Random Fourier Features

石川 徹也　<tiskw111@gmail.com>

2020 年 3 月 3 日

はじめに

本文書は Rahimi *et al.* [1] によって 2007 年に提案された Random Fourier Features（以下，RFF と略す）について，カーネル法をある程度ご存じの読者を対象に解説した文書です．RFF は，端的に言えばカーネル法におけるカーネル関数を有限次元近似する手法であり，カーネル法をより高速に，そして大規模なデータセットに適用することができるようになる手法です．

本文書では，記号の確認も含めてカーネル法について簡単にご説明はしていますが，初めてカーネル法に触れる読者には不十分極まりないと思います．カーネル法をご存じない方は，事前に赤穂 [3] や Bishop [4] などに目を通して頂くことをお薦めします．

また，本文書は意欲的な高校生や大学生諸君にとっても有用と思います．大学初年度の教養数学をきちんと修めていれば，本文書を理解することはさほど困難ではありません．少し背伸びをすれば高校生諸君でも十分に理解できる内容です．さらに RFF は，2007 年に提案され，すでに 2000 件近くの引用がなされ，現在でも改良が続けられている手法です．すでに 10 年以上昔の成果ですので，変化の激しい機械学習の分野ではもはや新しい結果とは言えません．それでも学生の皆さんが普段勉強している内容と比較すれば極めて最近の，そして現在でも研究対象として注目され続けている結果です．大昔の結果ばかり勉強させられ，世界の最先端がどこにあるのか分からなくなってしまった方々にとって，この文書は良いきっかけを提供できるかもしれません．

また，本文書の最後では，深層学習との関係性についても簡単に触れています．というのも，本文書を書くきっかけは

「カーネルサポートベクターマシンを深層学習のフレームワークで実装したいんだが，どうすりゃ良いんだコノヤロー！」

という質問（挑発？）を職場の同僚から受けたことに端を発しているためです．この問いもまた RFF を用いることで肯定的に解決できます．

何はともあれ，まずは RFF の目的と意義から順に見ていくことにしましょう．

> NOTE: 同僚の名誉のために述べておくと，上述の深層学習のフレームワークとは Tensorflow や PyTorch のような有名なワークステーション向けフレームワークではなく，組み込み向けのマイナーなものです．このフレームワークは，ニューラルネットワークの実装を主目的に設計されていること，組み込み向けゆえにサポートされている機能が極めて限定的なことから，カーネルサポートベクターマシンのようなニューラルネットワークでない機能を実装するのは容易ではないという事情があります．Tensorflow や PyTorch のような柔軟性の高いフレームワークであれば RFF を用いずとも実現は簡単です．

1　なぜ RFF が必要なのか？

機械学習におけるクラス分類問題を解く手法のひとつとして，カーネルサポートベクターマシンと呼ばれる手法がある．これは通常のサポートベクターマシン（ここではこれを線形サポートベクターマシンと呼ぶことにしよう）にカーネル法を適用することで柔軟性を格段に向上させたものであり，現在でも現役で使用されている，カーネル法の応用例のひとつでもある．しかし，カーネルサポートベクターマシンには，学習や推論にかかる時間が学習データ数に多項式的に比例してしまうという決定的な弱点がある．具体的には，学習データの数を N とすると，学習に要するステップ数は $O(N^3)$，推論に要するステップ数は $O(N)$ である．繰り返しになるが N は学習データの数である．これは，データの質よりも量の多さをよしとする機械学習の分野において，致命的とも言える弱点である．この恐しさがお分かり頂けるであろうか？

想像して頂きたい．何らかのクラス分類器を作成したところ，性能があまり出なかったとしよう．そうなったとき，学習データセットの拡充は間違いなく優先度の高い検討事項である．しかし，先に述べた通り，カーネルサポートベクターマシンの場合は学習データの数を増やせば増やすほど学習や推論にかかる時間が増えてしまう．学習時間が増えるのはまだ許容範囲内であろうが，推論の時間まで増加してしまうのは許容し難いのではないだろうか．もちろんケースバイケースではあるのだが，実応用では推論時間の上限は事前に決められており，また推論を行うデバイスも変更できない場合が多く，推論の計算量の増加は致命的となるケースが多い．つまり，カーネルサポートベクターマシンの場合は安易に学習データセットの追加ができないのである．

ちなみに，この弱点は線形サポートベクターマシンでは起こりにくい．線形のサポートベクターマシンの場合，少なくとも推論時間は学習データセットの数によらず一定である．つまり，この弱点はカーネル法によって高い柔軟性を得たことに由来するものなのである．とは言え，線形サポートベクターマシンでは性能が出ないケースが多いのも事実であるから，何とかしてカーネルサポートベクターマシンを高速化し，大量の学習

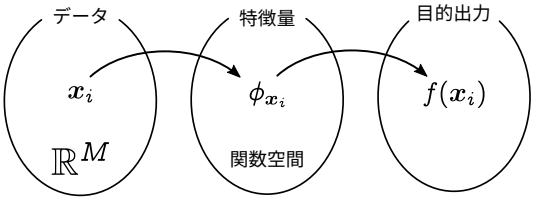

図 1　カーネル法と特徴抽出

データセットを適用したいものである．

この弱点を解消するために考え出されたのが，カーネル関数の有限次元近似であり，その中で最も成功しているのが RFF である．

これをキチンと説明するために，まずはカーネル法について簡単にご説明していこう．

2　カーネルとは？

本節ではカーネル法の概要を，数学的証明を用いずに説明してみたいと思う．本節でカーネル法を説明する主な目的は，次節で行う RFF の解説において事前に記号の理解を持って頂くことである．厳密さが多少なりとも犠牲になるとについてはご容赦頂きたい．

やや深層学習に寄ったものの見方ではあるが，機械学習は主に「データから特徴量を抽出する」ステップと「その特徴量を用いてタスクを行う」ステップに大別される．図 1 を参照されたい．以下ではこの 2 つのステップを踏むことでカーネル法を概観してみよう．

2.1　無限次元の特徴抽出と特徴関数

入力データの集合を $\mathcal{D} \triangleq \{\boldsymbol{x}_i \in \mathbb{R}^M \,\big|\, i = 1, 2, \ldots, M\}$ とおく．カーネル法は，入力データ $\boldsymbol{x}$ から特徴量として関数，すなわち無限次元の特徴量を抽出する手法である．このデータから無限次元の特徴量を抽出する関数，ここれはそれを特徴関数と呼ぶこととするが，これを $\phi(\boldsymbol{x}, \boldsymbol{p})$ で表すこととしよう．具体例を挙げるとすれば，例えば

$$\phi(\boldsymbol{x}, \boldsymbol{p}) \triangleq \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{p}\|^2}{2\sigma^2}\right), \tag{1}$$

である．これは有名なカーネル関数である RFB カーネル (*radial basis function kernel*) に対応する特徴関数である．

> NOTE: 後に説明するが，カーネル関数と特徴関数は**全くの別物**であるので注意されたい．カーネル関数は特徴関数同士の内積として定義されるもので，特徴関数とは立場上全くの別物である．ただ，RBF カーネルを含む有名どころのカーネル関数は結果として再生性をみたすゆえに特徴関数とカーネル関数の関数形が一致し，説明する側としてはヤヤコシいのだが... (^^;

注意して欲しいのは，特徴関数は入力データ $\boldsymbol{x}$ を受け取り，$\boldsymbol{p}$ に関する関数を返す関数である．そういう意味で，特徴関数は $\phi(\boldsymbol{x}, \boldsymbol{p})$ と書くよりも，$\phi_{\boldsymbol{x}}(\boldsymbol{p})$ と書いた方が分かり良いであろう．やや通常の関数表記から逸脱するが，以下では $\phi_{\boldsymbol{x}}(\boldsymbol{p})$ の表記を用いることとする．また，関数であることを特に強調したい場合は，引数の $\boldsymbol{p}$ を省略して $\phi_{\boldsymbol{x}}$ と書く．

> NOTE: 余談ではあるが，関数である $\phi_{\boldsymbol{x}}$ を特徴「量」と呼称して良いものであろうか？これはやや疑義を生ずる点であろうが，著者は量と呼称して問題ないと考えている．関数は L^2 空間に代表されるベクトル空間を構成する要素であるから，間違いなくベクトル「量」である．

2.2　特徴関数からのタスク実施

さて，データ $\boldsymbol{x}$ から特徴量 $\phi_{\boldsymbol{x}}$ が得られたところで，次はこの特徴量を用いて回帰や分類などのタスクを行うことを考えていこう．タスクの実施の仕方はタスクによって様々なのだが，最も一般的なアプローチは

$$f(\boldsymbol{x}) = \int \cdots \int_{\mathbb{R}^M} w(\boldsymbol{p}) \phi_{\boldsymbol{x}}(\boldsymbol{p}) \, \mathrm{d}p_1 \cdots \mathrm{d}p_M, \tag{2}$$

という形で関数を構築する方法である．回帰がしたいのであれば $f(\boldsymbol{x})$ が目的の曲線を追従するように，分類がしたいのであれば $f(\boldsymbol{x})$ が対数尤度を出力するようにすれば良い．ただし $w(\boldsymbol{p})$ は学習パラメータに相当する関数である．この関数を適切に学習させることで目的のタスクを実現するのがカーネル法である．

> NOTE: 式 (2) が分かりづれぇ！と思ったそこの貴方．是非とも関数解析を習得されることをお薦めする．式 (2) はただの関数同士の「内積」であり，$f(\boldsymbol{x}) = \langle w, \phi_{\boldsymbol{x}} \rangle$ と表記することもできる．つまりやっていることは線形回帰と何も変わらず，ただ扱う空間がベクトル空間から関数空間に変わっただけのことである．

もちろん，最適化パラメータが関数であるような最適化問題は数値計算問題として解きにくいので，様々なテクニックを駆使してこれを有限次元の最適化問題に落し込み，その最適化を問題を解くことで学習を行うのである．

メデタシ，メデタシ...

2.3　いやいや，カーネル関数はどこよ？

え，カーネル関数はいつ出てくるのかって？実はカーネル関数は，最適化問題をキチンと定式化し，その式をコネクリ回さないと出てこない．本節ではその部分の理解にあまり体力を使いたくないので，先人が数式をコネクリ回した結果，以下が得られたということだけ指摘しておくに留めておく．詳細については赤穂 [3] や Bishop [4] などを参照されたい．

1. 関数 w の最適化は有限次元のベクトルの最適化に落し込むことが出来る．ただしそのベクトルの次元は学習データ数 N である．この関数の最適化問題を有限次元の最適化問題に落し込むテクニックは，カーネルトリックと呼ばれる．
2. 上述 1 の最適化を解く上でも，最適化の結果を用いて推論を行う上でも，実は特徴関数 $\phi_{\boldsymbol{x}}$ は陽には出てこない．出てくるのは次の式

$$k(\boldsymbol{x}, \boldsymbol{y}) \triangleq \int \cdots \int \phi_{\boldsymbol{x}}(\boldsymbol{p}) \phi_{\boldsymbol{y}}(\boldsymbol{p}) \, \mathrm{d}p_1 \cdots \mathrm{d}p_M, \tag{3}$$

で与えられる関数 k だけである．この関数 k をカーネル関数と呼ぶ．

さて，上記には 2 つの重要な事実が含まれている．ひとつは，カーネルトリックによって関数次元の最適化問題を有限次元の

最適化問題に落とし込めたとはいえ，いまだに学習データ数 N と同じだけの次元を持っているという点である．これが本文書の冒頭で述べたカーネル法の弱点に直結しているのである．

もうひとつは，学習や推論を行う上で特徴関数は陽には出てこず，カーネル関数しか出てこないという点である．ゆえに，一般的にはカーネル関数だけを設計し，特徴ベクトルには言及しない場合が多い．ちなみに，式 (1) にて特徴関数の具体例をお見せしたが，これに対応するカーネル関数は

$$k(\boldsymbol{x},\boldsymbol{y}) = \exp(-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|^2), \quad \gamma = -\frac{1}{4\sigma^2}, \tag{4}$$

である．これは式 (3) を直接計算することで得られる．

さて，様々な式展開や証明を省略させて頂いたが，これでカーネル法のおさらいは終わりにして，早速 RFF の定式化に移ろう．

NOTE: カーネル関数の説明が適当すぎないかって？ご指摘の通りであるが，本文書としてはカーネル法の説明は前座にすぎないので，本節でパルプを 3 枚も 4 枚も費すことはしたくない．もちろんカーネル関数がいきなり天から降ってくるのは本文書の想定読者を考えると論外である．もっとスマートな導入の仕方があれば是非ご連絡頂きたい．

3 カーネルの有限近似

3.1 実確率変数の期待値

突然ではあるが，確率変数の期待値について簡単に説明をさせて頂きたい．まずは簡単のため，集合 $\{z_1, z_2, \ldots, z_I\}$ 上のいずれかの値を取る離散確率変数 Z を考える．この確率変数 Z の期待値は

$$\mathrm{E}[X] = \sum_{i=1}^{I} p_i z_i, \tag{5}$$

で与えられる．ただし p_i は Z が値 z_i を取る確率である．連続確率変数でも同じことが言える．確率変数 X をとある確率空間上で可測な実数値確率変数とし，X が実数値 x を取る確率密度関数を $p(x)$ とおく．このとき確率変数 X の期待値は

$$\mathrm{E}[X] = \int_{\mathbb{R}} p(x)x \,\mathrm{d}x, \tag{6}$$

で与えることができる．まさに離散確率変数の期待値を連続化したような式になっている．ちなみに確率変数 X が M 次元のベクトル確率変数であった場合は

$$\mathrm{E}[X] = \int\cdots\int_{\mathbb{R}} p(\boldsymbol{x})\boldsymbol{x} \,\mathrm{d}x_1 \cdots \mathrm{d}x_M, \tag{7}$$

となる．

さらに，式 (6) の右辺を解析的に解くことが困難であったとしよう．このとき我々には何が出来るであろうか．そう，期待値なのであるから，実際に確率変数 X をいくつか生成してみて，その平均値をもって期待値の近似値とすることができる．これはいわゆる Monte Carlo 法であり，数式で表現すると

$$\mathrm{E}[X] \approx \frac{1}{L}\sum_{\ell=1}^{L} X_\ell, \tag{8}$$

となる．ただし X_ℓ は確率変数 X のサンプリングである．

何でいきなりこんな話をしたかって？マァマァ，黙って式 (6), (7) を目に焼き付けておいて頂きたい．

3.2 カーネルの Fourier 変換

さて，本題の RFF に話を戻そう．以下ではカーネル関数 $k(\boldsymbol{x},\boldsymbol{y})$ を $k(\boldsymbol{x}-\boldsymbol{y})$ と表現できるものだけに限定しよう．多項式カーネルや Fisher カーネルなど多くの有名なカーネル関数が議論の対象外になってしまうものの，式 (4) で与えられる RBF カーネルは幸いにも $k(\boldsymbol{x}-\boldsymbol{y})$ はそのように書き表すことができる．

さて，カーネル関数 $k(\boldsymbol{x}-\boldsymbol{y})$ を Fourier 変換し，その結果を $p(\boldsymbol{\omega})$ と置いてみよう．すなわち

$$\begin{aligned} p(\boldsymbol{\omega}) &= \mathcal{F}[k](\boldsymbol{\omega}) \\ &= \int\cdots\int_{\mathbb{R}^M} k(\boldsymbol{z}) e^{-i\boldsymbol{\omega}^\mathsf{T}\boldsymbol{z}} \,\mathrm{d}z_1 \cdots \mathrm{d}z_M, \end{aligned} \tag{9}$$

である．これを逆 Fourier 変換を用いて $k(\boldsymbol{x}-\boldsymbol{y}) = \cdots$ の形に書き直すと

$$\begin{aligned} k(\boldsymbol{x}-\boldsymbol{y}) &= \mathcal{F}^{-1}[p](\boldsymbol{x}-\boldsymbol{y}) \\ &= \int\cdots\int_{\mathbb{R}^M} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\mathsf{T}(\boldsymbol{x}-\boldsymbol{y})} \,\mathrm{d}\omega_1 \cdots \mathrm{d}\omega_M, \end{aligned} \tag{10}$$

が成り立つ．これは逆 Fourier 変換の定理から明らかであろう．

さて，式 (10) を見て思うことはないだろうか．そう，期待値である．つまり p を確率密度関数とみなせば

$$k(\boldsymbol{x}-\boldsymbol{y}) = \mathrm{E}_{\boldsymbol{\omega}\sim p}\left[e^{i\boldsymbol{\omega}^\mathsf{T}(\boldsymbol{x}-\boldsymbol{y})}\right], \tag{11}$$

と表すことができる．ただし，上式の期待値は $\boldsymbol{\omega}$ を確率変数とみなし，さらにその確率分布は確率密度関数 $p(\boldsymbol{\omega})$ に従うとみなしていることに注意せよ．期待値記号 E の添字に $\boldsymbol{\omega}\sim p$ を書いたのは，そのことを明記するためである．

先に指摘した通り，我々は期待値計算を Monte Carlo 法で近似できることを知っている．確率密度関数 $p(\boldsymbol{\omega})$ にしたがって確率変数 $\boldsymbol{\omega}$ をサンプリングした値を $\{\boldsymbol{\omega}_\ell\}_{\ell=1}^L$ とすると，

$$\begin{aligned} k(\boldsymbol{x}-\boldsymbol{y}) &= \mathrm{E}_{\boldsymbol{\omega}\sim p}\left[e^{i\boldsymbol{\omega}^\mathsf{T}(\boldsymbol{x}-\boldsymbol{y})}\right] \\ &\approx \frac{1}{L}\sum_{\ell=1}^{L} e^{i\boldsymbol{\omega}_\ell^\mathsf{T}(\boldsymbol{x}-\boldsymbol{y})}, \end{aligned} \tag{12}$$

が成り立つのである．これにより，カーネル関数を，Fourier 変換と乱数を用いて近似できたことになる．これが RFF の本質である．

さて，興奮に任せて大切なポイントをひとつ取り落してしまったことは読者もお気付きであろう．式 (11) が成立するためには，関数 $p(\boldsymbol{\omega})$ が確率密度関数と同格でなければならない．すなわち，可積分かつ非負の値を取る関数でなければならない．しかし驚くべきことに，カーネル関数が正定値関数である限り，その Fourier 変換は可積分かつ非負であることが証明できる．これは非常に大事な事実であるから，定理としてまとめておく．ただし，議論の流れを妨げないよう証明は本文書の最後の節で行う．

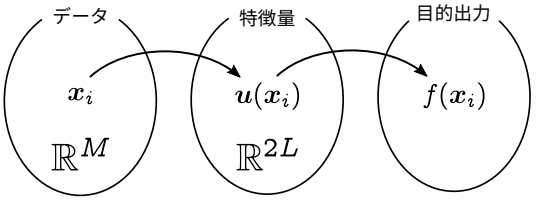

図 2　RFF と特徴抽出

定理 3.1 (平行移動不変な正定値カーネルの *Fourier* 変換)

正定値カーネル関数 $k(\boldsymbol{x},\boldsymbol{y})$ が平行移動不変，すなわち $k(\boldsymbol{x},\boldsymbol{y}) = k_\Delta(\boldsymbol{x}-\boldsymbol{y})$ をみたす関数 k_Δ が存在したとする．このとき，カーネル関数 $k_\Delta(\boldsymbol{x}-\boldsymbol{y})$ の *Fourier* 変換は L_1 ノルムの意味で可積分であり，かつ非負関数である．

さて，最後に近似したカーネル関数をどのように使うかについて説明しよう．一般に，カーネル関数は正定値関数であるから，式 (12) の最右辺の虚部は不要である．したがって

$$k(\boldsymbol{x}-\boldsymbol{y}) \approx \frac{1}{L}\sum_{\ell=1}^{L} e^{i\boldsymbol{\omega}_\ell^\mathsf{T}(\boldsymbol{x}-\boldsymbol{y})} \tag{13}$$

$$= \frac{1}{L}\sum_{\ell=1}^{L} \cos(\boldsymbol{\omega}_\ell^\mathsf{T}(\boldsymbol{x}-\boldsymbol{y})) \tag{14}$$

$$= \frac{1}{L}\sum_{\ell=1}^{L} \big[\cos(\boldsymbol{\omega}_\ell^\mathsf{T}\boldsymbol{x})\cos(\boldsymbol{\omega}_\ell^\mathsf{T}\boldsymbol{y}) + \sin(\boldsymbol{\omega}_\ell^\mathsf{T}\boldsymbol{x})\sin(\boldsymbol{\omega}_\ell^\mathsf{T}\boldsymbol{y})\big], \tag{15}$$

を得る．ただし式 (13) から式 (14) へは Euler の公式を，式 (14) から式 (15) へはかの懐しき三角関数の加法定理を用いた．さてここでベクトル

$$\boldsymbol{u}(\boldsymbol{x}) \triangleq \frac{1}{\sqrt{L}}\begin{pmatrix}\cos(\boldsymbol{\omega}_1^\mathsf{T}\boldsymbol{x})\\ \vdots \\ \cos(\boldsymbol{\omega}_L^\mathsf{T}\boldsymbol{x})\\ \sin(\boldsymbol{\omega}_1^\mathsf{T}\boldsymbol{x})\\ \vdots \\ \sin(\boldsymbol{\omega}_L^\mathsf{T}\boldsymbol{x})\end{pmatrix} = \begin{pmatrix}\cos(\boldsymbol{W}\boldsymbol{x})\\ \sin(\boldsymbol{W}\boldsymbol{x})\end{pmatrix} \tag{16}$$

を導入しよう．ただし行列 $\boldsymbol{W} = (\boldsymbol{\omega}_1,\ldots,\boldsymbol{\omega}_L)^\mathsf{T}$ であり，上式の最右辺ではベクトルの余弦関数を $\cos\boldsymbol{x} = (\cos x_1,\ldots,\cos x_L)^\mathsf{T}$ と定義して使用している．sin についても同様である．すると式 (15) の右辺は

$$\boldsymbol{u}(\boldsymbol{x})^\mathsf{T}\boldsymbol{u}(\boldsymbol{y}), \tag{17}$$

と非常にシンプルに書き改めることができる．すなわち

$$k(\boldsymbol{x}-\boldsymbol{y}) \approx \boldsymbol{u}(\boldsymbol{x})^\mathsf{T}\boldsymbol{u}(\boldsymbol{y}), \tag{18}$$

と近似することができたのである．

そもそもカーネル関数は，特徴量の内積であったことを思い出してほしい．式 (18) は，カーネル関数を近似することと，特徴量として $\phi_{\boldsymbol{x}}$ ではなく $\boldsymbol{u}(\boldsymbol{x})$ を使うことが等価だと言っているのである．つまり図 2 のような構図が成り立つ．

今，特徴量はもはや関数ではなく，ただの $2L$ 次元の実ベクトルである．したがって，回帰や分類などのタスクを実行するのにミョーチクリンなトリックはもはや必要ない．有限次元の線形回帰なり線形サポートベクターマシンなりを適用すれば良いのである．線形サポートベクターマシンの学習および推論は，カーネルサポートベクターマシンと比較すると桁違いに速い．しかも特徴量は，高々有限次元の近似とはいえ，カーネル関数の柔軟性を残している．このようにしてカーネル法はその柔軟性を残しつつ大規模データに適用しうるだけの高速性を手にすることができるのである．

3.3　RBF カーネルの場合

さて，実際に RFF を運用するためには，カーネル関数を何らかの関数に定め，そのカーネル関数に対して確率密度関数 $p(\boldsymbol{\omega})$ を算出しなければならない．RFF はカーネル関数が $k(\boldsymbol{x}-\boldsymbol{y})$ と表現できるものに限られるから，ここでは RBF カーネルの場合について考察しよう．

とはいえ，確率密度関数 $p(\boldsymbol{\omega})$ の導出は式 (9) にしたがって Fourier 変換を計算するだけである．直接計算だけであるので証明は省略するが，

$$p(\boldsymbol{\omega}) = \frac{1}{\sqrt{2\pi\sigma^2}^M}\exp\left(-\frac{\|\boldsymbol{\omega}\|^2}{2\sigma^2}\right), \tag{19}$$

となる．つまり確率変数 $\boldsymbol{\omega}$ は正規分布するのである．したがって確率変数 $\boldsymbol{\omega}$ は正規分布からサンプリングすれば良いということになる．

以上をまとめると，RBF ベースの RFF は次の手順で実施すれば良い．

1. RFF の自由度 $L \in \mathbb{Z}^+$ を適当な数に設定する．これはカーネル関数を近似する際のサンプリング数である．
2. 行列の各要素が正規分布にしたがうランダム行列 $\boldsymbol{W} \in \mathbb{R}^{L\times M}$ を生成する．
3. 各学習データ $\boldsymbol{x}_i$ に式 (16) の変換を施し，それを改めて学習データセットとみなす．
4. 変換後のデータセットに対し，線形回帰や線形サポートベクターマシンなどのアルゴリズム f を適用する．
5. 推論時には，推論データ $\boldsymbol{x}$ に対してもデータ変換式 (16) を適用して $\boldsymbol{u}(\boldsymbol{x})$ を算出し，その後に推論 $f(\boldsymbol{u}(\boldsymbol{x}))$ を行う．

3.4　深層学習との関係

最後に，RFF を用いたサポートベクターマシンと深層学習との関係性について述べておく．RFF は入力データ $\boldsymbol{x}$ の特徴量を式 (16) で定める手法であった．これを線形サポートベクターマシンと組み合わせてみよう．

問題設定としては，教師付き学習データを用いて C クラスのクラス分類器を作成することを目標としよう．学習データセットは $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ とする．ただし $\boldsymbol{x}_n \in \mathbb{R}^M$，$y_n \in \{1,2,\ldots,C\}$ であり，$\boldsymbol{x}_n$ が学習データに，y_n が教師データに対応する．

さて，線形サポートベクターマシンの判別式は線形関数であ

るから，入力データ $\boldsymbol{x}$ に対する各クラスの対数尤度は

$$f_c(\boldsymbol{x}) = \boldsymbol{a}_c^\mathsf{T}\boldsymbol{x} + b_c, \tag{20}$$

で表現される．ベクトル $\boldsymbol{a}_c$ およびスカラー b_c は学習パラメータであり，各クラスの対数尤度を上手く表現できるように学習される．

さて, 少しだけ表現を変更しよう．まず, すべてのクラスをまとめて記述するために，$\boldsymbol{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_C)^\mathsf{T}$, $\boldsymbol{b} = (b_1, \ldots, b_C)$ とおく．さらに，RFF の適用を前提とすれば，入力データは $\boldsymbol{x}$ ではなく $\boldsymbol{u}(\boldsymbol{x})$ に置き換えなければならない．すると式 (20) は

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\begin{pmatrix}\cos(\boldsymbol{W}\boldsymbol{x})\\ \sin(\boldsymbol{W}\boldsymbol{x})\end{pmatrix} + \boldsymbol{b}, \tag{21}$$

となる．

これを深層学習の立場から見ると，ちょっと特殊な活性化関数を用いた，2 層の全結合層を持つニューラルネットワークとみなすことができる．図 3 を参照されたい．

もちろん，学習の仕方はサポートベクターマシンと深層学習で全く異なることに注意されたい．サポートベクターマシンは区分的線形な目的関数を制約付き 2 次計画問題に帰着させているのに対し，深層学習ではクロスエントロピーを確率的勾配法で最小化させている．

4　数値実験

GitHub に RFF の実装およびそれを用いたいくつかの計算例を掲載してある．以下のレポジトリを参照されたい．

```
https://github.com/tiskw/
  Random-Fourier-Features/tree/master/
    examples/rff_svc_for_mnistsubsection
```

5　定理の証明

本節では，本文書で証明せずに残してあった定理を証明する．前節までは，原論文に合わせるために角周波数ベースの Fourier 変換を用いて議論をしていたが，以下では数学的シンプルさを優先して周波数ベースの Fourier 変換を用いることとする．

Fourier 変換と Parseval の等式

まずは Fourier 変換について簡単におさらいしておこう．本節では Fourier 変換の定義として

$$\mathcal{F}[f](\xi) \triangleq \int_{-\infty}^{\infty} f(x)e^{-2\pi i x\xi}\,\mathrm{d}x, \tag{22}$$

を採用する．この時，逆 Fourie 変換は

$$\mathcal{F}^{-1}[\widehat{f}](x) \triangleq \int_{-\infty}^{\infty} \widehat{f}(\xi)e^{2\pi i x\xi}\,\mathrm{d}\xi, \tag{23}$$

となる．原論文で用いられている角周波数ベースの Fourier 変換とはスカラー倍の違いがあるだけで，本質的には同一である．多変数関数 $f(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^M$ の Fourier 変換および逆 Fourie 変換は

$$\mathcal{F}[f](\boldsymbol{\xi}) \triangleq \int_{-\infty}^{\infty} f(x)e^{-2\pi i \boldsymbol{x}^\mathsf{T}\boldsymbol{\xi}}\,\mathrm{d}\boldsymbol{x}, \tag{24}$$

$$\mathcal{F}^{-1}[\widehat{f}](\boldsymbol{x}) \triangleq \int_{-\infty}^{\infty} \widehat{f}(\boldsymbol{\xi})e^{2\pi i \boldsymbol{x}^\mathsf{T}\boldsymbol{\xi}}\,\mathrm{d}\boldsymbol{\xi}, \tag{25}$$

となる．ただし記号 $\mathrm{d}\boldsymbol{x}$ は $\mathrm{d}x_1\mathrm{d}x_2\cdots\mathrm{d}x_M$ の略記であることに注意せよ．

> NOTE: Lebesgue 測度論の慣習に従えば，この $\mathrm{d}\boldsymbol{x}$ は本来 $\mu(\mathrm{d}\boldsymbol{x})$ あるいは $\mathrm{d}\mu(\boldsymbol{x})$ と表記されるべきものである．ただし μ は Lebesgue 測度である．だが表記の冗長さと，測度論に立ち入り結論に辿り着くまでの時間が長くなることで読者の興味が薄れることを恐れ，ここでは上記の略記を用いた．ゴマカシだという誇りは受けて然るべきである．より良いアプローチがあればぜひご連絡頂きたい．

オイオイ，式 (22) が定義できる関数 f なんてかなり少なくないかと思ったそこの読者，非常に良い眼をお持ちである．式 (22) の収束性については本文書末尾の付録で簡単に言及する．

次に Parseval の等式 (*Parseval's identity*) について見ていこう．一般に Parseval の等式と言うと $\|f\|^2 = \|\widehat{f}\|^2$ を指すことも多いが，本文書ではより一般的な形の定理を紹介する．

定理 5.1 (*Parseval* の等式)

任意の *Fourier* 変換可能な関数 f, g およびその *Fourier* 変換 $\widehat{f}, \widehat{g}$ に対して $\langle f, g\rangle = \langle \widehat{f}, \widehat{g}\rangle$ が成り立つ．

ここでは Parseval の等式の証明は省略する．証明が気になる読者は（そうあるべきである！）本文書末尾の付録を参照されたい．

定理の証明

定理を再掲し証明しよう．

定理 5.2 (平行移動不変な正定値カーネルの *Fourier* 変換)

正定値カーネル関数 $k(\boldsymbol{x}, \boldsymbol{y})$ が平行移動不変，すなわち $k(\boldsymbol{x}, \boldsymbol{y}) = k_\Delta(\boldsymbol{x} - \boldsymbol{y})$ をみたす関数 k_Δ が存在したとする．このとき，カーネル関数 $k_\Delta(\boldsymbol{x} - \boldsymbol{y})$ の *Fourier* 変換は L_1 ノルムの意味で可積分であり，かつ非負関数である．

証明： 以下，$\boldsymbol{x}, \boldsymbol{y}$ の次元を M とし，$\mathcal{X} = \mathbb{R}^M$ とおく．すなわち $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$ である．カーネル関数 k の対称性より

$$k_\Delta(\boldsymbol{x}) = k(\boldsymbol{x}/2, -\boldsymbol{x}/2) = k(-\boldsymbol{x}/2, \boldsymbol{x}/2) = k_\Delta(-\boldsymbol{x}) \tag{26}$$

が成り立つ．これを用いれば k_Δ の Fourier 変換は実数値関数であることがわかる．なぜならば k_Δ の Fourier 変換の複素共役を計算してみると

$$\mathcal{F}[k_\Delta]^*(\boldsymbol{\xi}) = \int_{\mathcal{X}} k_\Delta(\boldsymbol{x})e^{2\pi i \boldsymbol{x}^\mathsf{T}\boldsymbol{\xi}}\,\mathrm{d}\boldsymbol{x} \tag{27}$$

ここで $\boldsymbol{x} = -\boldsymbol{z}$ と置換すれば

$$= \int_{\mathcal{X}} k_\Delta(-\boldsymbol{z})e^{-2\pi i \boldsymbol{z}^\mathsf{T}\boldsymbol{\xi}}\,\mathrm{d}\boldsymbol{z} \tag{28}$$

$$= \int_{\mathcal{X}} k_\Delta(\boldsymbol{z})e^{-2\pi i \boldsymbol{z}^\mathsf{T}\boldsymbol{\xi}}\,\mathrm{d}\boldsymbol{z} = \mathcal{F}[k_\Delta](\boldsymbol{\xi}), \tag{29}$$

となり，複素共役をとる前の Fourier 変換と一致する．したがって $\mathcal{F}[k_\Delta](\boldsymbol{\xi})$ は少なくとも実数値関数である．次に，カーネル関数 k が正定値関数であることから，任意の実数値関数 y について

$$\iint_{\mathcal{X}\times\mathcal{X}} y(\boldsymbol{x}_1)k(\boldsymbol{x}_1, \boldsymbol{x}_2)y(\boldsymbol{x}_2)\,\mathrm{d}\boldsymbol{x}_1\mathrm{d}\boldsymbol{x}_2 \geq 0, \tag{30}$$

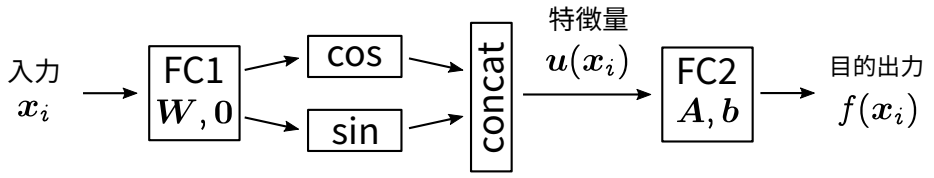

図 3　RFF とニューラルネットワーク

が成り立つ．上式の左辺を変形すると

$$\iint_{\mathcal{X}\times\mathcal{X}} y(\boldsymbol{x}_1)k(\boldsymbol{x}_1,\boldsymbol{x}_2)y(\boldsymbol{x}_2)\,\mathrm{d}\boldsymbol{x}_1\mathrm{d}\boldsymbol{x}_2 \tag{31}$$

$$= \int_{\mathcal{X}} y(\boldsymbol{x}_1)\langle k(\boldsymbol{x}_1,\cdot), y\rangle\,\mathrm{d}\boldsymbol{x}_1 \tag{32}$$

ここで内積 $\langle k(\boldsymbol{x}_1,\cdot), y\rangle$ に対してパーセバルの不等式を用いる．新たに $\widehat{k}_{\boldsymbol{x}_1}(\boldsymbol{\xi}_2) = \mathcal{F}[k(\boldsymbol{x}_1,\cdot)](\boldsymbol{\xi}_2)$ とおくと

$$= \int_{\mathcal{X}} y(\boldsymbol{x}_1)\langle \widehat{k}_{\boldsymbol{x}_1}, \widehat{y}\rangle\,\mathrm{d}\boldsymbol{x}_1 \tag{33}$$

$$= \iint_{\mathcal{X}\times\mathcal{X}} y(\boldsymbol{x}_1)\widehat{k}_{\boldsymbol{x}_1}(\boldsymbol{\xi}_2)\widehat{y}^*(\boldsymbol{\xi}_2)\,\mathrm{d}\boldsymbol{x}_1\mathrm{d}\boldsymbol{\xi}_2 \tag{34}$$

ここで見易さのために関数 $\widehat{k}_{\boldsymbol{x}_1}(\boldsymbol{\xi}_2) = l_{\boldsymbol{\xi}_2}(\boldsymbol{x}_1)$ とおくと

$$= \iint_{\mathcal{X}\times\mathcal{X}} y(\boldsymbol{x}_1)l_{\boldsymbol{\xi}_2}(\boldsymbol{x}_1)\widehat{y}^*(\boldsymbol{\xi}_2)\,\mathrm{d}\boldsymbol{x}_1\mathrm{d}\boldsymbol{\xi}_2 \tag{35}$$

$$= \int_{\mathcal{X}} \langle y, l_{\boldsymbol{\xi}_2}\rangle\widehat{y}^*(\boldsymbol{\xi}_2)\,\mathrm{d}\boldsymbol{\xi}_2 \tag{36}$$

ここで再び $\langle y, l_{\boldsymbol{\xi}_2}\rangle$ に対してパーセバルの不等式を用いる．新たに $\widehat{l}_{\boldsymbol{\xi}_2}(\boldsymbol{\xi}_1) = \mathcal{F}[l_{\boldsymbol{\xi}_2}](\boldsymbol{\xi}_1)$ とおくと

$$= \int_{\mathcal{X}} \langle \widehat{y}, \widehat{l}_{\boldsymbol{\xi}_2}\rangle\widehat{y}^*(\boldsymbol{\xi}_2)\,\mathrm{d}\boldsymbol{\xi}_2 \tag{37}$$

$$= \iint_{\mathcal{X}\times\mathcal{X}} \widehat{y}(\boldsymbol{\xi}_1)\widehat{l}_{\boldsymbol{\xi}_2}(\boldsymbol{\xi}_1)\widehat{y}^*(\boldsymbol{\xi}_2)\,\mathrm{d}\boldsymbol{\xi}_1\mathrm{d}\boldsymbol{\xi}_2, \tag{38}$$

となる．ここで $\widehat{l}_{\boldsymbol{\xi}_2}(\boldsymbol{\xi}_1)$ を真面目に計算してみると

$$\widehat{l}_{\boldsymbol{\xi}_2}(\boldsymbol{\xi}_1) = \iint_{\mathcal{X}\times\mathcal{X}} k(\boldsymbol{x}_1,\boldsymbol{x}_2)e^{-2\pi i(\boldsymbol{x}_1^\mathsf{T}\boldsymbol{\xi}_1 - \boldsymbol{x}_2^\mathsf{T}\boldsymbol{\xi}_2)}\,\mathrm{d}\boldsymbol{x}_1\mathrm{d}\boldsymbol{x}_2 \tag{39}$$

$$= \iint_{\mathcal{X}\times\mathcal{X}} k_\Delta(\boldsymbol{x}_1-\boldsymbol{x}_2)e^{-2\pi i(\boldsymbol{x}_1^\mathsf{T}\boldsymbol{\xi}_1 - \boldsymbol{x}_2^\mathsf{T}\boldsymbol{\xi}_2)}\,\mathrm{d}\boldsymbol{x}_1\mathrm{d}\boldsymbol{x}_2$$

$$= \delta(\boldsymbol{\xi}_1-\boldsymbol{\xi}_2)\widehat{k}(\boldsymbol{\xi}_1), \tag{40}$$

となる．これを式 (38) に戻すと

$$= \iint_{\mathcal{X}\times\mathcal{X}} \widehat{y}(\boldsymbol{\xi}_1)\delta(\boldsymbol{\xi}_1-\boldsymbol{\xi}_2)\widehat{k}(\boldsymbol{\xi}_1)\widehat{y}^*(\boldsymbol{\xi}_2)\,\mathrm{d}\boldsymbol{\xi}_1\mathrm{d}\boldsymbol{\xi}_2 \tag{41}$$

$$= \int_{\mathcal{X}} \widehat{y}(\boldsymbol{\xi})\widehat{k}(\boldsymbol{\xi})\widehat{y}^*(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi} \tag{42}$$

$$= \int_{\mathcal{X}} \|\widehat{y}(\boldsymbol{\xi})\|^2\,\widehat{k}(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi} \geq 0, \tag{43}$$

となる．これが任意の関数 y について成立するから，$\widehat{y}(\boldsymbol{\xi})$ が定数関数の場合も成立しなければならない．したがって

$$\int_{\mathcal{X}} \widehat{k}(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi} \geq 0, \tag{44}$$

が成り立つ必要がある．したがって $\widehat{k}$ は L_1 ノルムの意味で可積分である．同様に関数 $\widehat{y}(\boldsymbol{\xi})$ が限りなくデルタ関数に近い形状の関数に対しても式 (38) が成立しなければならないから，任意の $\boldsymbol{\xi}$ に対して $\widehat{k}(\boldsymbol{\xi}) \geq 0$ が成り立つ必要がある．したがって $\widehat{k}(\boldsymbol{\xi}) \geq 0$ は非負関数である．■

付録：Fourier 変換

本節では，定理証明の際に省略した Fourier 変換および Parseval の等式の解説を行う．本節でも数学的シンプルさを優先して周波数ベースの Fourier 変換を用いることとする．

Fourier 変換に関する補足

Fourier 変換の定義としては本文の式 (22) で十分だが，より良い表記，より深い理解のために，Fourier 変換を関数空間の内積を用いて表現してみよう．まず関数 $\eta_\xi(x)$ を

$$\eta_\xi(x) \triangleq e^{2\pi i x\xi}, \tag{45}$$

と定義する．次に関数の内積 $\langle\cdot,\cdot\rangle$ を最も標準的な内積演算

$$\langle f, g\rangle \triangleq \int_{-\infty}^{\infty} f(x)g^*(x)\,\mathrm{d}x, \tag{46}$$

で定めるとしよう．ただし $g^*(x)$ は関数 $g(x)$ の複素共役である．このとき式 (22) で表現でされる Fourier 変換は

$$\mathcal{F}[f](\xi) = \langle f, \eta_\xi\rangle, \tag{47}$$

と表すことができる．このとき関数群 $\{\eta_\xi(x)\,|\,\xi\in\mathbb{R}\}$ は正規直交基底と非常に似た性質を持つ．すなわち

$$\langle \eta_{\xi_1}, \eta_{\xi_2}\rangle = \delta(\xi_1-\xi_2), \tag{48}$$

が成り立つ．ただし δ は Dirac のデルタ関数 (*Dirac delta function*) である．したがって Fourier 変換は正規直交基底との内積，つまるところただの基底交換のようなもの，と見ることもできるのである．

> NOTE: Fourier 変換の定義を初めて見た若かりし頃，諸君らも「何で Fourier 変換が逆回転 $e^{-2\pi ix\xi}$ で逆 Fourier 変換が順回転 $e^{2\pi ix\xi}$ なんだよ，フツーは逆だろ！」とキレかかった経験があるのではないだろうか．上述の内積による Fourier 変換の理解はこの疑問を解消してくれる．そう，基底 $\eta_\xi(x)$ はちゃんと順回転 $e^{2\pi ix\xi}$ で定義されていたのだ．しかし内積を計算する際に共役をとる必要があり，そこで回転方向が反転したのである．少しはスッキリしたであろうか．

Fourier 変換の収束性について

Parseval の等式の証明に移る前に，読者が気になっているであろう Fourier 変換の定義式 (22) の収束性について簡単に述べ

ておこう．つまり，式 (22) が定義できるのはどんな関数か？という問題である．結論から言えば，式 (22) が収束する条件は

$$\int_{-\infty}^{\infty} |f(x)|\,\mathrm{d}x < \infty, \tag{49}$$

である．ハッキリ言って，上式を満たす関数は応用上の観点から見ても非常に少ない．例えば三角関数や指数関数ですら上式の制約は満たさない．とはいえ少なくとも応用上必要な関数に対しては Fourier 変換ができないと困る．この問題に対するアプローチは様々あるが，著者の知る限り，多くの人の理解は以下の (1) の方法で理解していると思われる．初学者は (1) の理解で良いであろう．ちなみに著者は理論的にも便利な (2) で理解している．興味があれば調べてみると良い．

(1) 扱う関数はすべて有限の台を持つと仮定する方法．関数の台が有限であれば，応用上必要な関数はほぼすべて積分可能となる．
(2) 両側 Laplace 変換の特殊な場合 ($s = i\omega$) として Fourier 変換を定義する方法である．実は両側 Laplace 変換は，解析接続を用いた巧妙な仕組みにより，非常に幅広い関数に対して定義可能である．

Parseval の等式

次に Parseval の等式 (*Parseval's identity*) について見ていこう．一般に Parseval の等式と言うと $\|f\|^2 = \|\widehat{f}\|^2$ を指すことも多いが，本文書ではより一般的な形の定理を証明する．Fourier 変換が正規直交基底を用いた基底の交換であることが分かれば，ほぼ自明な定理である．

定理 5.3 (*Parseval* の等式)

任意の *Fourier* 変換可能な関数 f, g およびその *Fourier* 変換 $\widehat{f}, \widehat{g}$ に対して $\langle f, g\rangle = \langle \widehat{f}, \widehat{g}\rangle$ が成り立つ．

証明： 証明は直接計算による．

$$\begin{aligned}\langle \widehat{f}, \widehat{g}\rangle &= \int_{-\infty}^{-\infty} \langle f, \eta_\xi\rangle\langle g, \eta_\xi\rangle\,\mathrm{d}\xi \\ &= \iiint_{-\infty}^{-\infty} f(x_1)g(x_2)\eta_\xi(x_1)\eta_\xi(x_2)\,\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}\xi\end{aligned}$$

ここで $\eta_\xi(x_1)\eta_\xi(x_2) = \eta_{x_1}(\xi)\eta_{x_2}(\xi)$ に注意すれば

$$= \iint_{-\infty}^{-\infty} f(x_1)g(x_2)\langle \eta_{x_1}, \eta_{x_2}\rangle\,\mathrm{d}x_1\mathrm{d}x_2$$

関数 $\eta_x(\xi)$ の正規直交性に注意すれば $\langle \eta_{x_1}, \eta_{x_2}\rangle$ は $\delta(x_1 - x_2)$ であるから

$$= \int_{-\infty}^{-\infty} f(x)g(x)\,\mathrm{d}x = \langle f, g\rangle,$$

となり，題意が成り立つ． ■

ここまで読み進めた読者には指摘するまでもないであろうが，上記定理の特殊な場合として $f = g$ とおけば，良く知られた $\|f\|^2 = \|\widehat{f}\|^2$ を得る．

おわりに

本文書では，非常に簡単にではありますが，Random Fourier Features の解説を試みてみました．個人的には，簡単のためにデータを 1 次元にして説明しようかとも考えたのですが，さすがに実応用の段階でつまづくであろうと思い，最初から多次元のデータを想定して書いてみました．この点についてはもしかしたら将来的に修正するかもしれません．

Random Fourier Features の拡張としては，2017 年に Google より ORF (*Orthogonal Random Features*) および SORF (*Structured ORF*) が提案されています．これは RFF のランダム行列 $\boldsymbol{W}$ が直行行列でないことに注目し，直交性を強制的に持たせることでより高い近似効率を狙ったものです．私も手元で実装し，いくつかの有名なデータセットに対して ORF および SORF を試しましたが，確かにカーネルの近似精度は高いのですが，各データセットの推論精度という意味では RFF と大きな差はありませんでした．どうやらカーネルの近似精度と推論精度は単純には相関しないようです．まだ仮説の域を出ませんが，おそらく有限次元近似が良い正則化として機能しているのでしょう．本文書で ORF や SORF に触れていないのは，それが主な理由です．

私の数学の師は常に「人生で一度も証明を与えたことのない定理を自ら使用するのは数学的犯罪である」と私に教えて下さいました．これは教育的な配慮だけでは決してないと思います．内部を理解せずに扱うことによる誤用，それの招く惨めな結末．数学だけに限った話ではないでしょう．近年の機械学習の研究者やエンジニアはこれをよくよく心に留めおくべきです．もちろん私自身を含めて，です．

最後に，Toyota Research Institute Advanced Development, Inc. の石原昌弘さんには本文書の誤植を指摘して頂きました．この場を借りて厚く御礼申し上げます．また，私の数学的活動は，2017 年に逝去された恩師，山下弘一郎先生や，大学および大学院で私の指導教官を担当して下さった早川朋久准教授をはじめ，数学で私と関わりを持ったすべての方々のおかげで成り立っています．そして，数学的活動の以前に，そもそも私の生は両親によって与えられ，妻によって支えられています．

参考文献

[1] A. Rahimi and B. Recht, “Random Features for Large-Scale Kernel Machines”, Neural Information Processing Systems, 2007.
[2] F. X. Yu, A. T. Suresh, K. Choromanski, D. H. Rice and S. Kumar, “Orthogonal Random Features”, Neural Information Processing Systems, 2016.
[3] 赤穂昭太郎，「カーネル多変量解析」，岩波書店，2008.
[4] C. Bishop, “Pattern Recognition and Machine Learning”, Springer, 2010.