Machine Learning, 48, 169–191, 2002

Kernel Matching Pursuit

PASCAL VINCENT Vincent@iro.umontreal.ca
YOSHUA BENGIO bengioy@iro.umontreal.ca
Department of IRO, Université de Montréal, C.P. 6128, Montreal, Qc, H3C 3J7, Canada

Editors: Yoshua Bengio and Dale Schuurmans

【算法】Matching Pursuit

张楚珩
清华大学 交叉信息院博士在读

Matching pursuit 是一种贪心算法，本科生的课程就应该学过，可惜我没有学过，现在来补课。

原文传送门

Vincent, Pascal, and Yoshua Bengio. "Kernel matching pursuit." Machine learning 48.1-3 (2002): 165-187.

特色

前面讲到了 RKHS 用于策略网络和价值函数网络，其最为重要的优势在于基于 kernel 的方法能够使用 kernel matching pursuit（KMP） 来做 sparsification，这使得模型占用的内存空间不会随着采样而持续增长。这里就具体介绍该算法。该算法与广泛适用的 Boosting（比如，算法竞赛中的效果最好的算法，gradient boosting decision tree）、SVM 等有着丰富的联系。

过程

这里先讲三种基本的 KMP 算法，然后讲一下它们的变种以及它们与其他方法的区别与联系。

1. Problem formulation

给定若干个带标签的数据点，希望从一个给定的函数字典里面找出若干个函数，用其线性加权组成一个函数，使得该函数能够尽可能拟合给定数据。

We are given l noisy observations $\{y_1, \ldots, y_l\}$ of a target function $f \in \mathcal{H}$ at points $\{x_1, \ldots, x_l\}$. We are also given a finite dictionary $\mathcal{D} = \{d_1, \ldots, d_M\}$ of M functions in a Hilbert space $\mathcal{H}$, and we are interested in sparse approximations of f that are expansions of the form

$$f_N = \sum_{n=1}^{N} \alpha_n g_n \tag{1}$$

where

- N is the number of *basis functions* in the expansion,
- $\{g_1, \ldots, g_N\} \subset \mathcal{D}$ shall be called the *basis* of the expansion,
- $\{\alpha_1, \ldots, \alpha_N\} \in \mathbb{R}^N$ is the set of corresponding *coefficients* of the expansion,
- f_N designs an approximation of f that uses exactly N distinct basis functions taken from the dictionary.

对于基底的选择来说，相当于是在 M 个函数里面选择 N 个函数，标记一下选择出来的函数指标及其被选择出来的顺序。

Notice the distinction in notation, between the dictionary functions $\{d_1, \ldots, d_M\}$ ordered as they appear in the dictionary, and the particular dictionary functions $\{g_1, \ldots, g_N\}$ ordered as they appear in the expansion f_N. There is a correspondence between the two, which can be represented by a set of indices $\Gamma = \{\gamma_1, \ldots, \gamma_N\}$ such that $g_i = d_{\gamma_i} \forall i \in \{1..N\}$ with $\gamma_i \in \{1..M\}$. Choosing a basis is equivalent to choosing a set Γ of indices.

由于我们这里只讨论在给定数据集上做拟合，不考虑函数的泛化性能，因此所有字典里面的函数都可以看做是在所给定数据点上的向量。因此，问题可以完全转化为向量-矩阵的形式。

- For any function $f \in \mathcal{H}$ we will use $\vec{f}$ to represent the l-dimensional vector that corresponds to the evaluation of f on the l training points:

$$\vec{f} = (f(x_1), \ldots, f(x_l)).$$

- $\vec{y} = (y_1, \ldots, y_l)$ is the *target vector*.
- $\vec{R}_N = \vec{y} - \vec{f}_N$ is the *residue*.
- $\langle \vec{h}_1, \vec{h}_2 \rangle$ will be used to represent the usual dot product between two vectors $\vec{h}_1$ and $\vec{h}_2$.
- $\|\vec{h}\|$ will be used to represent the usual L_2 (Euclidean) norm of a vector $\vec{h}$.

2. Basic matching pursuit algorithm

对于该问题来说，如果想得到最优的解，需要尝试所有 $\frac{M!}{N!(M-N)!}$ 种函数组合的可能性，这样的计算量显然是不可以接受的。因此这里采用贪心算法来近似解决这个问题。即每次在剩余的函数中找出一个函数，使得该函数能够最小化残差，同时确定与该函数相关的系数。

INPUT:

- data set $\{(x_1, y_1), \ldots, (x_l, y_l)\}$
- dictionary of functions $\mathcal{D} = \{d_1, \ldots, d_M\}$
- number N of basis functions desired in the expansion (or, alternatively, a validation set to decide when to stop)

INITIALIZE: current residue vector R and dictionary matrix D

$$R \leftarrow \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix} \quad \text{and} \quad D \leftarrow \begin{pmatrix} d_1(x_1) & \cdots & d_M(x_1) \\ \vdots & \ddots & \vdots \\ d_1(x_l) & \cdots & d_M(x_l) \end{pmatrix}$$

FOR $n = 1..N$ (or until performance on validation set stops improving):

- $\gamma_n \leftarrow \arg\max_{k=1..M} \left| \frac{\langle D(.,k), R \rangle}{\|D(.,k)\|} \right|$
- $\alpha_n \leftarrow \frac{\langle D(.,\gamma_n), R \rangle}{\|D(.,\gamma_n)\|^2}$
- $R \leftarrow R - \alpha_n D(.,\gamma_n)$

RESULT:

The solution found is defined by $f_N(x) = \sum_{n=1}^{N} \alpha_n d_{\gamma_n}(x)$

Figure 1. Basic matching pursuit algorithm.

该算法相当于每次在剩余的基函数集合中找到一个基函数 $D(\cdot,\gamma_n)$，使得该基函数尽量和残差方向平行，并且选取一个合适的比例系数 α_n，把残差中与该基函数平行的分量抵消。系数 α_n 的选取使得在刚刚加入 $D(\cdot,\gamma_n)$ 之后，所得到的残差与 $D(\cdot,\gamma_n)$ 垂直。

注意到，该算法中每一步选择加入的基函数不一定是最优的，同时其确定的系数也不是最优的。

3. Matching pursuit with back-fitting

前面提到 matching pursuit 每次选择加入的基函数及其系数都不是最优的。但是其实容易做到的是，每次加入一个新的基函数之后，重新计算当前一组基函数对应的最优系数，这样至少保证当前基函数组对应的系数是最优的。即，在加入新的基函数之后重新计算系数（线性回归）

$$\alpha_{1..n+1}^{(n+1)} = \underset{(\alpha_{1..n+1} \in \mathbb{R}^{n+1})}{\arg\min} \left\| \left(\sum_{k=1}^{n+1} \alpha_k \vec{g}_k \right) - \vec{y} \right\|^2 \tag{6}$$

Back-fitting 版本的算法也叫做 orthogonal matching pursuit（OMP），导师开的本科生算法课上面应该讲过，然而我都忘记了。。。

下面更加形象地解释一下 back-fitting：

考虑已经上一步已有的基函数组合 $(g_1,\cdots,g_n)$，这些基函数组合张成一个空间 $\mathcal{B}_n=\text{span}(g_1,\cdots,g_n)\subseteq\mathbb{R}^l$。记其补空间为 $\mathcal{B}_n^\perp$。记其相应的投影算子 $P_{\mathcal{B}_n},P_{\mathcal{B}_n^\perp}$。对于任意的向量都可以分解为这两个投影的向量和，即

$$\text{any } \vec{g}\in\mathbb{R}^l \text{ can be decomposed as } \vec{g}=P_{\mathcal{B}_n}\vec{g}+P_{\mathcal{B}_n^\perp}\vec{g}$$

由于我们要求已有的基函数组合 $(g_1,\cdots,g_n)$ 要能够最小化残差，因此

$$\vec{f}_n=P_{\mathcal{B}_n}\vec{y} \text{ and } \vec{R}_n=P_{\mathcal{B}_n^\perp}\vec{y}$$

当我们考虑要加入一个新的基函数 g_{n+1} 的时候，如果不考虑改变之前基函数相应的系数，即 $f_{n+1}=f_n+\alpha_{n+1}g_{n+1}$。那么新的基函数 g_{n+1} g_{n+1} 在空间 $\mathcal{B}_n^\perp$ 上的分量会尽量减小残差，但是它 $\mathcal{B}_n$ 上的分量部分一定会相应地增大残差，而这一部分完全可以通过调整 $(g_1,\cdots,g_n)$ 上的系数来抵消。如下图所示。

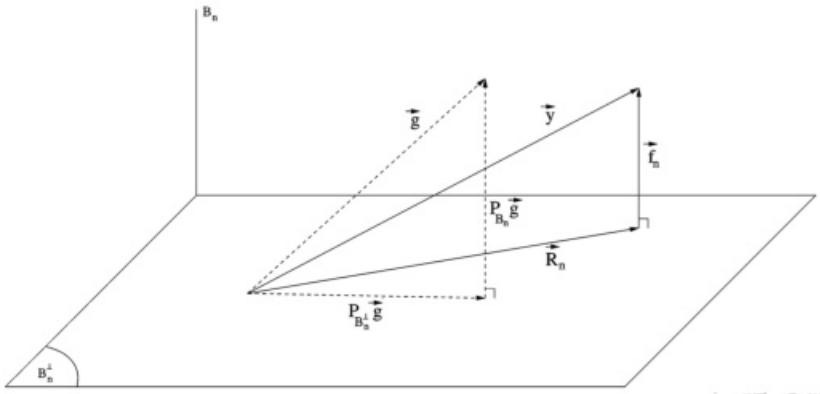

Figure 2. Geometrical interpretation of matching pursuit and back-projection.

4. Matching pursuit with pre-fitting

在 back-fitting 中，先找到新的基函数 g_{n+1}，然后再优化相应的系数。这可能导致我们找到的新基函数并不是最好的那一个。下面考虑每一步同时找到一个新的基函数以及当前基函数组的系数，即

$$\left(g_{n+1},\alpha_{1..n+1}^{(n+1)}\right)=\underset{(g\in\mathcal{D},\alpha_{1..n+1}\in\mathbb{R}^{n+1})}{\arg\min}\left\|\left(\sum_{k=1}^{n}\alpha_k\vec{g}_k\right)+\alpha_{n+1}\vec{g}-\vec{y}\right\|^2 \tag{7}$$

算法在每一步都维护两个矩阵 $D_{\mathcal{B}^\perp}$ 和 β。其中，$D_{\mathcal{B}^\perp} \in \mathbb{R}^{l \times M}$，其每一列表示相应基函数在 $\mathcal{B}_n^\perp$ 上的投影，可以看出，如果某个基函数已经被选入基函数组了，那么这一列应该为零向量；$D_{\mathcal{B}} \in \mathbb{R}^{n \times M}$，其每一列表示相应基函数在在 $\mathcal{B}_n$ 上的投影，该列的每一维代表该基函数分解到 $\mathrm{span}(g_1, \cdots, g_n)$ 上的系数。维护 $D_{\mathcal{B}^\perp}$ 的用处在于每次就能够直接找出剩余基函数里面能够最小化 (7) 式的函数；维护 β 的用处在于，加入新的基函数之后，就能够直接修正之前基函数组上的系数。算法如下。

INPUT:

- data set $\{(x_1, y_1), \ldots, (x_l, y_l)\}$
- dictionary of functions $\mathcal{D} = \{d_1, \ldots, d_M\}$
- number N of basis functions desired in the expansion (or, alternatively, a validation set to decide when to stop)

INITIALIZE:

Current residue vector R and dictionary components $D_{\mathcal{B}^\perp}$ and $D_{\mathcal{B}}$

$$R \leftarrow \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix} \quad \text{and} \quad D_{\mathcal{B}^\perp} \leftarrow \begin{pmatrix} d_1(x_1) & \cdots & d_M(x_1) \\ \vdots & \ddots & \vdots \\ d_1(x_l) & \cdots & d_M(x_l) \end{pmatrix}$$

$D_{\mathcal{B}}$ is initially empty, and gets appended an additional row at each step (thus, ignore the expressions that involve $D_{\mathcal{B}}$ during the first iteration when $n = 1$)

FOR $n = 1..N$ (or until performance on validation set stops improving):

- $\gamma_n \leftarrow \arg\max_{k=1..M} \left| \frac{\langle D_{\mathcal{B}^\perp}(.,k), R \rangle}{\|D_{\mathcal{B}^\perp}(.,k)\|} \right|$
- $\alpha_n \leftarrow \frac{\langle D_{\mathcal{B}^\perp}(.,\gamma_n), R \rangle}{\|D_{\mathcal{B}^\perp}(.,\gamma_n)\|^2}$
- the $\mathcal{B}^\perp$ component of $\alpha_n d_{\gamma_n}$ reduces the residue: $R \leftarrow R - \alpha_n D_{\mathcal{B}^\perp}(.,\gamma_n)$
- compensate for the $\mathcal{B}$ component of $\alpha_n d_{\gamma_n}$ by adjusting previous α: $(\alpha_1, \ldots, \alpha_{n-1}) \leftarrow (\alpha_1, \ldots, \alpha_{n-1}) - \alpha_n D_{\mathcal{B}}(.,\gamma_n)'$
- Now update the dictionary representation to take into account the new basis function d_{γ_n}:
  FOR $i = 1..M$ AND $i \neq \gamma_n$:
  $$\beta_i \leftarrow \frac{\langle D_{\mathcal{B}^\perp}(.,\gamma_n), D_{\mathcal{B}^\perp}(.,i) \rangle}{\|D_{\mathcal{B}^\perp}(.,\gamma_n)\|^2}$$
  $$D_{\mathcal{B}^\perp}(.,i) \leftarrow D_{\mathcal{B}^\perp}(.,i) - \beta_i D_{\mathcal{B}^\perp}(.,\gamma_n)$$
  $$D_{\mathcal{B}}(.,i) \leftarrow D_{\mathcal{B}}(.,i) - \beta_i D_{\mathcal{B}}(.,\gamma_n)$$
- $D_{\mathcal{B}^\perp}(.,\gamma_n) \leftarrow 0$
- $D_{\mathcal{B}}(.,\gamma_n) \leftarrow 0$
- $\beta_{\gamma_n} \leftarrow 1$
- $D_{\mathcal{B}} \leftarrow \begin{pmatrix} D_{\mathcal{B}} \\ \beta_1, \ldots, \beta_M \end{pmatrix}$

RESULT:

The solution found is defined by $f_N(x) = \sum_{n=1}^{N} \alpha_n d_{\gamma_n}(x)$

Figure 3. Matching pursuit with pre-fitting.

5. 拓展到非平方误差（non-squared error loss）

上述算法使用了线性空间的性质，这主要得益于最小化的误差函数为平方误差。如果希望最小化任意误差 $L(y_i, f_n(x_i))$ ，那么上述方法不再适用。相应地，我们可以考虑每次添加的基函数能够尽可能地与误差的梯度方向平行，即

$$\tilde{R}_n = \left(-\frac{\partial L(y_1, \vec{f}_n(x_1))}{\partial \vec{f}_n(x_1)}, \ldots, -\frac{\partial L(y_l, \vec{f}_n(x_l))}{\partial \vec{f}_n(x_l)} \right) \tag{8}$$

i.e. $\vec{g}_{n+1}$ is chosen such that it is most collinear with this gradient:

$$g_{n+1} = \arg\max_{g \in \mathcal{D}} \left| \frac{\langle \vec{g}_{n+1}, \tilde{R}_n \rangle}{\|\vec{g}_{n+1}\|} \right|$$

然后再选择对应的系数（basic version）

$$\alpha_{n+1} = \arg\min_{\alpha \in \mathbb{R}} \sum_{i=1}^{l} L\left(y_i, f_n(x_i) + \alpha \vec{g}_{n+1}(x_i)\right) \tag{10}$$

或者，也可以考虑使用 back-fitting

$$\alpha_{1..n+1}^{(n+1)} = \underset{(\alpha_{1..n+1}\in\mathbb{R}^{n+1})}{\arg\min} \sum_{i=1}^{l} L\left(y_i, \sum_{k=1}^{n+1} \alpha_k g_k(x_i)\right) \tag{11}$$

由于无法使用前面线性空间投影等性质，因此在这种情况下 pre-fitting 就会很耗时以至于不能适用了。相应的 back-fitting 算法如下图所示。

INPUT:

- data set $\{(x_1, y_1), \ldots, (x_l, y_l)\}$
- dictionary of functions $\mathcal{D} = \{d_1, \ldots, d_M\}$
- number N of basis functions desired in the expansion (or, alternatively, a validation set to decide when to stop)
- how often to do a full back-fitting: every p update steps
- a loss function L

INITIALIZE: current approximation $\hat{f}$ and dictionary matrix D
Notice that $\hat{f}$ here is the *current approximation vector*, which changes at every step n, so that $\hat{f}_i$ here corresponds to $f_n(x_i)$ in the accompanying text.

$$\hat{f} = \begin{pmatrix} \hat{f}_0 \\ \vdots \\ \hat{f}_l \end{pmatrix} \leftarrow 0 \quad \text{and} \quad D \leftarrow \begin{pmatrix} d_1(x_1) & \cdots & d_M(x_1) \\ \vdots & \ddots & \vdots \\ d_1(x_l) & \cdots & d_M(x_l) \end{pmatrix}$$

FOR $n = 1..N$ (or until performance on validation set stops improving):

- $\tilde{R} \leftarrow \begin{pmatrix} -\frac{\partial L(y_1, \hat{f}_1)}{\partial f_1} \\ \vdots \\ -\frac{\partial L(y_l, \hat{f}_n)}{\partial f_l} \end{pmatrix}$
- $\gamma_n \leftarrow \underset{k=1..M}{\arg\max} \left| \frac{\langle D(.,k), \tilde{R} \rangle}{\|D(.,k)\|} \right|$
- If n *is not* a multiple of p do a simple line minimization:

$$\alpha_n \leftarrow \underset{\alpha\in\mathbb{R}}{\arg\min} \sum_{i=1}^{l} L(y_i, \hat{f}_i + \alpha D(i, \gamma_n))$$

  and update $\hat{f}$: $\hat{f} \leftarrow \hat{f} + \alpha_n D(., \gamma_n)$
- If n *is* a multiple of p do a full back-fitting (for ex. with gradient descent):

$$\alpha_{1..n} \leftarrow \underset{\alpha_{1..n}\in\mathbb{R}^n}{\arg\min} \sum_{i=1}^{l} L(y_i, \sum_{k=1}^{n} \alpha_k D(i, \gamma_k))$$

  and recompute $\hat{f} \leftarrow \sum_{k=1}^{n} \alpha_k D(., \gamma_k)$

RESULT:

The solution found is defined by $f_N(x) = \sum_{n=1}^{N} \alpha_n d_{\gamma_n}(x)$

Figure 4. Back-fitting matching pursuit algorithm with non-squared loss.

文章后面还说明了不同的算法其实对应了这里不同的损失函数，能够与之相联系的有 SVM、Adaboost 等。

发布于 2019-07-01

强化学习 (Reinforcement Learning)

文章被以下专栏收录

强化学习前沿
读呀读paper