, x, r, i) for i in 1:length(x)])
...     @assert diff < 1e-8
... end
```