ANOVA decomposition of a Gaussian Process model

November 23, 2020

1 ANOVA docomposition of a general function

Functional ANOVA decomposition represents a high-dimensional function as a function of the form

$$f(x_1, x_2, \dots, x_D) = f_0 + \sum_{i=1}^{D} f_i(x_i) + \sum_{i<j} f_{ij}(x_i, x_j) + \sum_{i<j<k} f_{ijk}(x_i, x_j, x_k) + \dots + f_{1,\dots,D}(x_1, \dots, x_D).$$

Let's first talk about how to compute the ANOVA components in general and then we focus on how to do it for $f(x_1, \dots, x_D)$ which is evaluated as a mean of a Gaussian process.

Note: without the loss of generality, we assumy that the input is restricted to a hypercube $[0,1]^D$. It will significanlty simplify the notation. We can normalize the input whe computing.

The first step is to choose the projection operator P:

$$Pf := \int_{[0,1]} f(x) d\mu(x)$$

We are going to use the projection operator P using Lebegue measure $Pf := \int_{[0,1]} f(x)dx$, so all intergrals should work out as expected.

1.1 Two-dimensional case

Now we use the projection operator to define the constant and the main effects. We assume $D = 2$ for now and then introduce some more notation to generalize:

$$f_0 = \int_{[0,1]} \int_{[0,1]} f(x_1, x_2) dx_1 dx_2 \tag{1}$$

$$f_1(x_1) = \int_{[0,1]} \Big(f(x_1, x_2) - f_0 \Big) dx_2 \tag{2}$$

$$f_2(x_2) = \int_{[0,1]} \Big(f(x_1, x_2) - f_0 \Big) dx_1 \tag{3}$$

The interaction effect $f_{1,2}(x_1, x_2)$ is defined as the remainder to make the ANOVA decomposition to work out correctly:

$$f_{1,2}(x_1, x_2) = f(x_1, x_2) - f_0 - f_1(x_1) - f_2(x_2).$$

The **total variance** (TV) of the predictor is defined as

$$\sigma^2(f) := \int_{[0,1]} \int_{[0,1]} (f(x_1, x_2) - f_0)^2 dx_1 dx_2$$

One can show that TV is decomposible into the sum of variances of main effects and interactions defined above:

$$\sigma_1^2(f_1) := \int_{[0,1]} (f_1(x_1))^2 dx_1 \tag{4}$$

$$\sigma_2^2(f_2) := \int_{[0,1]} (f_2(x_2))^2 dx_2 \tag{5}$$

$$\sigma_{1,2}^2(f_{1,2}) := \int_{[0,1]} \int_{[0,1]} (f_{1,2}(x_1, x_2) - f_0 - [f_1(x_1) - f_0] - [f_2(x_2) - f_0])^2 \, dx_1 dx_2 \tag{6}$$

$$\sigma^2(f) = \sigma_1^2(f_1) + \sigma_2^2(f_2) + \sigma_{1,2}^2(f_{1,2}). \tag{7}$$

And so, by dividing individual variances by TV we can express these compoents as percentages.

1.2 General case

Using subsets $u \subseteq \{1, \ldots, D\}$, we can establish a shorthand notation for ANOVA components, where f_u and x_u represents a subset of vector x with components $x_i, i \in u$. The we have

$$f(x_1, \ldots, x_D) = \sum_{u \subseteq \{1 \ldots, D\}} f_u(x_u), \tag{8}$$

$$f_u(x_u) = \int_{[0,1]^{D-|u|}} \Big(f(x) - \sum_{v \subsetneq u} f_v(x_v)\Big) dx_{-u} \tag{9}$$

$$\sigma^2(f_u) = \int_{[0,1]^D} \Big(f_u(x_u)\Big)^2 dx \tag{10}$$

$$\sigma^2(f) = \int_{[0,1]^D} \Big(f(x) - f_0\Big)^2 dx = \sum_{u \subseteq \{1,\ldots,D\}, u \neq \emptyset} \sigma^2(f_u). \tag{11}$$

1.3 ANOVA for Gaussian process

For Bayesian optimization we are using the Gaussian proces defined by a parametrized RBF kernel of the form

$$K(x, x' | \theta_0, \vec{\theta}_1) = \theta_0^2 \exp\Big(-(x - x')^t D(\vec{\theta}_1)(x - x'))\Big),$$

where $[D(\vec{\theta}_1)]_{ij} = (\theta_{1,i}^2 + \epsilon)\delta_{ij}$ is a diagonal matrix of scaling coefficients, ϵ beeing a constant and δ_{ij} beeing Dirac-delta.

$$K(x, x' | \theta_0, \vec{\theta}_1) = \theta_0^2 \exp\Big(-\sum_{d=1}^{D} (\theta_{1d}^2 + \epsilon)(x_d - x_d')^2\Big) = \theta_0^2 \prod_{d=1}^{D} \exp\Big(-(\theta_{1d}^2 + \epsilon)(x_d - x_d')^2\Big).$$

I drop θ's from the definition of K for shorthand.

Let Σ be the correlation matrix of the training set $\{(x_i, y_i)\}_{i=1}^n$ and $k(x)$ a vector with correlations to the new point x, to evaluate the functional given by the GP and get the mean at the points x we do:

$$\begin{aligned}
\Sigma &:= \{K(x_i, x_j)\}_{ij} &(12)\\
k(x) &:= \{K(x, x_i)\}_{i=1}^n &(13)\\
f(x) &:= k(x)^t \Sigma^{-1} y &(14)\\
&= \sum_{i=1}^n K(x, x_i)\Sigma_i^{-1} y, &(15)
\end{aligned}$$

where Σ_i^{-1} is the i'th row of the inverse correlation matrix (of course, we are just solving $\Sigma z = y$ and take the i'th component of z, so I will just use z_i as a shorthand:

$$f(x) = \sum_{i=1}^n z_i K(x, x_i)$$

I'll use wolframalpha to derive integrands. We also define a shorthand

$$c_d := \sqrt{\theta_{1d}^2 + \epsilon}$$

For the constant we have:

$$\begin{aligned}
f_0 &= \int_{[0,1]^D} f(x)dx = \int_{[0,1]^D} \sum_{i=1}^n z_i K(x, x_i) dx &(16)\\
&= \sum_{i=1}^n z_i \theta_0^2 \prod_{d=1}^D \int_0^1 \exp\Big(-c_d^2 (x_d - x_{id})^2\Big) dx_d &(17)\\
&= \theta_0^2 \sum_{i=1}^n z_i \prod_{d=1}^D \frac{\sqrt{\pi}\Big[\operatorname{erf}\Big(c_d - c_d x_{id}\Big) + \operatorname{erf}\Big(c_d x_{id}\Big)\Big]}{2c_d} &(18)
\end{aligned}$$

For the total variance we have:

$$\sigma^2(f) = \int_{[0,1]^D} \Big(f(x) - f_0\Big)^2 dx = \int_{[0,1]^D} \Big(\sum_{i=1}^n z_i K(x, x_i) - f_0\Big)^2 dx \tag{19}$$

$$= \int_{[0,1]^D} \sum_{i=1}^n \sum_{j=1}^n z_i z_j K(x, x_i) K(x, x_j) dx - 2 f_0 \int_{[0,1]^D} \sum_{i=1}^n z_i K(x, x_i) dx + f_0^2 \int_{[0,1]^D} dx \tag{20}$$

$$= f_0^2 \Big(1 - 2\Big) + \theta_0^4 \sum_{i=1}^n \sum_{j=1}^n z_i z_j \prod_{d=1}^D \int_0^1 \exp\Big(-c_d^2 \Big[(x_d - x_{id})^2 + (x_d - x_{jd})^2\Big]\Big) dx_d \tag{21}$$

$$= \theta_0^4 \sum_{i=1}^n \sum_{j=1}^n z_i z_j \prod_{d=1}^D \frac{\sqrt{\frac{\pi}{2}} \exp\Big(-1/2 c_d^2 (x_{id} - x_{jd})^2\Big) \Big[\operatorname{erf}\Big(\frac{c_d(x_{id}+x_{jd})}{\sqrt{2}}\Big) - \operatorname{erf}\Big(\frac{c_d(-2+x_{id}+x_{jd})}{\sqrt{2}}\Big)\Big]}{2 c_d} - f_0^2 \tag{22}$$

For the main effects we have:

$$f_t(x_t) = \theta_0^2 \sum_{i=1}^n z_i \exp\Big(-c_t^2 (x_t - x_{it})^2\Big) \prod_{d \neq t}^D \frac{\sqrt{\pi}\Big[\operatorname{erf}\Big(c_d(1 - x_{id})\Big) - \operatorname{erf}\Big(c_d(-x_{id})\Big)\Big]}{2 c_d} - f_0 \tag{23}$$

$$=: \theta_0^2 \sum_{i=1}^n z_i \exp\Big(-c_t^2 (x_t - x_{it})^2\Big) P_i - f_0 \tag{24}$$

$$\sigma^2(f_t(x_t)) = \int_{[0,1]^D} \Big(f_d(x_d)\Big)^2 dx = \int_{[0,1]^D} \Big(\theta_0^2 \sum_{i=1}^n z_i \exp\Big(-c_t^2 (x_t - x_{it})^2\Big) P_i - f_0\Big)^2 dx \tag{25}$$

$$= \int_{[0,1]^D} \theta_0^4 \sum_{i=1}^n \sum_{j=1}^n z_i z_j P_i P_j \exp\Big(-c_t \Big[(x_t - x_{it})^2 + (x_t - x_{jt})^2\Big]\Big) \tag{26}$$

$$- 2\theta_0^2 f_0 \sum_{i=1}^n z_i P_i \int_{[0,1]^D} \exp\Big(-c_t^2 (x_t - x_{it})^2\Big) dx + f_0^2 \int_{[0,1]^D} dx \tag{27}$$

$$= \theta_0^4 \sum_{i=1}^n \sum_{j=1}^n z_i z_j P_i P_j \frac{\sqrt{\frac{\pi}{2}} e^{-1/2 c_t^2 (x_{it} - x_{jt})^2} \Big[\operatorname{erf}\Big(\frac{c_t(x_{it}+x_{jt})}{\sqrt{2}}\Big) - \operatorname{erf}\Big(\frac{c_t(-2+x_{it}+x_{jt})}{\sqrt{2}}\Big)\Big]}{2 c_t} \tag{28}$$

$$- 2\theta_0^2 f_0 \sum_{i=1}^n z_i P_i \frac{\sqrt{\pi}\Big[\operatorname{erf}\Big(c_t(1 - x_{it})\Big) + \operatorname{erf}\Big(c_t(x_{it})\Big)\Big]}{2 c_t} + f_0^2 \tag{29}$$

2 Numeric verification

[1]:
```
import sklearn
from sklearn.gaussian_process.kernels import RBF
from sklearn.datasets import make_regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import pairwise_kernels
import numpy as np
```

```python
from scipy.special import erf
from scipy.integrate import dblquad
```

[2]:
```python
## Define GP and use normalization
dim = 2
theta_0 = 0.3
theta_1 = np.array([2.0]*dim)
  = 1e-4
c = np.sqrt(theta_1**2 +  )
kernel_test = lambda x, y: theta_0 * np.exp(-(x-y)**2@c**2)
length_scale = 1.0/(np.sqrt(2)*c)
kernel = RBF(length_scale = length_scale)
```

[3]:
```python
X, y = make_regression(n_samples=20, n_features=dim, n_informative=dim, bias=0.
↪5, random_state=1000)

X_norm = MinMaxScaler().fit_transform(X)

X_train = X_norm[:10, :]
y_train = y[:10]
X_test = X_norm[10:,:]
y_test = y[10:]
```

[4]:
```python
K = theta_0**2*pairwise_kernels(X_train, metric=kernel)
z = np.linalg.solve(K, y_train)

def kernel_eval(X):
    return theta_0**2*pairwise_kernels(X, X_train, metric=kernel) @ z

K_test = pairwise_kernels(X_train, metric=kernel_test)
z_test = np.linalg.solve(K_test, y_train)
def kernel_test_eval(X):
    return pairwise_kernels(X, X_train, metric=kernel_test) @ z
```

[5]:
```python
y_eval = theta_0**2 * pairwise_kernels(X_test, X_train, metric=kernel) @ z
print(y_eval)
print(kernel_eval(X_test))
print(kernel_test_eval(X_test))
```

```
[-61.30361206 -57.98808062  22.53900295 -24.28231192 -16.72602774
  -2.12929409 -41.86531635  44.5630799   41.67531984 -12.97457399]
[-61.30361206 -57.98808062  22.53900295 -24.28231192 -16.72602774
  -2.12929409 -41.86531635  44.5630799   41.67531984 -12.97457399]
[-204.34537353 -193.29360207   75.13000982  -80.94103974  -55.7534258
   -7.09764696 -139.55105451  148.54359967  138.9177328   -43.24857997]
```

[6]:
```python
np.prod(np.sqrt(np.pi)/(2*c)* erf(c*(1-X_train)) + erf(c*X_train), axis=1)
```

[6]: array([0.91226262, 0.97676297, 1.35994095, 1.25343965, 1.52405704,
        1.51234951, 1.08494107, 1.18481574, 0.67870414, 1.37359053])

[7]:
```
np.sqrt(np.pi)/(2*c)* (erf(c*(1-X_train)) + erf(c*X_train))
```

[7]:
```
array([[0.74524907, 0.57208312],
       [0.64781443, 0.60612862],
       [0.74680421, 0.72391626],
       [0.74265738, 0.70187627],
       [0.7429866 , 0.74143716],
       [0.74660759, 0.73705457],
       [0.67494038, 0.54701156],
       [0.67883566, 0.74305994],
       [0.62949642, 0.50133195],
       [0.70611856, 0.5676102 ]])
```

2.1 Contant f_0

[8]:
```
# Constant analytic
f_0 = theta_0**2 * z.T @ np.prod(np.sqrt(np.pi)/(2*c)* (erf(c*(1-X_train)) +
 ↪erf(c*X_train)), axis=1)
f_0
```

[8]: -2.1358132079939054

[9]:
```
# Constant MC
for n in 10**np.array([1,2,3,4,5, 6, 7]):
    X = np.random.uniform(size=2*n).reshape((-1,dim))
    sol = np.mean(kernel_eval(X))
    print('%10d value = %.6f rel_error = %.6f' % (n, sol, np.abs((sol-f_0)/f_0))
```

```
        10 value = -12.693817 rel_error = -4.943318
       100 value = 4.239582 rel_error = -2.984997
      1000 value = -5.003225 rel_error = -1.342539
     10000 value = -1.203545 rel_error = -0.436493
    100000 value = -2.029017 rel_error = -0.050002
   1000000 value = -2.113715 rel_error = -0.010347
  10000000 value = -2.142167 rel_error = -0.002975
```

2.2 Total variance $\sigma^2(f)$

[47]:
```
def prod_ij(i, j ):
    return z[i] * z[j] * np.prod(np.sqrt(np.pi/2)/(2*c)
              * np.exp(-0.5*c**2*(X_train[i,:]- X_train[j, :])**2)
             * (erf(c/np.sqrt(2)*(X_train[i,:] + X_train[j, :]))
```

```
                - erf(c/np.sqrt(2)*(X_train[i,:] + X_train[j, :]-2))) )
s = 0
for i in range(X_train.shape[0]):
    for j in range(X_train.shape[0]):
        s += prod_ij(i, j)
```

[48]:
```
sigma2 = -f_0**2 + theta_0**4* s
sigma2
```

[48]: 1775.787792007982

[49]:
```
# Constant MC
for n in 10**np.array([1,2,3,4,5, 6, 7]):
    X = np.random.uniform(size=2*n).reshape((-1,dim))
    sol = np.mean((kernel_eval(X)-f_0)**2)
    print('%10d value = %.6f rel_error = %.6f' % (n, sol, np.abs(sol-sigma2)/
 ↪sigma2))
```

```
        10 value = 1405.138943 rel_error = 0.208724
       100 value = 1822.151065 rel_error = 0.026109
      1000 value = 1768.487300 rel_error = 0.004111
     10000 value = 1760.572574 rel_error = 0.008568
    100000 value = 1773.654062 rel_error = 0.001202
   1000000 value = 1775.426417 rel_error = 0.000204
  10000000 value = 1776.453419 rel_error = 0.000375
```

2.3 Main effects f_d

[ ]:
```

```