The Gory Details

Pavel Komarov

March 12, 2018

Here lies a full explanation of how multivariate Projection Pursuit Regression (PPR) and univariate Classification work, to the degree I currently understand. It is as much for me as for you, because I much prefer my own notation, and every time I have to dig in to the bones of the code and faff with the loss function, I end up having to refer to the original paper's somewhat ambiguous, derivationless equations. Formerly I was attempting to deposit some of this knowledge in code comments, but they took up too much space while somehow remaining marvelously unreadable.

This is all in LaTeXbecause native math in markdown is amazingly still not supported by github, so a `.md` would really be no less uncomfortable than code comments. If you happen to want to know how I accomplished all this formatting, the source `.tex` is also in this directory and can be compiled with `pdflatex`.

I attempt to follow a structure based on the five components of any machine learning algorithm:

1. A Task/Problem Reduction
2. A Loss Function
3. An Optimization Scheme
4. A Model
5. Data

–or at least the middle three, since by the time the data for your task meets my algorithm you'll have abstracted away concerns about where it came from and what it actually means.

1 The Model

PPR is a statistical model of the form:

$$\vec{y_i} = \sum_{j=1}^{r} f_j(\vec{x_i} \cdot \vec{\alpha_j}) \otimes \vec{\beta_j}^T$$

where:

- i iterates examples, the rows of input and output matrices
- j iterates the number of terms in the PPR "additive model"
- r is the total number of projections and functions (terms) in the PPR
- $\vec{y_i}$ is a d-dimensional vector, the ith row in an output matrix $\boldsymbol{Y} \in \mathbb{R}^{d \times n}$
- $\vec{x_i}$ is a p-dimensional vector, the ith row of an input matrix $\boldsymbol{X} \in \mathbb{R}^{p \times n}$
- $\vec{\alpha_j}$ is the jth projection vector in the mdoel, a p-dimensional vector inner-producted with x_i
- f_j is the jth function in the model, mapping from $\mathbb{R}^1 \to \mathbb{R}^1$

- $\vec{\beta_j}^T$ is the transpose of $\vec{\beta_j}$, a d-dimensional vector outer-producted with the result of f_j to yield a result in the output space
- $\cdot$ is an inner product
- $\otimes$ is an outer product

I also term this the "evaluation function". It may seem complicated, but the idea is simple:

1. Linearly project the input down to one dimension where it is easier to work with, thereby sidestepping the curse of dimensionality.
2. Find a sensible mapping from this reduced space to "residuals", linear combinations of variance in the output. This is where the nonlinearity happens.
3. Unpack from the single-dimensional residual space to the output space with a kind of inverse projection.

1.1 A Word on Additive Models

In practice a single projection-mapping-expansion is not descriptive enough to capture the richness of what may be a very complicated underlying relationship between X and Y, so it is repeated r times, each new "stage" only accounting for the variance left unexplained by the stages that have come before. Notice that, as per Taylor's Theorem and the no-doubt familiar universal approximation theorems, for certain classes of functions f, as r goes to infinity the evaluation function can approximate any continuous functional relationship between inputs and outputs.

2 The Loss Function

The (supervised) learning process consists of minimizing a standard quadratic cost function:

$$\sum_{i=1}^{n} w_i(y_i - \hat{y}_i)^2$$

where:

- i iterates all training examples
- n is the total number of training examples
- w_i the weight of the ith example
- y_i is the known answer for example i
- $\hat{y}_i$ ("y-i-hat") is the answer predicted by the model for example i

In words: get as close as you can for all examples. (TODO: There should maybe be some regularization here too. It will be handy later to have all this calculus lying around.)

Plugging the evaluation function in to the cost function yields a relationship between model parameters and cost or "loss". Because there are multiple dimensions in our vector $\vec{y_i}$, we introduce a sum over them so the PPR is motivated to make good predictions for all entries of the output:

$$loss = \sum_{i=1}^{n} w_i \sum_{k=1}^{d} w_k[y_{ik} - \sum_{j=1}^{r} f_j(\vec{x_i} \cdot \vec{\alpha_j}) \otimes \beta_{jk}]^2$$

where this new fauna:

- k iterates the columns of the output $\boldsymbol{Y}$

- d is the number of outputs, the width of the output matrix $\boldsymbol{Y}$
- w_k is a scalar weight, the relative importance of the kth output dimension
- y_{ik} is the scalar kth entry in the vector y_i, itself the ith row of $\boldsymbol{Y}$
- β_{jk} is the scalar kth entry of $\vec{\beta_j}$ from the evaluation function

The parameters we need to optimize to make the PPR "learn" are $\vec{\alpha_j}$, f_j, and $\vec{\beta_j}$. w_k are hyperparameters chosen by the user, just as r is chosen.

3 The Optimization Scheme

The macroscopic optimization scheme to solve for so many different parameters is non-obvious but straightforward:

1. Initialize all $\vec{\alpha_j}$, f_j and $\vec{\beta_j}$ to something random. Let $j = 1$.
2. Find the "residual" variance undexplained by all stages fit so far.
3. Project the input in to single dimension: $\boldsymbol{X} \cdot \vec{\alpha_j}$.
4. Fit f_j to a weighted residual target versus projections.
5. Use this f_j to find a better setting for $\vec{\beta_j}$.
6. Use a Gauss-Newton scheme to solve for an update to $\vec{\alpha_j}$.
7. Repeat steps 3-6 until f_j, $\vec{\beta_j}$, and $\vec{\alpha_j}$ converge.
8. (optional) Use the newly converged parameters to retune all previous f_t, $\vec{\beta_t}$, $\vec{\alpha_t}$ where $t \leq j$. (backfitting)
9. Increment j and go back to step 2 until j reaches r.

This is a form of *alternating optimization*, wherein all parameters except one are held constant, the best setting for that parameter given those constants is found, and the process cycled through all parameters until convergence.

But this leaves some details unexplained. How exactly is the residual found? How are parameters found given fixed solutions to the others?

3.1 Finding The Residual

The residual trick cleverly separates the contribution of the jth stage from the other terms in the additive model. Consider rephrasing the loss function as:

$$loss = \sum_{i=1}^{n} w_i \sum_{k=1}^{d} w_k [y_{ik} - \sum_{t \neq j} f_t(\vec{x_i} \cdot \vec{\alpha_t}) \otimes \beta_{tk} - f_j(\vec{x_i} \cdot \vec{\alpha_j}) \otimes \beta_{jk}]^2$$

Now if we let

$$r_{ijk} = y_{ik} - \sum_{t \neq j} f_t(\vec{x_i} \cdot \vec{\alpha_t}) \otimes \beta_{tk}$$

then

$$loss,\ L = \sum_{i=1}^{n} w_i \sum_{k=1}^{d} w_k [r_{ijk} - f_j(\vec{x_i} \cdot \vec{\alpha_j}) \otimes \beta_{jk}]^2$$

In practice we will wish to find all r_{ijk} for a particular j. Call this $\boldsymbol{R}_j \in \mathbb{R}^{n \times d}$, the same space as the output. It can be found with

$$R_j = Y - \sum_{t \neq j} f_t(\boldsymbol{X} \cdot \vec{\alpha_t}) \otimes \vec{\beta_t}$$

3.2 Optimizing $\vec{\beta_j}$ Given $\vec{\alpha_j}$ and f_j

$$loss\ for\ the\ jth\ term,\ L_j = \sum_{i=1}^{n} w_i \sum_{k=1}^{d} w_k [r_{ijk} - f_j(\vec{x_i} \cdot \vec{\alpha_j}) \otimes \beta_{jk}]^2$$

To optimize with respect to a parameter, use good ol' calculus: Take a derivative, set equal to zero, and solve. Let's select $\beta_{jk'}$, the $(k = k')$th entry of $\vec{\beta_j}$ as the parameter of interest.

$$\frac{\partial L_j}{\partial \beta_{jk'}} = \sum_{i=1}^{n} w_i w_{k'} [2(r_{ijk'} - f_j(\vec{x_i} \cdot \vec{\alpha_j}) \otimes \beta_{jk'})(-f_j(\vec{x_i} \cdot \vec{\alpha_j}))] = 0$$

Notice that the sum over k disappears because no term where $k \neq k'$ will contain our variable $\beta_{jk'}$, so for the purposes of differentiation they are constant, and the derivative of constants is zero. Only the weight $w_{k'}$ remains.

Also, I've been using $\otimes$ for consistency, but since $\beta_j k$ is scalar, it's nothing special, just an ordinary multiplication. So we can do some algebra:

$$-2w_{k'} \sum_{i=1}^{n} w_i [r_{ijk'} f_j(\vec{x_i} \cdot \vec{\alpha_j})] + 2w_{k'} \beta_{jk'} \sum_{i=1}^{n} w_i [f_j^2(\vec{x_i} \cdot \vec{\alpha_j})] = 0$$

$$\rightarrow \beta_{jk'} = \frac{\sum_{i=1}^{n} w_i [r_{ijk'} f_j(\vec{x_i} \cdot \vec{\alpha_j})]}{\sum_{i=1}^{n} w_i [f_j^2(\vec{x_i} \cdot \vec{\alpha_j})]}$$

This can be vectorized to find all entries of $\vec{\beta_j}$ at once. Notice that all $f_j(\vec{x_i} \cdot \vec{\alpha_j})$ can be stacked together in an n-vector $f_j(\boldsymbol{X} \cdot \vec{\alpha_j})$, that all weights w_i can be stacked together as an n-vector $\vec{w_I}$, that the residuals come from $\boldsymbol{R_j}$, and that the sums iterate the length-n dimensions of these objects:

$$\vec{\beta_j} = \frac{\boldsymbol{R_j^T} \cdot (\vec{w_I} \odot f_j(\boldsymbol{X} \cdot \alpha_j))}{f_j(\boldsymbol{X} \cdot \alpha_j) \cdot (\vec{w_I} \odot f_j(\boldsymbol{X} \cdot \alpha_j)}$$

where $\odot$ is a Hadamard product, and the sum disappears inside the inner products.

3.3 Optimizing f_j Given $\vec{\alpha_j}$ and $\vec{\beta_j}$

Now a similar argument, but this time consider the parameter of interest to be $f_i' j$, the ith entry of the n-vector formed by taking the inner product of $\boldsymbol{X}$ with α_j and applying f to each entry.

$$\frac{\partial L_j}{\partial f_{i'j}} = \sum_{k=1}^{d} w_k w_{i'} [2(r_{i'jk} - f_{i'j} \otimes \beta_{jk'})(-\beta_{jk})] = 0$$

The sum over i disappears because only the single term where $i = i'$ isn't constant to the derivative.

$$\rightarrow w_i \sum_{k=1}^{d} w_k [-2r_{i'jk}\beta_{jk} + 2f_{i'j}\beta_{jk}^2] = 0$$

$$\rightarrow 2f_{i'j}\sum_{k=1}^{d} w_k\beta_{jk}^2 = 2\sum_{k=1}^{d} w_k r_{i'jk}\beta_{jk}$$

$$\rightarrow f_{i'j} = \frac{\sum_{k=1}^{d} w_k r_{i'jk}\beta_{jk}}{\sum_{k=1}^{d} w_k\beta_{jk}^2}$$

As in the case of $\vec{\beta_j}$, this can be vectorized.

$$f_j(\boldsymbol{X}\cdot\vec{\alpha_j}) = \frac{\boldsymbol{R}\cdot(\vec{w_K}\odot\vec{\beta_j})}{\vec{\beta_j}\cdot(\vec{w_K}\odot\vec{\beta_j})}$$

where $\vec{w_K}$ is a vector containing all output dimension weights, w_k.

This provides targets for the funtion f. The task is to find the function that maps from this input to this output, for which there are numerous solvers (finding a polynomial by reducing to a linear inverse problem, for example). The example weights w_i disappear in the algebra and so do not affect the targets, but they can be passed on to the function-fitter so it considers some examples more important to fit than others.

3.4 Optimizing $\vec{\alpha_j}$ Given $\vec{\beta_j}$ and f_j

This is by far the toughest set of parameters to optimize, because they are nested inside the function. This time express the loss as:

$$L_j = \sum_{k=1}^{d} w_k[\vec{w_I}\odot\vec{g_{jk}}^2]$$

where

$$\vec{g_{jk}} = \vec{r_{jk}} - f_j(\boldsymbol{X}\cdot\vec{\alpha_j})\otimes\beta_{jk}$$

where $\vec{r_{jk}}$ is the n-vector formed by stacking $r_{ijk}\forall i$ together, or equivalently the kth column of $\boldsymbol{R}_j$.

The weights vector can be factored in to the square to yield a form solveable with Gauss-Newton.

$$L_j = \sum_{k=1}^{d} w_k\vec{g_{jkw}}^2$$

$$\vec{g_{jkw}} = \sqrt{\vec{w_I}}\odot(\vec{r_{jk}} - f_j(\boldsymbol{X}\cdot\vec{\alpha_j})\otimes\beta_{jk})$$

Find the Jacobian:

$$J_{jk}[u,v] = \frac{\partial g_{jkw}[u](\vec{\alpha_j})}{\partial\vec{\alpha_j}[v]} = -\sqrt{w_u}\dot{f}_j(\vec{x_u}\cdot\vec{\alpha_j})\beta_{jk}x_{uv}$$

$$\rightarrow J_{jk} = -\beta_{jk}\begin{bmatrix} \sqrt{w_0}\dot{f}_j(\vec{x_0}\cdot\vec{\alpha_j})x_{00} & \sqrt{w_0}\dot{f}_j(\vec{x_0}\cdot\vec{\alpha_j})x_{01} & \dots & \sqrt{w_0}\dot{f}_j(\vec{x_0}\cdot\vec{\alpha_j})x_{0p} \\ \sqrt{w_1}\dot{f}_j(\vec{x_1}\cdot\vec{\alpha_j})x_{10} & \sqrt{w_1}\dot{f}_j(\vec{x_1}\cdot\vec{\alpha_j})x_{11} & \dots & \sqrt{w_1}\dot{f}_j(\vec{x_1}\cdot\vec{\alpha_j})x_{1p} \\ \vdots & & \ddots & \vdots \\ \sqrt{w_n}\dot{f}_j(\vec{x_n}\cdot\vec{\alpha_j})x_{n0} & \dots & \dots & \sqrt{w_n}\dot{f}_j(\vec{x_n}\cdot\vec{\alpha_j})x_{np} \end{bmatrix}$$

$$= -\beta_{jk}(\sqrt{\vec{w_I}} \odot \dot{f}_j(\boldsymbol{X} \cdot \vec{\alpha_j})) \odot \boldsymbol{X}$$

where that last $\odot$ is with each column of $\boldsymbol{X}$ individually.

As per Gauss-Newton, the update to the parameter $\vec{\alpha_j}$ to the function g is given by the solution δ to:

$$\left[\sum_{k=1}^{d} w_k J_{jk}^T J_{jk}\right] \vec{\delta} = \sum_{k=1}^{d} w_k J_{jk}^T \vec{g_{jkw}}$$

On the left side is a $p \times p$ matrix, and on the right a $p \times 1$ vector, so we have an easy-to-solve linear inverse problem.

$$\vec{\alpha_j} = \vec{\alpha_j} + \vec{\delta}$$

4 Classification