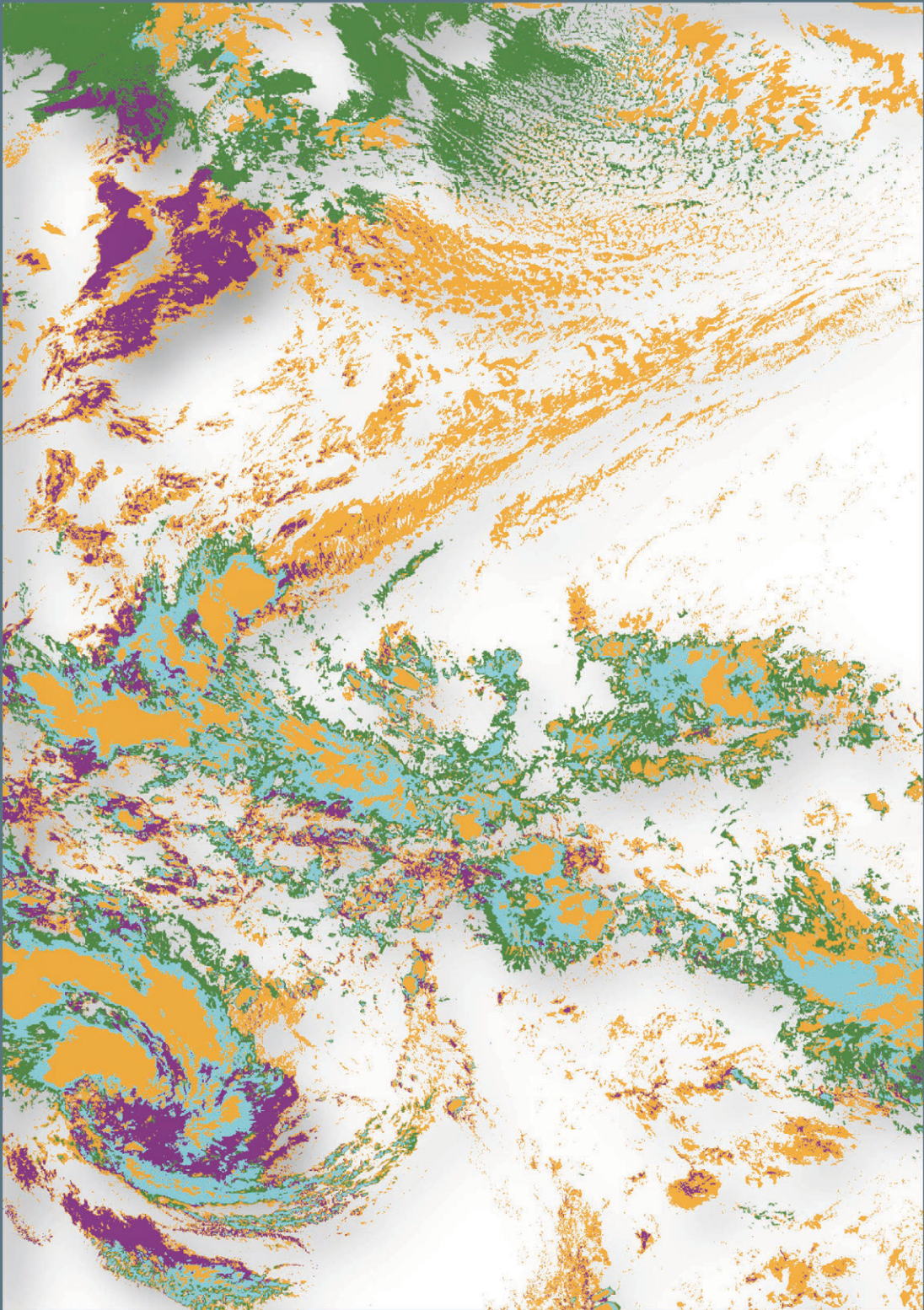


宇宙から色

んな雲を見つけよう！



衛星画像 × 機械学習入門

金子凌・小槻峻司

環境リモートセンシング研究センター

VL講習会@千葉大学

Day1: 2025-03-03

Day2: 2025-03-04

はじめに

この度は、本実習を受講いただきありがとうございます。

ひまわり 8号で観測が始まってから、早 10年が経ちました。今はひまわり 9号にその観測が受け継がれ、毎日、ほぼ絶え間なく地球の様子を送り続けてくれています。観測の始まった 2015年 は、私が学部生として研究を始めた年でした。2016年、弊学で開催された VL 講習会[1]にて、ひまわりデータの取り扱い方を学んだのが、ひまわりとの出会いでした。あれから様々なデータの取り扱い方法を学び、機械学習モデルにどのように学習させれば良いかなどを考えて今に至ります。

ひまわりのデータの特徴の 1つは、高頻度且つ高解像度な観測にあります。しかし、観測開始から 10年が経った今でも、これらのデータを「どのように」「どのようなモデルに」学習させるかという事に関しては、まとまった知見がなかなかないのが現状です。これは、観測データが余りにも巨大であるため取り扱いが難しく、「衛星×機械学習」分野に新規参入する方の障壁となっていることが、1つの原因だと考えています。すでに、ひまわり 10号の計画も進みつつありますが[2]、将来的に更に巨大化するデータセットを鑑みても、この障壁をなんとか取り払う必要があるように思います。

そこで、本実習では、私が学んだノウハウをお伝えすることで、気象衛星ひまわり 8号・9号の衛星画像から、雲の分類を行う機械学習モデルを作成することを目標とします。対象は機械学習を全くやったことがない、もしくは初学の方です。この実習を通して、深層学習を始めとした機械学習に最適な衛星画像のデータの扱い方、簡単な可視化の方法、そして機械学習の基礎を学んでいただければ幸いです。

2025年 3月 3日 金子 凌

本実習の概要とその背景・目的

本実習では、そのタイトルにあるとおり、衛星画像から雲型の分類を行います。地上から見た場合は、巻雲、層雲、積雲など、様々な雲の形が立体的に観察されますが、宇宙からはどう見えるのでしょうか。

気象衛星センターでは以前より衛星画像を用いた雲形の分類を行っています[3]。ここでは、衛星画像は、解像度の関係で、地上から観測する十種雲型とは異なると言及されており、様々な側面から衛星画像を用いた雲型の分類を紹介しています。¹この時代より、遥かに観測項目が増えたひまわり 8号・9号では、どのように衛星からの雲を分類すれば良いのでしょうか。また、観測項目が増えたと同時に、計算機の性能も大幅に向上し、できることも格段に増えました。例えば、機械学習や深層学習など、昨今の世の中を賑わせている技術も選択肢となりうる時代となりました。

¹面白いので読んでみることをオススメします[3]。

そこで、本実習では、機械学習を用いて、ひまわり 8 号・9 号の高精細な観測を用い、雲型の分類を行ったらどのようなことがわかるのか、本実習では、そんな疑問を元に、衛星画像を用いた雲型の分類を行うことを目的としています。観測項目が 16 に及ぶ高次元なデータを、機械学習により分類する事で、今まで私たちが見つけられなかった特徴が見つかったり、自動化することによって気象の業務に役立てられたり、そんなことも将来的には期待できるかもしれません。また、衛星画像を使うことで、人が観測することができない広域な観測も可能となるのは言うまでもないメリットとなるでしょう。



Figure 1. 地上から見た雲形ほど衛星データからは細かく見られません

本実習で取り扱うこと、取り扱わないこと

本実習では、基本的には以下のように、取り扱うことと取り扱わないことを分けています。ただし、興味があれば直接聞いていただければ、可能な限り対応できると思います。また、昨今は、本や検索エンジンで調べれば出てくること、AIに聞けば分かることも沢山ありますから、適宜調べたほうが、実は効率的かもしれません。

2時間30分の実習であり、Google Colab を用いるので、計算時間やメモリの制限があります。そのため、大規模なデータセットを用いた学習は行わないこととします。しかし、殆どコードを変えずに大規模なデータセットなどに対応できるため、ご所有の計算機などでも、学習した内容を応用することはできるはずです。こちらも興味があれば、挑戦してみてください。

項目	取り扱うこと	取り扱わないこと
データ	圧縮された数時刻の衛星データ	非圧縮の全期間の衛星データ
Python	機械学習の解説	基本的なコードの解説
可視化	Xarray を用いた可視化手法	可視化ライブラリーの詳細な解説
計算環境	Google Colab を利用	ローカル環境での実行
機械学習モデル	K-means を用いたシンプルな手法	深層学習・非線形の機械学習モデル
学習内容	Google Colab で行うため小さめ	大規模なデータセット

第2版発行にあたって

お陰様で無事にVL講習会も終わりました。改めまして、受講者として、スタッフとして関わってくださった皆様に御礼申し上げます。第2版では、講習会で実際に躓いた場所の修正、そしてコードの修正を行っています。特に、クラスタリングの結果の際、雲以外のクラスタを除去するプロセスがありますが、ここに大きな修正があります。初版のプログラムでも計算は間違いないのですが、カラーバーの更新が行われていませんでした（カラーバーに除外したクラスタが表示されっぱなしになっていた）。この場所を修正し、更に見やすい図となるよう修正しています。他にもZarrファイルの保存方法など、ちょっとした内容の追加や修正を行っています。

もし、この本が少しでもお役に立てたなら、ご感想などいただけると嬉しいです、是非周りの方々にご紹介いただければ幸いです。

2025年3月13日 金子 凌

参考文献

- [1] VL講習会2016のページ, <https://www.cr.chiba-u.jp/databases/VL/VL-Lecture2016/>, 2025/02/27 閲覧.
- [2] ひまわり10号について, https://www.data.jma.go.jp/sat_info/himawari/kondan/kai8/shiryoku8_1.pdf, 2025/02/27 閲覧.
- [3] 気象衛星センター, 気象衛星画像の解析と利用, 2000.

目次

はじめに	
本実習の概要とその背景・目的	2
本実習で取り扱うこと、取り扱わないこと	3
第2版発行にあたって	5
1 ひまわり8号・9号	
1.1 歴史と CEReS の役割	8
1.2 性能	9
2 機械学習概要	
2.1 本実習の目標	10
2.2 K-means	10
2.3 ひまわり8号・9号のデータにどうやって適用するのか	13
3 Xarray を用いた 巨大な衛星データのハンドリング	
3.1 この実習に関して	15
3.2 特筆すべきライブラリーの紹介とそのメリット	15
3.3 教材のコピーと Google Colab の起動	16
3.4 Google Colaboratory の起動とデータの解凍	18
3.5 Google Colaboratory に必要なライブラリーの準備をしよう	19
3.6 Xarray を使ってデータを読み込む	19
3.7 変数を抽出してみよう	21
3.8 可視化してみよう	21
3.9 領域を指定してみよう	24
3.10 必要な場所を自在に抽出しよう	25
3.11 統計量を求めたりデータの性質を見てみよう	26
3.12 複数のファイルを1つのファイルとして読み込もう	27
3.13 RGB 画像の作成	29
3.14 zarr 形式への書き出し方（発展）	32
3.15 Himawari Bench の紹介	34
4 機械学習モデルを トレーニングしよう	
4.1 データと計算環境の準備	35

4.2 K-means の学習	37
4.3 結果の可視化	38
4.4 学習済みモデルを他のデータの推論に利用しよう	49
4.5 モデルの保存と読み込み	52
5 考察を深めよう	
5.1 解析手法いろいろ	53
5.2 どうやったら精度が良くなるの?	60
おわりに	

1.

ひまわり 8号・9号

この章のポイント

- 衛星データの需要は年々高まり様々な分野に応用されている
- ひまわり 8号・9号のデータは可視光を含む 16 バンド
- CEReS ではこのデータを加工しグリッドデータとして提供している

1.1 歴史と CEReS の役割

気象衛星ひまわりは、1977年のひまわり1号（正式名称GMS, Geostationary Meteorological Satellite）の打ち上げを皮切りに、約50年間、日本領域を中心とした気象観測を行っています。2014年にはひまわり8号、2016年にはそのバックアップとしてのひまわり9号が打ち上げられ、現在は9号が運用されています。ひまわり8号・9号からは、可視画像による観測も行われるようになり、今まで見えてこなかった気象現象が鮮明に捉えられるようになりました。例えば、黄砂や火山灰の分布を捉えることができるようになり、より多様な予報・研究を行えるようになっていきます。衛星データの需要も産官学にかかわらず年々高まり、地球温暖化、氷域の変化、災害監視、砂漠化、植生量の評価、大気環境問題など、多くの問題に直接、間接に活用されるようになりました。

千葉大学のCEReS（環境リモートセンシング研究センター）は、リモートセンシングに関する中核的研究機関として、先端的研究の実施、地球表層環境変動の研究推進、社会貢献につながる研究の遂行という3つの重要な使命を担っています。CEReSは衛星データと地上観測データの総合環境情報拠点として、これらのデータの処理・アーカイブ・公開を行い、国内外の研究者との連携を積極的に推進しています。さらに、千葉大学の共同利用・共同研究センターとして、次世代の研究者育成にも力を入れており、地球環境研究の発展に大きく貢献しています。

1.2 性能

ひまわりには観測用のセンサーが16個搭載されており、それぞれをバンドと呼んでいます[4]。これらのバンドは以下のように分類され、観測できる項目が異なります。CEReSではこれらのデータをグリッドデータに加工し品質を管理することで、一般の人に使いやすいデータを提供しています。CEReSでのバンドは、以前のひまわり達に対応させるために、8号・9号が利用しているバンド名と異なる命名をしています[5]。本講座では、CEReSでのバンド名を利用します。

ひまわりバンド	CEReSでのバンド	波長(μm)	pixel x line	水平解像度(km)	観測項目
Band 01	VIS.01	0.47	12000x12000	1.0	植生、エアロゾル、カラー合成画像
Band 02	VIS.02	0.51	12000x12000	1.0	植生、エアロゾル、カラー合成画像
Band 03	EXT.01	0.64	24000x24000	0.5	植生、下層雲・霧、カラー合成画像
Band 04	VIS.03	0.86	12000x12000	1.0	植生、エアロゾル
Band 05	SIR.01	1.6	6000x6000	2.0	雲相判別
Band 06	SIR.02	2.3			雲粒有効半径
Band 07	TIR.05	3.9	6000x6000	2.0	下層雲・霧、自然火災
Band 08	TIR.06	6.2			上層水蒸気
Band 09	TIR.07	6.9			上中層水蒸気
Band 10	TIR.08	7.3			中層水蒸気
Band 11	TIR.09	8.8 ²			雲相判別、SO ₂
Band 12	TIR.10	9.6			オゾン全量
Band 13	TIR.01	10.4			雲画像、雲頂情報
Band 14	TIR.02	11.2			雲画像、海面水温
Band 15	TIR.03	12.4	雲画像、海面水温		
Band 16	TIR.04	13.3		雲頂高度	

参考文献

- [4] 放射計(AHI), https://www.data.jma.go.jp/mscweb/ja/info/spsg_ahi.html, 2025/02/27 閲覧.
- [5] 静止気象衛星ひまわりアーカイブ・ダウンロード情報, <https://www.cr.chiba-u.jp/japanese/database-himawari.html>, 2025/02/27 閲覧.

²文献によって8.6と記載されている箇所もあり要検証

2.

機械学習概要

この章のポイント

- ひまわりのデータから雲分類をする機械学習を検討するのが目標
- 機械学習モデルの基礎の K-means を用いたクラスタリングを採用
- クラスタリングはデータから特徴を抽出し分類する手法

2.1 本実習の目標

昨今、AIの時代と言われていますが、本演習ではその基礎の1つである K-means という手法を用いたクラスタリングを学びます。K-means を学び、今後も学習を続けていけば、生成モデルにたどり着きます。なるほどナと思えると思いますので、是非実装だけではなく、原理まで楽しんでみてください。³

本実習では Bishop (2006)[6] を参考に解説を行います。

2.2 K-means

2.2.1 概要

機械学習モデルをトレーニングするとき、モデルは入力に対する答えとなる「教師データ」が必要です。しかし、この教師データを準備することは容易ではありません。例えば、衛星データは時空間的に大変大きなデータであるため、一つ一つに正解ラベルを付与する労力は想像に難くありません。

一方で、教師データを用いない、教師なし学習という手法があります。その1つである K-means は、クラスタリングの代表的な手法です。クラスタリングとは、データセットを、データの特徴を元に幾つかのグループに分ける手法であり、教師データを必要としません。今回は、モデルを用いてデータを K 個のグループに自動的に分類することを考えましょう。

この実習では、特に断りが無いとき、小文字の変数はスカラー、太字はベクトルを表します。また、ベクトルは縦ベクトルを表します。

³もし興味があるのでしたら、この講義の後に EM アルゴリズムを勉強すると良いと思います。EM アルゴリズムは K-means と共通するような学習方法です。

クラスターってなに？

クラスターとは、何かの集まり・集団のことを指します。例えば、化学の教科書などで見た水晶の結晶は、小さな結晶の集まりです。また、ブドウの房のことを"a cluster of grapes"といたりします。



Figure 2. クラスターの例

(SNS 黎明期は同じ趣味を持つ人たちをクラスターなんて言ったりしました。アニメクラスターとかですね。懐かしい。)

2.2.2 アルゴリズム

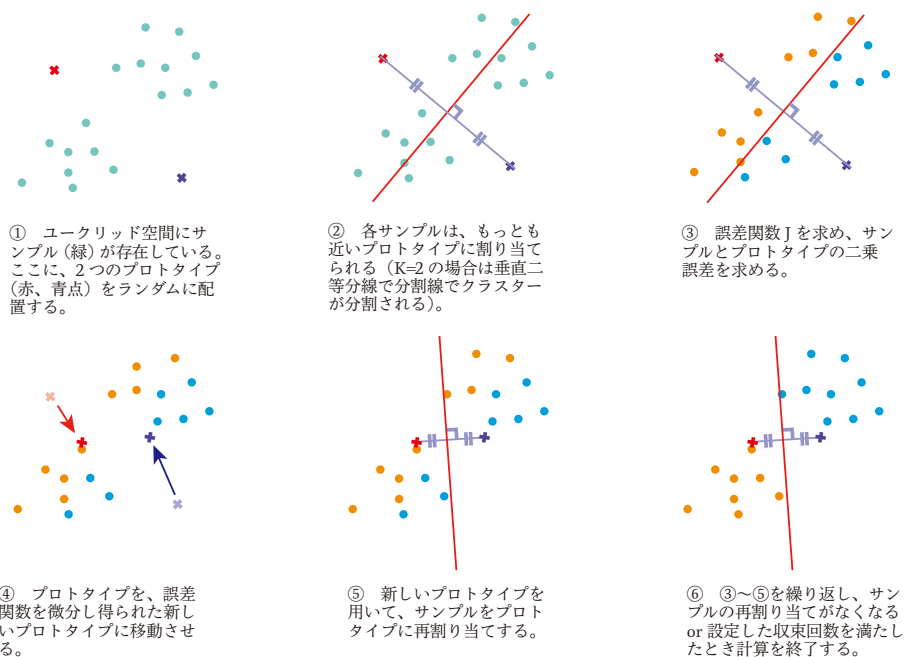


Figure 3. K-means の直感的理解

今、データセット \mathcal{D} があり、以下のように定義されるとします。

$$\mathcal{D}_n = \{d_1, d_2, d_3, \dots, d_n\} \quad (1)$$

これらのデータはユークリッド空間上に存在すると仮定します。このデータを、 K 個のクラスターに分類する事を目標とします。なお、この時の K は、ユーザーが任意に設定する必要があります。

ります。

まず、データが存在する空間に、ランダムに K 個のプロトタイプ $\mu_k (k = 1, 2, \dots, K)$ を配置します。 μ_k はクラスター k の中心を表します(Figure 3 の①)。

ここで、データ d_i がどのクラスターに割り当てられるかを表す変数として、二値変数 $r_{ik} \in \{0, 1\}$ を定義しましょう。 r_{ik} は、データ d_i がクラスター k に割り当てられるとき $r_{ik} = 1$ 、そうでないとき $r_{ik} = 0$ となります。

次に、アルゴリズムを考えるにあたり、目的関数(誤差関数)を定義しましょう。これは、データ d_i がクラスター k に割り当てられるとき、そのデータとプロトタイプ μ_k との距離の二乗の総和を取ったものとして定義されます。

$$J = \sum_{i=1}^N \sum_{k=1}^K r_{ik} \|d_i - \mu_k\|^2 \quad (2)$$

K-meansの目的は、この目的関数 J を最小化することですが、この最小化する方法として、 r_{ik} を適切に求めること、 μ_k を適切に求めることの2つが必要です。

アルゴリズムは2つのステップに分けられます。以下の2つを繰り返し r_{ik} と μ_k を交互に動かすことで、目的関数 J を最小化することができます。

ステップ1: プロトタイプへのデータの割り当て

配置した μ_k を固定します。このステップでは、 r_{ik} を決定し、それぞれのデータ d_i に対して、最も近いプロトタイプ μ_k を割り当てます(Figure 3 の②③)。数式で表すと、以下のようになります。

$$r_{ik} = \begin{cases} 1 & \text{if } k = \arg \min_k \|d_i - \mu_k\|^2 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

ステップ2: 誤差関数の最小化とプロトタイプの更新

r_{ik} を固定します。このステップでは、 J を最小化するように μ_k を更新します。いま、 J は μ_k について凸関数(下に凸な関数)であるため、 J を最小化する μ_k は、 J を微分したものを0にするものとして求めることができます。

$$\begin{aligned} \frac{\partial J}{\partial \mu_k} &= 2 \sum_{i=1}^N r_{ik} (d_i - \mu_k) = 0 \\ \therefore \mu_k &= \frac{\sum_{i=1}^N r_{ik} d_i}{\sum_{i=1}^N r_{ik}} \end{aligned} \quad (4)$$

この式を解くことで、 μ_k を更新することができます(Figure 3 の④)。この式は分子があるクラスター k に含まれるデータの総和を表し、分母がそのクラスターに含まれるデータの数を表しています。すなわち、プロトタイプとして準備した μ_k は、そのクラスターに含まれるデータの平均値を表していることになるのです。このあと、またステップ1に戻り、 d_i をプロトタイプに割り当てます(Figure 3 の⑤⑥)。

このような繰り返しにより、 J を最小化することができます。こうして得られた μ_k を用いて、 d_i を割り当てることで、クラスタリングを行うことができます。

ユークリッド空間

ユークリッド空間を厳密に定義するのは大変なので、本書では取り扱いませんが、ユークリッド空間とは、ユークリッド距離を用いて定義される空間のことを言います。ユークリッド距離は、私たちの直感に反しない距離です。

実数を n 個並べたもの全体の集合を \mathbb{R}^n とし、以下のように表します。

$$\mathbb{R}^n = \{(x_1, x_2, \dots, x_n) \mid x_i \in \mathbb{R}\} \quad (5)$$

この集合に対して、二点 p と q のユークリッド距離を以下のように定めます。

$$\|p - q\| = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} \quad (6)$$

このように距離が定義された空間をユークリッド空間といいます。

2.3 ひまわり 8 号・9 号のデータにどうやって適用するのか

今回の実習では、1 グリッドごとに、雲か雲でないか、或いはどんな雲かをクラスタリングします。つまり、上の解説で言う d_i が、あるグリッドのデータということになります。この d_i は、ひまわりのバンド全てを持っていますから、16次元のベクトルとなります。解説では二次元の時のクラスタリングを考えましたが、これを16次元にしたときにも考え方は同じです。16次元にすることで、データの情報を増やすことができます。

データをベクトルで表すということ

データをベクトルで表すと、例えばどういうことが分かるでしょうか？ ここでは簡単な例を用いて、データをベクトルで表すということを考えてみましょう。

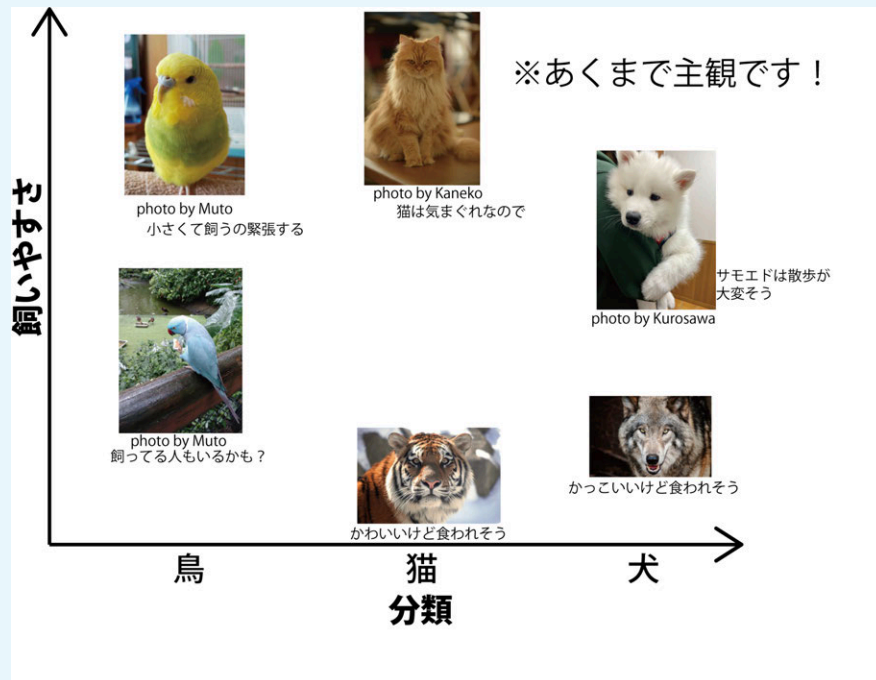


Figure 4. データをベクトルで表すということ

これは、特任助教や特任研究員の家族の写真と、野生の動物の写真を二次元にプロットしたものです。各次元は「分類」と「飼いやささ」を表しています。このようにデータの特徴を表す情報を数値化し、それをベクトルとしてまとめます。この特徴を増やしていくことで、多次元のデータができあがるのです。例えば、3次元目には「色」、4次元目には「食べるもの」を表す情報を加えることで多次元化できます。衛星データでしたら、これがバンドに対応します。このようにして、データをベクトルとして表すことができ、これをクラスター分析することができるのです。

参考文献

[6] Bishop, C. M. (2006). Pattern recognition and machine learning. New York:Springer.

3.

Xarray を用いた 巨大な衛星データのハンドリング

この章のポイント

- ひまわりのデータは巨大で取り扱いが大変
- Xarray を用いて領域の抽出や統計処理などデータを快適にあやつる
- データに入っているデータを確認し、それを可視化する

3.1 この実習に関して

本実習の醍醐味は、大規模なデータをいかに簡単にハンドリングし、計算速度を稼ぐかです。今回の実習では、Google Colaboratory を利用します。これは、Google LLC が提供する、無料でクラウドベースの Jupyter Notebook⁴環境です。ブラウザ上で Python のコードを実行できるため、環境構築の手間がなく、すぐにデータ分析を始めることができます。また、今回は使用しませんが、GPU (Graphics Processing Unit) を利用することが出来るため、深層学習を始めとする GPU を用いた計算にも利用することが可能です。無料で使える分、計算リソースも限られますが、このような環境であっても、Xarray を使った解析や学習などを容易に行うことができます。

冒頭にも書きましたが、今回は短時間の実習であり、計算時間・リソースともに限られるため、圧縮されたデータを数ステップのみ用いています。しかし、これから取り上げる手法は、データのサイズを大きくしていったときに、真価を発揮します。ぜひ、やり方を学び、ご自身の計算機などで試してみることをオススメします。

3.2 特筆すべきライブラリーの紹介とそのメリット

3.2.1 Xarray

このライブラリーは、気象データのような多次元データを扱うのに特化したライブラリーです。また、緯度経度座標を扱うときにも優れた力を発揮し、従来までは複雑で勘違いしやすいような座標の取り扱いを、非常にシンプルな手法で可能にします。可視化に関しても、Xarray から直接可視化することができるため、解析などの作業をより効率的に行うことができます。

⁴Jupyter Notebook は逐次 Python のコードを実行し確認できるので、練習にも実践にも便利です。

3.2.2 Dask

Daskは、大規模なデータを効率的に処理するためのライブラリーです。Pythonの標準的なデータ処理ライブラリーであるPandasやNumPyと同じような操作手段を持ちながら、並列処理による高速な計算・メモリに収まらない大規模データの処理・遅延評価による効率的な計算を可能とします。特に、今回のような衛星データの処理では、データサイズが大きくなりがちのため、Daskの特徴を活かすことができます。Xarrayとの相性も良く、よく一緒に使われています。

3.2.3 Zarr

Zarrは、多次元配列を効率的に保存・アクセスするためのデータのフォーマット、およびPythonライブラリです。⁵並列処理なども得意としていて、巨大な数値データを扱う科学計算、機械学習、などの分野で非常に役立ちます。今回のひまわりのデータセットは、筆者がZarr形式に変換したものです。

3.2.4 Cartopy

Cartopyは、地図の可視化を行うためのライブラリーです。本実習では、主に座標系の決定と海岸線の可視化に用います。

3.3 教材のコピーと Google Colab の起動

3.3.1 使用するデータ

本実習で使用する、ひまわり8/9号グリッドデータは千葉大学環境リモートセンシング研究センターで提供されたものです。このデータを更に加工したものを利用します。これは、ひまわりの全バンドの輝度温度のデータを、機械学習モデルで使いやすいようにZarr形式に変換したものが入っています。対象の時刻は、2022年1月15日3時00分(UTC, 以下断りがなければUTCとする)と、2022年1月15日5時00分(UTC)です。ひまわりのデータは、豊嶋(2021)[7]を参考に作成しました。これを、EXT.01は8分の1に、VIS.01-03は4分の1に、TIRやSIRは1/2にダウンサンプリングしています。ダウンサンプリングは単にデータを減らすだけでなく、データの平均値を計算することで、データの品質を極力維持しています。できあがったデータは、緯度×経度が3000×3000グリッドのデータになります。

⁵気象データはnetCDF4など色々ありますが、その内の1つです。

3.3.2 教材の準備と計算環境の準備

ここ (<https://drive.google.com/drive/folders/1V957bKnnGGNtQYFr4DpnmmBII9JzH-4g?usp=sharing>) から教材を3つコピーします。20220115.tar.gz はデータで、他が計算が記入されたファイルです。右クリックからコピーを作成をクリックしてください。



Figure 5. データのコピー

そうすると、自分のドライブにデータのコピーが作成されます。左上の「ドライブ」という文字をクリックして自分のドライブに移動し確認してみましょう。下の図のようになっていれば大丈夫です。

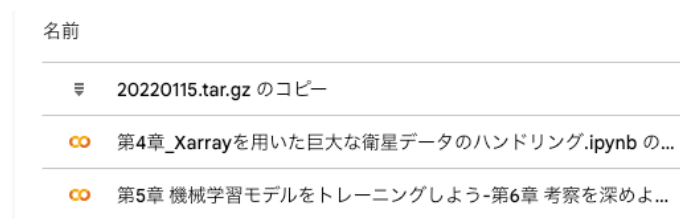
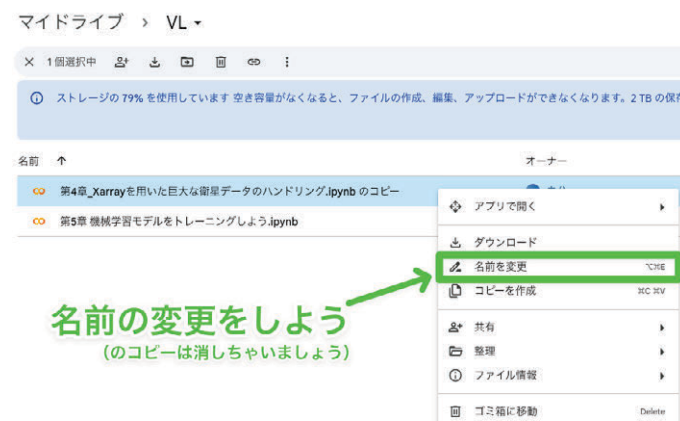


Figure 6. ファイルがコピーできたかの確認

この3つのファイルを、Google Drive の自分のフォルダーなどに移動させておくと、整理できて便利です。また、この際、ファイル名を変更してしましましょう。今回は、Google Drive 内に VL フォルダーを作って、そこにファイルを移動後、ファイル名を変更しました。「のコピー」⁶という文字列はいらないので削除しました。



⁶ 「の」の前に半角スペースが入っているので削除をお忘れなく！

Figure 7. ファイル名を変更した例

3.4 Google Colaboratory の起動とデータの解凍

さて、第4章の教材をダブルクリックで起動してみましょう。そうしたら、まずは一番上のセルを実行します。実行は、Shift+Enterを押すか、セルの右上にある再生ボタンを押します。⁷

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Python

これは、Google Drive をマウントして (1~2 行目)、そこにある圧縮ファイルを解凍する (4 行目) というコマンドです。Google Drive をマウントする際、マウントに関する許可を求められます。問題なければ許可をしてしまいましょう。



Figure 8. マウント許可その 1



Figure 9. マウント許可その 2

⁷Cntl+Enter でも実行できます。次のセルに移動しません。

さて、マウントしたら、先ほど保存した VL フォルダ内のデータを解凍します (Shift+Enter で実行)。保存先やファイル名は色々だと思いますが、次のプログラムのファイルの場所を変更することで、対応します。また、ファイルの場所はこの Figure 10 の様な手続きでコピーでき、簡単に下のプログラムに貼り付けることができます。

```
1 # ↓↓↓自分が保存した場所を指定し解凍する。 Python
2 !tar zxvf "/content/drive/Shareddrives/VL2025_ML-SAT/materials/20220115.tar.gz"
```

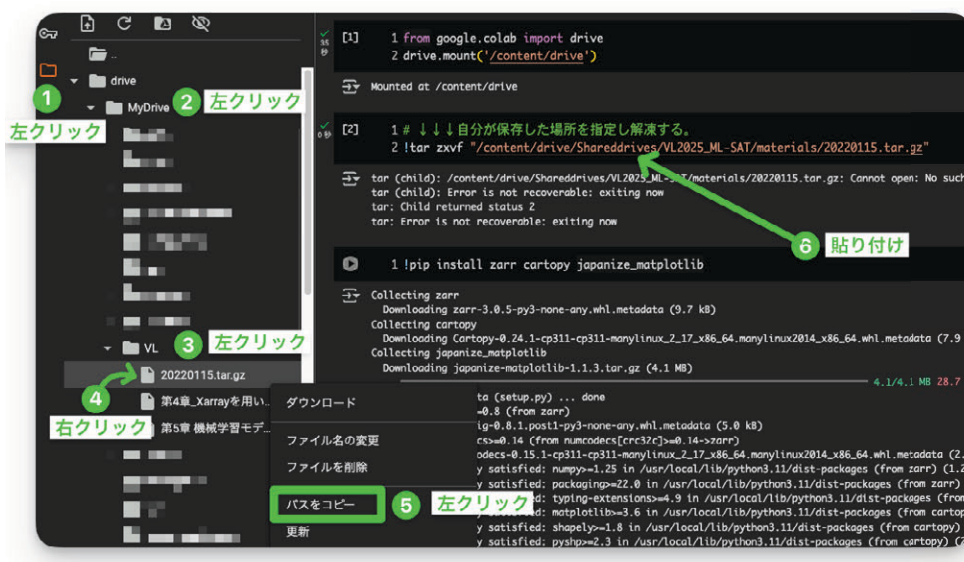


Figure 10. ファイルの場所の調べ方

3.5 Google Colaboratory に必要なライブラリーの準備をしよう

上で説明したライブラリーを導入するため、以下のコードを入力しましょう。入力したら、Shift+Enter を押して実行です。

```
1 !pip install zarr cartopy japanize_matplotlib Python
```

次々とインストールの経過が表示されると思いますが、そのまま完了するまで待ちましょう。

3.6 Xarray を使ってデータを読み込む

Xarray を用いて Zarr ファイルを読み込んでみましょう。読み込むためのコマンドはとても簡単で、以下のように一行で完了します(engine の内容を変えることで、netcdf4 などを読み込むこともできます)。

```
1 import xarray as xr Python
2
3 ds_vis = xr.open_dataset(
4     "/content/VIS_COARSE/himawari_data_2022011505_q.zarr",
```

```

5     engine="zarr",
6 ) # 可視光データの読み込み
7 ds_vis = xr.open_dataset(
8     "/content/TIR-SIR_COARSE/himawari_data_2022011505_h.zarr",
9     engine="zarr",
10 ) # 熱赤外データの読み込み

```

また、内容を確認する場合、Jupyter Notebook 上では単に変数名を記述し実行するだけでOKです。

```

1 """
2 どちらか1つを書いて実行してください (2つ書くとうまく表示されません)
3 """
4 ds_vis # 可視光データの内容を確認
5 ds_tir # 熱赤外データの内容を確認

```

ファイルの内容を表示すると以下のように見えます。データマークをクリックして、更に詳細に内容を確認することもできます。格納されている変数の大きさや型名などが分かるはずです。ds_vis と ds_tir でそれぞれ確認してみましょう。

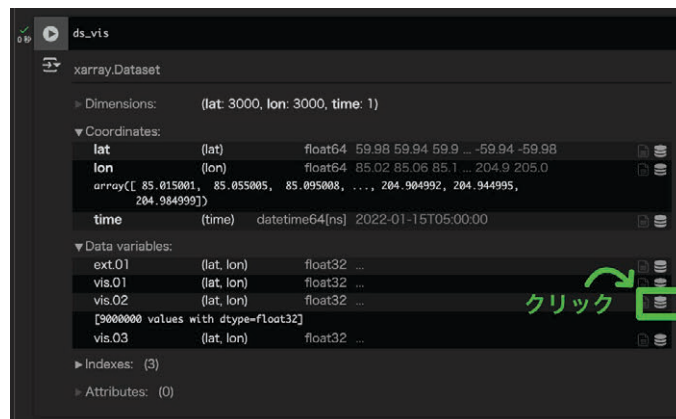


Figure 11. データセットの内容

ファイルの中身を見てみると、Dimensions, Coordinates, Data variables, Indexes, Attributes の6つの項目があることが分かります。これらが組み合わさって1つのデータとして整合しています。それぞれの項目の内容は以下のように表されます。⁸


⁸初学の頃は気にする必要がなく使って覚えた方が早いです。機会があればここを参照してみてください。

項目名	説明	例
Dimensions	データ配列の軸の名前を定義する。	time, latitude, longitude
Coordinates	各次元に沿った実際の座標値。次元の各位置に意味のある値を付与する。	time = ['2025-01-01', '2025-01-02'], latitude = [35.0, 35.5, 36.0], longitude = [122.0, 122.5, 123.0]
Data variables	実際のデータ値を保持する変数。N次元配列として格納される。	temperature, precipitation
Indexes	xarrayにおける特別な種類のCoordinates。データの効率的なアクセスと整列を可能にする。	時系列データの日付インデックスなど
Attributes	メタデータを格納。データセット全体や個々の変数に関する追加情報を提供する。	units = 'celsius', description = 'daily temperature'

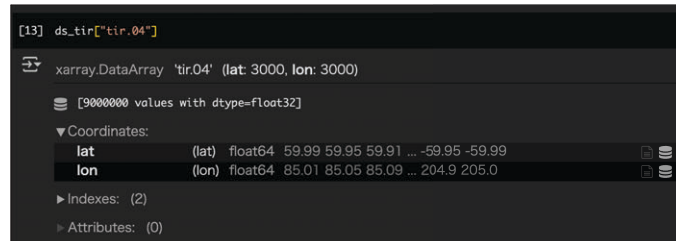
3.7 変数を抽出してみよう

上で見たように、このデータセットには、ひまわり8号・9号の観測変数が格納されています。これらそれぞれの変数を選択してみましょう。これは以下のように非常に簡単に実行できます。

```
1 ds_tir["tir.04"]
```

 Python

たしかに、抽出結果を見ると、以下のように変数が1つだけになっていることが分かります。




```
[13] ds_tir["tir.04"]
xarray.DataArray 'tir.04' (lat: 3000, lon: 3000)
[9000000 values with dtype=float32]
Coordinates:
  lat        (lat) float64 59.99 59.95 59.91 ... -59.95 -59.99
  lon        (lon) float64 85.01 85.05 85.09 ... 204.9 205.0
Indexes: (2)
Attributes: (0)
```

Figure 12. データセットの抽出

また、この抽出したものを、他の変数として紐付けることが可能です。

```
1 tir04 = ds_tir["tir.04"]
```

 Python

```
2 tir04 # tir04 だけが抽出されたデータセットになる
```

3.8 可視化してみよう

3.8.1 クイックルック

1つの変数を可視化するだけなら、とっても簡単です。以下のようなコマンドを実行してみましょう。適切なプロット方法が自動で選ばれてプロットすることができます。まずは、TIR.04 を可視化してみましょう。

```
1 ds_tir["tir.04"].plot()
```

Python

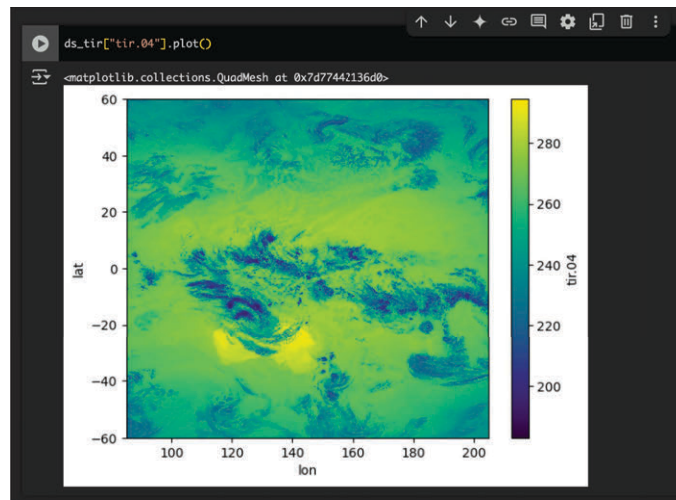


Figure 13. データセットの可視化

このように、データセットに対して、“[変数名]”としてやることで、指定した変数を抽出し、それを plot() という機能をもちいて簡単に可視化することができるのです。

輝度温度

Figure 13 では、カラーバーのラベルを見ると、tir.04 と書かれていますが、これは正確には等価黒体温度（Equivalent Blackbody Temperature）ですから、“TBB [K]”と書かれるべきでしょう。以下のように書けば、カラーバーのラベルを変更することができます。

```
1 ds_tir["tir.04"].plot(cbar_kwarg={"label": "TBB[K]"})
```

Python

このように、cbar_kwarg という引数を用いて様々な設定ができるほか、今後の内容でもカラーバーの細かい設定を色々な手法で行っています。本実習は入門者を対象とするため、厳密な書き方をしない箇所がありますが、留意した方が良いでしょう。

等価黒体温度の詳しい解説は、他の書籍に譲りますが ([7], [9])、これは、観測された赤外線放射量が、黒体から放射されたと仮定した時の、黒体温度のことを言います。雲頂を観測可能な波長を持った TIR.04 の観測を例に取りましょう。厚い雲が存在する場合、雲頂からの放射が大気の減衰を大きく受けずに衛星に到達します。そのため、TBB が低くなります。一方、薄い雲や観測解像度よりも小さな雲が存在する場合、雲頂温度から下方の放射も検知されるため、TBB は高く観測されます。ひまわり 8 号・9 号は様々なバンドを持ち、様々な TBB を観測可能なのです。

3.8.2 地図としてプロットしてみよう

クイックルックだけでも十分に綺麗な図が書けるのですが、これに更に、海岸線を追加したり、投影法を変えたり、地図として表示するには、以下のようにします。

```
1 fig = plt.figure(figsize=(12, 8))
2 ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree(central_longitude=180))
3 ds_tir["tir.04"].plot(ax=ax, transform=ccrs.PlateCarree())
4 ax.coastlines(linewidth=0.5, color="black", resolution="50m")
```

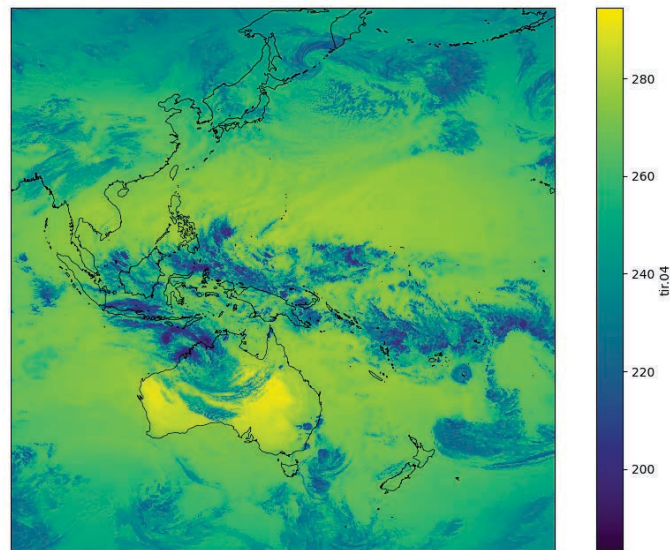


Figure 14. 地図としてプロット

ここで重要なのは、`projection` は出力データの座標系を設定し、`transform` は入力データの座標系を設定するという事です。CERESが変換したひまわりのデータは精密幾何補正処理を行い、緯度経度直交座標に変換しています。そのため、基本的には `transform=ccrs.PlateCarree()` としておけば良いです。

さらに、投影法を変えることで、地球儀のようなプロットをしてみましょう。

```
1 # Orthographic を使って地球儀のように表示
2 # projection は出力データの座標系を指定
3 fig = plt.figure(figsize=(12, 8))
4 ax = fig.add_subplot(
5     1,
6     1,
7     1,
8     projection=ccrs.Orthographic(
9         central_longitude=135,
10        central_latitude=20,
11    ),
12 )
13
```

```

14 # 全球表示
15 ax.set_global()
16
17 # transformは入力データの座標系を指定
18 ds_tir["tir.04"].plot(ax=ax, transform=ccrs.PlateCarree())
19
20 # 海岸線を追加
21 ax.coastlines(linewidth=0.5, color="black", resolution="50m")
22
23 # グリッド線を追加
24 ax.gridlines()

```

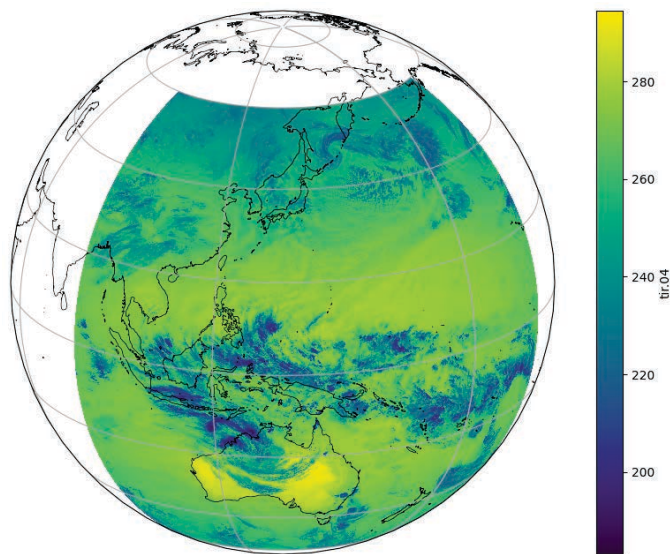


Figure 15. 地球儀のようなプロット

この例を見ていただくと、中心の経度を東経135度、中心の緯度を北緯20度にしたときのプロットだという事がプログラムよりわかります。このように、投影法を変えることで、様々なマップにプロットすることができます。Cartopyのprojectionリストのページを見てみると、これらの投影法の説明が載っています[8]。

3.9 領域を指定してみよう

上で表示してきたマップは、ひまわりのフルディスク観測のデータを全てプロットしています。そのため、より詳細に解析・可視化をしたい場合などは、領域を指定する必要があります。ここでは、日本周辺の矩形領域を抽出し可視化を行います。領域の抽出には、データセットのdimを使います。上の図で示したとおり、データセットにはdimが用意されており、これを指定（範囲指定）することによって、データの抽出が可能です。これは、selメソッドを使って簡単に利用可能です。まずは以下のコマンドを入力して領域を抽出してみましょう。

```

1 fig = plt.figure(figsize=(12, 8))
2 ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree(central_longitude=180))

```

Python


```

3
4 # 領域を sel メソッド使って抽出する
5 ds_tir["tir.04"].sel(lat=slice(46, 22), lon=slice(122, 149)).plot(
6     ax=ax, transform=ccrs.PlateCarree()
7 )
8 ax.coastlines(linewidth=0.5, color="black", resolution="50m")

```

```

[13] ds_tir["tir.04"]
xarray.DataArray 'tir.04' (lat: 3000, lon: 3000)
  [9000000 values with dtype=float32]
  Coordinates:
    lat          (lat) float64 59.99 59.95 59.91 ... -59.95 -59.99
    lon          (lon) float64 85.01 85.05 85.09 ... 204.9 205.0
  Indexes: (2)
  Attributes: (0)

```

Figure 16. 日本周辺の領域を抽出

緯度経度を確認してみてください。このように、日本の領域のみが抽出されました。slice は Python で範囲を指定するための関数ですが、ここでは詳細は省略します⁹。緯度は北から南へ格納されていますので、slice(22, 46)ではなく、slice(46, 22)と書きます（これはデータセットの性質によって変わります）。これらを plot で可視化すると、以下のような図になります。なんとなく、冬の雲の分布が見られると思います。

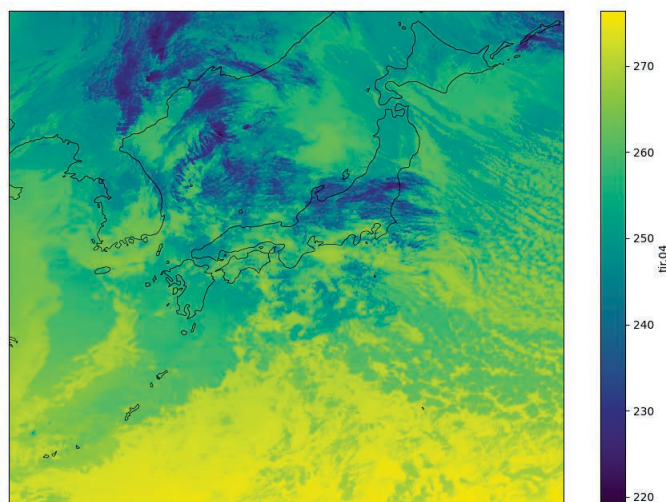



Figure 17. 日本周辺の領域を可視化

3.10 必要な場所を自在に抽出しよう

ここで、領域を指定するのではなく、点や線で指定する方法を考えます。例えば線で指定する場合は、以下のように指定します。

```
1 ds_tir["tir.04"].sel(lon=140.104042, method="nearest")
```

 Python

⁹こういう使い方ができるんだと覚える程度でまずは大丈夫です。

method という引数が付与されたことがわかります。これは、データに存在する座標に、指定した座標がない場合に、再近傍の点を自動的に取得してくれるというオプションです。これを plot で可視化すると、以下のような折れ線グラフとなります。

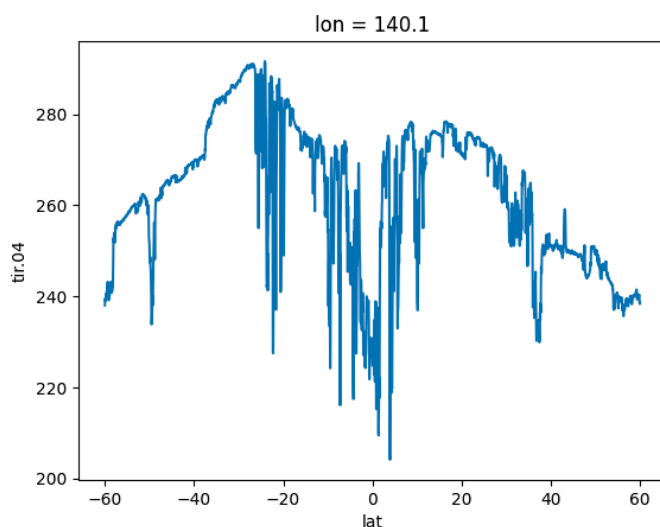


Figure 18. 特定の経度で抽出

このように、Xarray を用いると様々なデータを簡単に抽出できるようになります。気象データは、時間ごとのデータであったり、鉛直方向に積層されたデータもあったりするので、Xarray は単に解析をするだけでも（機械学習をしないのであっても）強力なツールとなるでしょう。

3.11 統計量を求めたりデータの性質を見てみよう

Xarray のデータセットは統計値の算出も容易に可能です。mean メソッドを使います。平均する対象を dim により指定が可能です。以下の例では、dim により“lat”と“lon”を選択、すなわち空間で平均しています。

```
1 ds_tir["tir.04"].mean(dim=["lon", "lat"])
```

Python

また、以下のように、末尾に values メソッドを付与することで、numpy.ndarray 形式で値を利用することも可能です。（他にもデータについても適用できますが、これは講義の最後で説明します。）

```
1 spacial_mean = ds_tir["tir.04"].mean(dim=["lat", "lon"]).values #
```

```
numpy.ndarray 形式へ
```

Python

```
2 spacial_mean # 結果を表示
```



Figure 19. 空間平均

また、以下のように様々な統計量を求められます。直感的にできてしまいます。

```
1 ds_tir["tir.04"].std(dim=["lat", "lon"]) # 標準偏差
2 ds_tir["tir.04"].max(dim=["lat", "lon"]) # 最大値
3 ds_tir["tir.04"].min(dim=["lat", "lon"]) # 最小値
```

様々な統計量を求められますので、気になる方は調べてみましょう。

さらに、サンプルのヒストグラムをプロットすることも可能です。

```
1 ds_tir["tir.04"].plot.hist(bins=20)
```

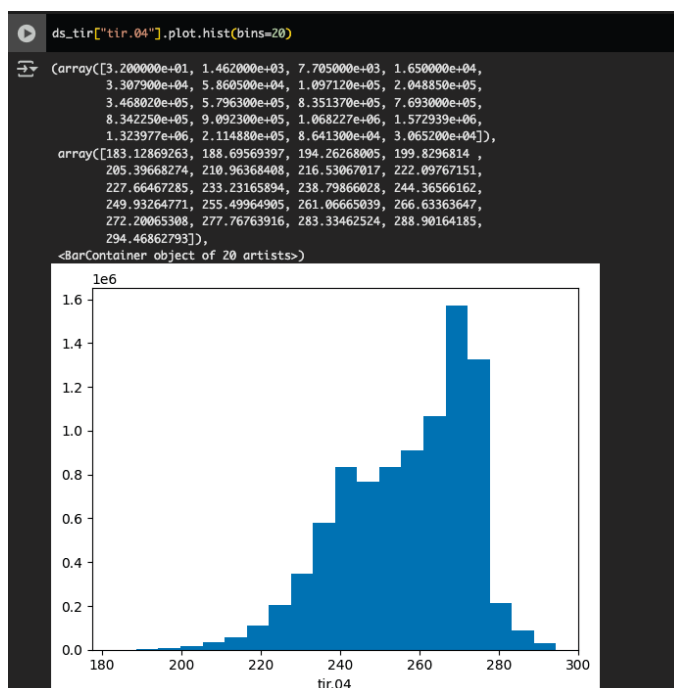


Figure 20. ヒストグラムの作成

3.12 複数のファイルを1つのファイルとして読み込もう

共通の次元を持つファイルは、1つのデータセットとして開くことができます。

実際のデータは、様々な形で保存されています。例えば、時間ごとに1つのファイルとして保存されていたり、地域ごとにデータが保存されていたり、その形態は様々です。そこで、これらのバラバラに保存されたデータを1つのデータとして開くことができるメソッドが用意されています。

```
1 file_list = glob("VIS_COARSE/*") + glob("TIR-SIR_COARSE/*")
2 print(file_list) # ファイルリストの表示
3 ds = xr.open_mfdataset(file_list, engine="zarr") # 1つのデータセットとして開く
```

読み込みが終わったらどんなデータが読み込まれたか確認しましょう (Figure 21)。

```
1 ds
```

Python

ここで、VIS_COARSE/に格納されているのは2つの zarr ファイルで、2022年1月15日の03時と05時のデータです。EXT.01とVIS.01-03の四変数がそれぞれのファイルに格納されています。また、TIR-SIR_COARSE/に格納されているのは、2022年1月15日の03時と05時のデータで、TIRとSIRのデータが格納されています。これを全て1つとして読むと、時刻が2時刻分格納された、16変数を持つ1つのデータセットとして取り扱うことが可能です。



Figure 21. 複数ファイルを1つのデータとして開く

これを表示してみると、確かに時間の次元が2に増えていることがわかります (Figure 21)。この時刻も、selを利用して抽出することができます。時刻のフォーマットは決まっていますが、以下のように指定すると良いでしょう。

```
1 ds.sel(time="2022-01-15 03:00")
```

Python

クイズ

第1問 :2022-01-15 05:00での、日本領域として上で示した範囲の平均値を求めてみましょう

領域を切り抜く方法と、平均する方法を上で学びました。最後に答えを得るときは、.compute()を計算式の最後に付けることによって、答えを算出することができますよ。

第2問 :2022-01-15 05:00において、経度方向に平均し、北から南までのTIR.04の緯度方向の変化を可視化しましょう。

できあがるグラフは、横軸が緯度、縦軸がTIR.04の輝度温度になるはずです。

第3問 :2022-01-15 03:00と2022-01-15 05:00のデータの経度200度に於けるTIR.04の差分を取ってプロットしましょう。

時間が違うデータの差分を取ることで、どの領域に変化があるのかを明らかにすることができます。これができるだけで、考察の幅が広がりますね。

3.13 RGB 画像の作成

今までは、TIR.04の可視化のみを行ってきましたが、ひまわり8号・9号の魅力の1つはRGBで撮影する美しい可視画像です。そこで、先ほどから使っているデータセットを使って、可視画像を作ってみましょう。ちょっとテクニックがあるので、予め関数を作成しておきました。これを Colab に貼り付けて実行しましょう。

```
1 Python
2 def make_rgb(ds: xr.Dataset, uint8_transform: bool = False) -> np.ndarray:
3     """xarray.Dataset から可視画像(RGB)データを抽出し、numpy 配列に変換する関数。
4
5     Args:
6         ds (xarray.Dataset): 可視画像データを含む xarray データセット。
7         uint8_transform (bool, optional): 8ビット整数(0-255)に変換するかどうか。デフォルトは False。
8
9     Returns:
10        numpy.ndarray: RGB 画像データ。uint8_transform=True の場合は 0-255 の整数値、
11        False の場合は 0-1 の浮動小数点数。
12    """
13    assert ("time" not in ds.coords) or ds.time.shape == (), (
14        "一時刻のデータを設定してください。現在の time 次元は" + str(ds.time.shape) + "です。"
15    )
16
17    rgb_stack = np.dstack(
18        [
19            ds["ext.01"].values, # Red channel
20            ds["vis.02"].values, # Green channel
21            ds["vis.01"].values, # Blue channel
22        ]
23    )
24
25    # データが 0-1 の範囲になるように正規化
26    # 必要に応じて、min-max 値を調整してください
27    rgb_normalized = np.clip(
28        (rgb_stack - np.nanmin(rgb_stack)) / np.nanmax((rgb_stack) -
29            np.nanmin(rgb_stack)), 0, 1
30    )
31    if uint8_transform:
32        rgb = (rgb_normalized * 255).astype(np.uint8)
33    else:
34        rgb = rgb_normalized
```

```

35     return rgb
36
37
38 def gamma_correction(
39     image: np.ndarray | xr.DataArray | xr.Dataset, gamma: float = 1.0
40 ) -> np.ndarray:
41     """ガンマ補正を行う関数。
42
43     Args:
44         image (numpy.ndarray): ガンマ補正を行う画像データ。
45         gamma (float, optional): ガンマ値。デフォルトは1.0。
46
47     Returns:
48         numpy.ndarray: ガンマ補正が適用された画像データ。
49     """
50     epsilon = 1e-10
51     if isinstance(image, (xr.Dataset, xr.DataArray)):
52         image = xr.where(image <= 0, epsilon, image)
53     else:
54         image = np.where(image <= 0, epsilon, image)
55
56     return image ** (1 / gamma)

```


画像というのは、3次元テンソル（行列を重ねたもの）のようなもので、（バンド数, xグリッド数, yグリッド数）のような形をしています。ひまわり8号・9号のデータにもRGBに対応するバンド（EXT.01, VIS.02, VIS.01）がありますから、これを3バンドのデータとして作成するのが、この関数の役割です¹⁰。Pythonに詳しい方は関数の中身を追うと良いと思いますが、そうで無い方は、最初は利用するだけで大丈夫です。

また、二つ目の関数は、ガンマ補正を行うための関数です。ガンマ補正を行わない可視画像は、暗い場合があります。そのため、目で見て分かりやすいように、ガンマ補正を行うことがあります。この関数も、初学の方は追わなくても良いでしょう。画像を可視化する `plt.imshow` を用いて、作成したRGB画像を可視化します。

```

1 rgb = make_rgb(ds.isel(time=0)) # RGBデータの取得
2 rgb_gamma = gamma_correction(rgb, gamma=1.4)
3 plt.imshow(rgb_gamma)

```

 Python

¹⁰RGB合成画像だけでも色々な種類があります[10]

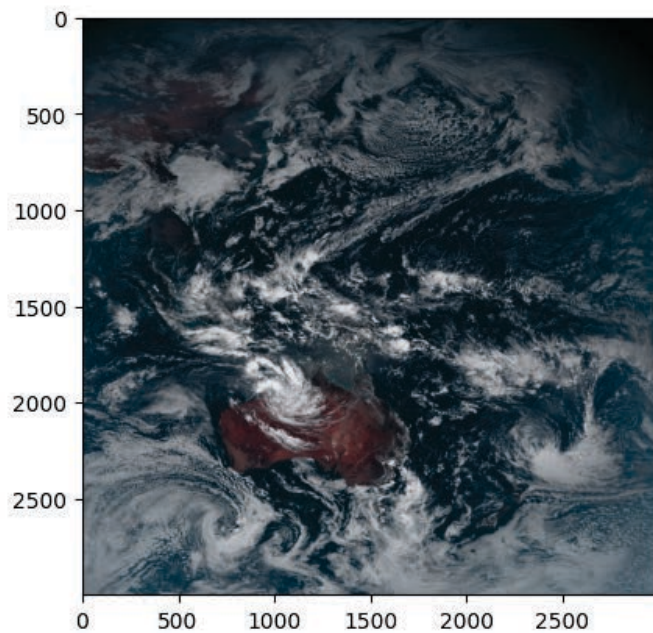


Figure 22. RGB 画像の作成

一方、これでは地図であるのにもかかわらず、緯度経度の情報と対応することができません。そこで、作成した RGB のデータを、データセットに加えてしまいましょう。こうすることで、地図として描画するときに便利になります。

```

1  # データセットに RGB 画像を加える
2  ds["rgb"] = (("lat", "lon", "rgb_channel"), rgb_gamma)
3
4  # ==== 可視化 ====
5  fig = plt.figure(figsize=(12, 8))
6  ax = fig.add_subplot(1, 1, 1,
7  projection=ccrs.PlateCarree(central_longitude=180))
8  # RGB 画像の表示
9  ax.pcolormesh(ds["lon"], ds["lat"], ds["rgb"].values,
10 transform=ccrs.PlateCarree())
11 # 海岸線の表示
12 ax.coastlines(linewidth=1.0, color="lightgreen", resolution="50m")
13
14 # グリッドの表示
15 gl = ax.gridlines(
16     color="white",
17     linestyle="--",
18     linewidth=0.5,
19     draw_labels=True,
20 )
21 gl.top_labels = False # 上の経度ラベルを非表示
22 gl.right_labels = False # 右の緯度ラベルを非表示

```

```

23 gl.xlocator = plt.FixedLocator(np.arange(-180, 181, 20)) # 経度間隔を 20 度に
24 gl.ylocator = plt.FixedLocator(np.arange(-90, 91, 20)) # 緯度間隔を 20 度に
25 gl.xlabel_style = {"size": 10} # 経度ラベルのサイズ
26 gl.ylabel_style = {"size": 10} # 緯度ラベルのサイズ

```

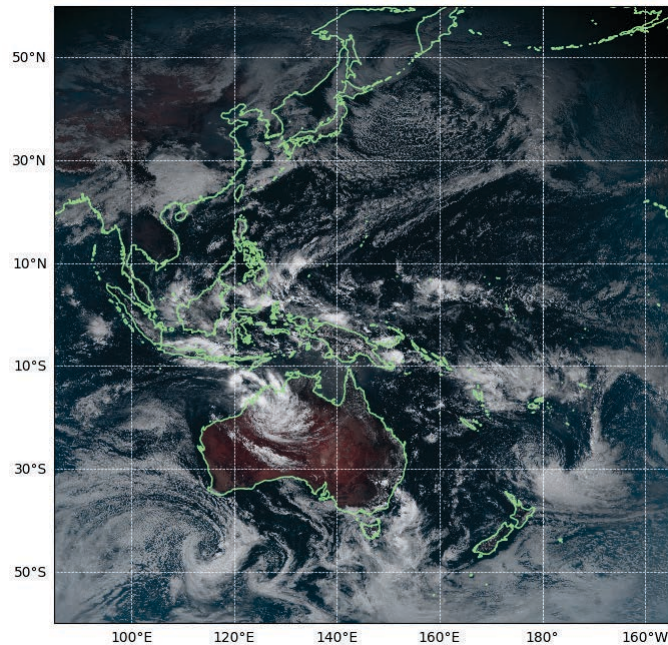


Figure 23. RGB 画像の可視化

これを実行すると、Figure 23 のような画像が表示されます。緯度経度線や海岸線も表示されて、地図らしくなりました。¹¹

クイズ

日本の領域の可視画像を表示してみよう

今までにやってきた方法を使って、可視化することができるはずです。領域を指定するには、sel メソッドが便利です。

トンガの噴火を見てみよう 使用しているデータは 2022 年 1 月 15 日のものです。この日は、トンガが噴火した日です。トンガの領域に絞って可視化することで、噴火を鮮明に捉えられるかもしれません。

3.14 zarr 形式への書き出し方（発展）

https://ceres.chiba-u.jp/vl_lecture/の実習教材「Python によるひまわり 8 号衛星データの可視化」などを用いて作成した、ひまわりのデータを Zarr 形式として保存しましょう。ひまわり 8 号のデータが既に、np.ndarray として読み込まれ、輝度温度に変換された後、二次元の

¹¹地図にこのようなプロットをしたことがある方なら、Xarray の座標の取り扱いが大変楽なことに気付いていただけたと思います。

グリッドデータとして格納されているとして、以下のようにして Zarr 形式として保存することができます。

```
1 import datetime as dt
2
3 import pandas as pd
4 import xarray as xr
5 import zarr
6
7
8 # 対象の時刻
9 target_date = dt.datetime(2022, 1, 15, 5, 0)
10
11 # 使用するデータ
12 print(himawari_tir04.shape)
13 print(himawari_tir05.shape)
14 # (6000, 6000) tir.04, tir.05 の 1 時刻のデータ
15
16 # データを次元の情報とともに辞書に格納
17 data_vars = {
18     "tir.04": (("lat", "lon"), himawari_tir04),
19     "tir.05": (("lat", "lon"), himawari_tir05),
20 }
21
22 # 緯度経度データの作成
23 IR_LON = np.linspace(85, 205, 6000)
24 IR_LAT = np.linspace(60, -60, 6000)
25
26 # xarray.Dataset の作成
27 ds = xr.Dataset(
28     data_vars=data_vars,
29     coords={
30         "lat": IR_LAT,
31         "lon": IR_LON,
32         "time": pd.to_datetime([target_date]),
33     },
34 )
35
36 # 保存するための設定。圧縮率とチャンクサイズ。
37 encoding = {
38     var: {
39         "compressor": zarr.Blosc(cname="zstd", clevel=5),
40         "chunks": (1000, 1000, 1),
41     }
42     for var in target_channel
43 }
```

```
44
```


```
45 # データを Zarr 形式に書き出し
```

```
46 ds.to_zarr(zarr_path, mode="w", encoding=encoding)
```


手元の netCDF4 を Zarr に変換する

netCDF4 は、気象分野の人にはおなじみのデータフォーマットです。これを Zarr に変換することで、効率的にデータにアクセスできることがあります¹²。

netCDF4 は、Xarray で以下のように読み込みます。engine オプションで netcdf4 を指定します。事前に pip など Python 用の netcdf4 をインストールしておく必要があります。

```
1 ds = xr.open_dataset("data/himawari_data.nc", engine="netcdf4")  Python
```

これを Zarr に変換するには、以下のようにします。

```
1 ds.to_zarr("data/himawari_data.zarr")  Python
```

3.15 Himawari Bench の紹介

現在、著者らは「Himawari Bench (仮)」というベンチマークセットを開発中です。これは、気象衛星ひまわりのデータを Zarr 形式に落とし込んだもので、機械学習のモデルを評価するためのデータセットとして利用可能です。完成次第公開する予定ですので、是非ご利用ください。

参考文献

- [7] Python によるひまわり 8 号衛星データの可視化, 豊嶋紘一, https://drive.google.com/drive/folders/1bpXEU_MudvBhF00H59Da4AjchE83cY82, 2021.
- [8] Cartopy projection list, <https://scitools.org.uk/cartopy/docs/v0.15/crs/projections.html>, 2025/02/28 閲覧.
- [9] 気象衛星センター. (2002). 気象衛星画像の解析と利用 - 航空気象編 -. https://www.data.jma.go.jp/mscweb/ja/prod/pdf/book/book_koukukisho_c.pdf
- [10] 気象衛星センター. 「画像活用の目次」. <https://www.data.jma.go.jp/mscweb/ja/prod/product.html>, 2025/03/11 閲覧.

¹²今までに紹介した Xarray を使う手法は、Zarr に変換せずに netCDF4 を用いるだけでも利用可能です。

4.

機械学習モデルを トレーニングしよう

この章のポイント


- グリッド毎にどんな雲が存在するかを分類する k-means を作成
- 各グリッドのデータを 1 サンプルとして合計 9,000,000 サンプルの全バンド 16 次元のベクトルをクラスタリングする
- RGB 画像と重ねてクラスタリングの結果を可視化する

4.1 データと計算環境の準備

前の章を参考にして、データと計算環境の準備をしましょう。まず、新しい Jupyter Notebook を作成します。Colab の画面の左上の「ファイル」をクリックし、「新しいノートブック」を選択します。これで、新しい Notebook が作成されます。前章の復習も兼ねて、以下にコードを示します。


必要なパッケージのインストール

```
1 !pip install zarr cartopy dask_ml japanize_matplotlib
```

 Python


以下のコマンドより、データの解凍を行います。

```
1 !tar -zxvf data/data.tar.gz
```

 Python

必要なライブラリーのインポートと関数の定義、データの読み込みを行います。(今後も必要に応じて適宜 import していきます。)

```
1 import xarray as xr
2 import zarr
3 import cartopy.crs as ccrs
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import japanize_matplotlib
7
8 def make_rgb(ds: xr.Dataset, uint8_transform: bool = False) -> np.ndarray:
9     ... # 前章と同じ
10    ...
11    return rgb
12
13
14 def gamma_correction(
```

 Python

```

15     image: np.ndarray | xr.DataArray | xr.Dataset, gamma: float = 1.0
16 ) -> np.ndarray:
17     ... # 前章と同じ
18     ...
19     return image ** (1 / gamma)
20
21
22 # データの読み込み
23 file_list = glob("VIS_COARSE/*") + glob("TIR-SIR_COARSE/*")
24 print(file_list) # ファイルリストの表示
25 ds = xr.open_mfdataset(file_list, engine="zarr") # 1つのデータセットとして開く

```


さて、これで第4章のように、下準備は整いました。以下からは、K-meansの基礎的な実装をやっていきましょう。まずは、簡単な実装例を通して、K-meansの基本的な流れを理解しましょう。¹³

次に、`sel`メソッドを用い、利用するデータの時刻を選択します。今回は、2022年1月15日05時のデータを選択します。元のデータセット`ds`を書き換えると後々考察なども行う際にデータが使えなくなってしまうので、新しい変数`ds_analysis`を作成します。

```

1 ds_analysis = ds.sel(time="2022-01-15 05:00:00")

```


 Python

このデータを、各グリッドごとの各バンドのデータにするために、 $9,000,000 \times 16$ のデータになるように変換します。16次元、9万個のグリッドデータ（すなわち9万個の学習サンプル）になります。

```

1 # データセットを dask array に変換し、lat, lon をスタック
2 stacked_data = da.vstack(
3     [
4         ds_analysis[var].stack(pixels=("lat", "lon")).data.astype("float32")
5         for var in ds_analysis.data_vars
6     ]
7 )
8 # (9000000, 16)の形に転置
9 X = stacked_data.T

```


 Python

次に、チャンク数を定義します。チャンク数とは、Daskにおける並列処理の最小単位です。これを定義することによって、Daskが並列処理を行う際の最小単位を決めることができます。

```

1 # チャンクを適切に設定
2 # 1軸目（サンプル）は100000ごと、2軸目（特徴量）は全体を1チャンクに
3 CHUNK_SIZE = 100000
4 X = X.rechunk((CHUNK_SIZE, -1))

```

 Python

¹³前章と同じように、細かな理解よりも、実装の流れを理解することを優先しましょう。Pythonの書き方などは、書いていくうちに覚えるものです。

学習データの中の各バンドは、そのバンドによって統計値が異なります。そのため、もしかしたら、あるバンドに極端に引っ張られてしまい、K-means が上手く学習できないことがあります。そのため学習の前に、データの標準化を行いましょ。標準化は、以下の式により定義されます。

$$z = \frac{x - \mu}{\sigma} \quad (7)$$

ここで、 x は、それぞれのグリッドに於けるバンドの値をベクトルにしたもの、 μ は、 x のバンド平均、 σ は、 x のバンド標準偏差を表します。この式を観察すると、それぞれのグリッドに於けるデータの分布は、平均0、分散1のデータに変換されることがわかります。このようにして、各バンドによるデータの分布の偏りを標準化し、クラスタリングをより正しい評価で行えるようになるのです。

これを適用するには、自分で平均や分散を求め変換を行う関数を書いても良いですが、Dask の StandardScaler を使用すると、より簡単かつ効率的に計算することが可能です。

```
1 # データの標準化 Dask の StandardScaler を使用 Python
2 scaler = StandardScaler()
3 X_scaled = scaler.fit_transform(X)
```

scaler の fit_transform メソッドは、データの標準化を行います。これにより、データの平均が0、分散が1になります。scaler の中には、計算に用いた平均や分散が保存されています。

```
1 scaler.mean_, scaler.var_ # 平均と分散を確認 Python
2 """こういう風に表示されると思います。16次元のベクトルになっていますね。
3 (array([ 15.925109,  20.137419,  18.078812,  17.650541,  12.265163,
4          9.442699, 272.03745 , 271.17715 , 268.68463 , 256.83273 ,
5          283.30933 , 234.14215 , 242.77388 , 249.69768 , 269.68967 ,
6          251.50043 ], dtype=float32),
7  array([255.47043 , 258.32532 , 247.82205 , 317.97324 , 146.52873 ,
8          87.72339 , 468.30597 , 478.11847 , 430.42798 , 260.99216 ,
9          339.75766 ,  74.911476, 113.10117 , 153.00252 , 434.95862 ,
10         240.38736 ], dtype=float32))
11 """
```

さて、ここまではデータの準備でした。次からはK-means を準備して、実際に計算を行わせてみましょう。

4.2 K-means の学習

さて、ここまで来たらあと少しです。まず、K-means を実装するために、何個のクラスターに分類するかを定義をしましょう。

```
1 K = 10 # クラスタ数 Python
```

そして、K-means の学習を行うために、以下のようにモデルを定義します。

```
1 from dask_ml.cluster import KMeans
2 kmeans = KMeans(n_clusters=K, random_state=0)
```

Python

これで準備は完了です。この `dask_ml` の `KMeans` は、`Dask` の並列処理を用いて、高速に大規模なデータの計算を行うことができます。

それでは、実際に計算を行わせてみましょう。

```
1 kmeans.fit(X_scaled)
```

Python

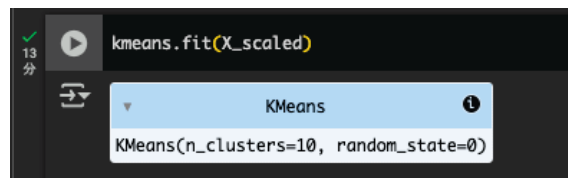


Figure 24. 計算終了

計算が終了すると Figure 24 のように表示されます。計算機のスペックにも依存しますが、時間が掛かると思えます。参考までに、金子が使っているちょっと速い計算機 (Ryzen 9 5900X, RAM 128GB, RAID6 HDD) では、この計算で3分くらい掛かりました。Google Colab の環境では、さらに時間が掛かると思われます。¹⁴

4.3 結果の可視化

4.3.1 結果の格納と簡単な可視化

計算の結果を可視化してみましょう。それぞれのグリッドのラベル (クラスター番号) として格納されていて、 $K=10$ としたので、 $0 \leq k \leq 9$ の範囲のラベルが付与されています。計算結果を確認してみましょう。結果は、`kmeans.labels_` に格納されています。

```
1 kmeans.labels_
```

Python

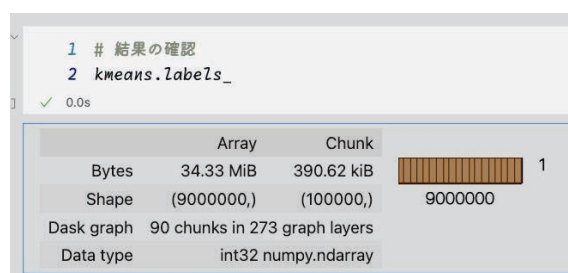


Figure 25. 計算結果の確認

¹⁴ 計算機の混み具合にもよるみたいで、大変な時は Colab で 13 分ほど掛かりました。人間はその間に勉強しておきましょう。

これを見ると、合計9,000,000グリッドのデータが、1次元の配列に格納されているようです。これを改めて地図（3000×3000グリッドの2次元画像）に戻すために、以下のような処理を適用し、ds_analysisにclusterという変数名で格納しましょう。¹⁵

```
1 ds_analysis["cluster"] = (("lat", "lon"), kmeans.labels_.reshape((3000, 3000)))
```

さて、変数 cluster を可視化してみましょう。

```
1 ds_analysis["cluster"].plot()
```

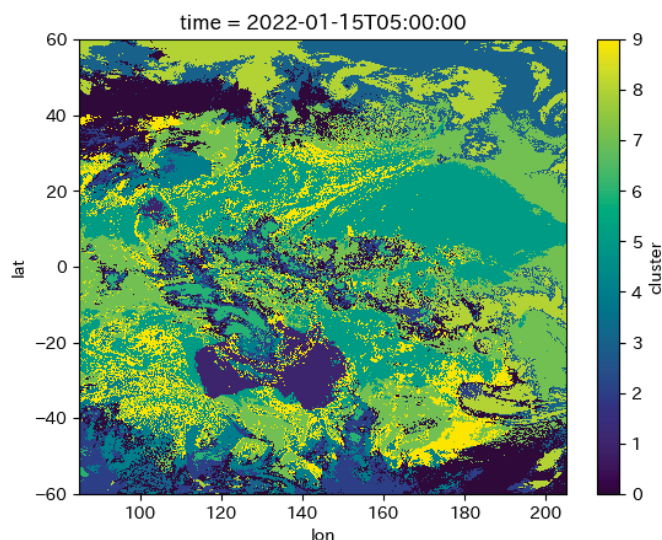


Figure 26. 計算結果の可視化

4.3.2 クラスタリング結果を見やすくしよう

ここまでは、簡単にクラスタリングの結果を可視化しました。たった数行で簡単に可視化できるのが、Xarrayの良いところですが、更にこだわった可視化をして考察をしやすくしましょう。¹⁶

現段階で、簡易的な可視化のクオリティを上げるためには、以下の2つが大切でしょう。

1. 地図（海岸線）の可視化
2. 投影法に沿った可視化
3. カラーマップの改善

1, 2に関しては、前章でやったとおりです。cartopy を用いて海岸線を可視化し、地図を投影法に沿って可視化することができます。

```
1 fig = plt.figure(figsize=(12, 8))
2 ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree(central_longitude=180))
3 ax.coastlines() # 海岸線の描画
```

¹⁵Python に詳しい方ですと、データの格納方法がちょっと変わっているなど思うかもしれません。

¹⁶綺麗に可視化できると気持ちもスッキリしますネ。

```

4 ax.gridlines() # 緯度経度の目盛りの描画
5 im = ds_analysis["cluster"].plot(transform=ccrs.PlateCarree()) # クラスタリング結果の可視化
6 plt.title("クラスタリング結果") # タイトルの描画

```

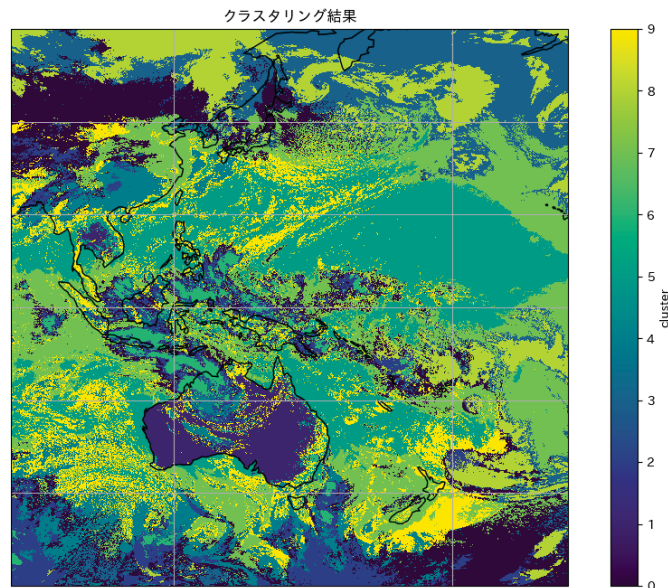


Figure 27. 海岸線の描画と投影法に沿った可視化

地図が書かれてグリッド線が書かれるだけでも、見やすくなりました。例えば、オーストラリア大陸の殆どは、紫色のクラスターで覆われていることがわかります。

一方、カラーマップの問題で、カテゴリー0と1の区別が付きづらかったりと、隣り合ったカテゴリー同士の区別が難しいです。今は $k=10$ ですが、これが増加したとき、隣り合ったクラスターの区別は更に難しくなることは、想像に難くありません。これは、matplotlibに用意されたカラーマップが、量的データの可視化に向けて作成されており、カテゴリー値のような質的データ、とりわけ名義尺度のデータに対しては、適切に可視化ができないことが原因です。

そこで、カテゴリー値用のカラーマップ、tab10やtab20などを用いることで、より見やすい可視化を行うことができます。以下では、tab10を用いた可視化の例を示します。

```

1 fig = plt.figure(figsize=(12, 8))
2 ax = fig.add_subplot(1, 1, 1,
3 projection=ccrs.PlateCarree(central_longitude=180))
4 ax.coastlines()
5 ax.gridlines()
6 im = ds_analysis["cluster"].plot(
7     transform=ccrs.PlateCarree(), cmap="tab10", vmin=0, vmax=9,
8     add_colorbar=False
9 )
10 cbar = plt.colorbar(im, label="クラスタ")
11 cbar.set_ticks(np.linspace(0.4, 8.6, 10))
12 cbar.set_ticklabels(range(10))
13 plt.title("クラスタリング結果")

```

Python

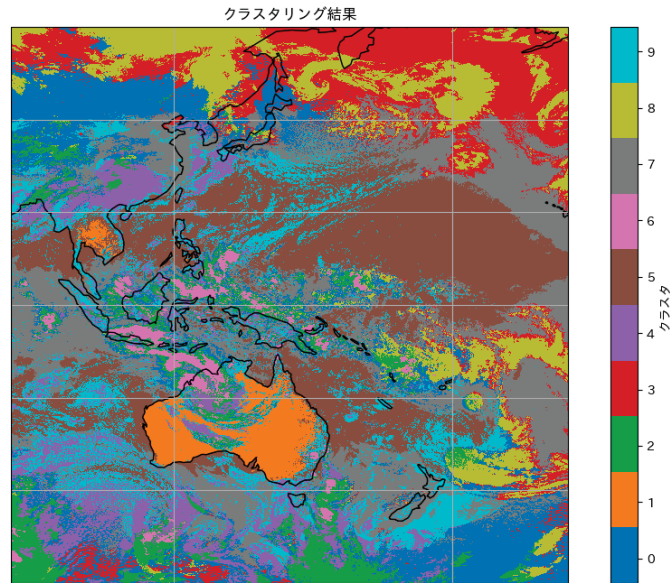


Figure 28. カラーマップの改善

これを見ると、例えばオーストラリアで見られたカテゴリ 1 とモンゴルや南極圏で見られたカテゴリ 0 は別のクラスターだということがはっきりわかるようになりました。¹⁷

4.3.3 実際の雲と重ねて表示してみよう

これまで、単にクラスタリングの結果を表示しただけでしたが、実際の雲の画像と比較することで、よりクラスタリングについて考察することができます。そこで、RGB 可視画像とクラスタリング結果を重ねて表示してみましょう。まずは、前章で作成した RGB 画像を同じ手順で作成します。

```
1 rgb = make_rgb(ds_analysis)
2 rgb_gamma = gamma_correction(rgb, 1.4)
3 ds_analysis["rgb"] = (("lat", "lon", "rgb_channel"), rgb_gamma)
```

Python

これをクラスタリングの結果と重ねて表示してみましょう。ここでは、クラスタリングの結果と RGB を同時に可視化しますが、クラスタリングの結果のカラーマップに `alpha` を設定することで、RGB 画像の透過度を調整します。ここでは、クラスターのカラーマップの透過度を 0.35 に設定しています。

```
1 fig = plt.figure(figsize=(12, 8))
2 ax = fig.add_subplot(1, 1, 1,
3 projection=ccrs.PlateCarree(central_longitude=180))
4 ax.coastlines()
5 ax.gridlines()
6 ax.pcolormesh(
```

Python

¹⁷tab10 などカテゴリのカラーマップはカラーバーの作成にテクニックが必要です。コード内の `cbar` の設定がそれに当たります。

```

6     ds_analysis.lon, ds_analysis.lat, ds_analysis["rgb"].values,
7     transform=ccrs.PlateCarree()
8 )
9 im = ds_analysis["cluster"].plot(
10     transform=ccrs.PlateCarree(),
11     cmap="tab10",
12     vmin=0,
13     vmax=9,
14     add_colorbar=False,
15     alpha=0.35, # クラスタリングの結果の透過度を見やすいように調整
16 )
17 cbar = plt.colorbar(im, label="クラスタ")
18 cbar.set_ticks(np.linspace(0.4, 8.6, 10))
19 cbar.set_ticklabels(range(10))
20 plt.title("クラスタリング結果")

```

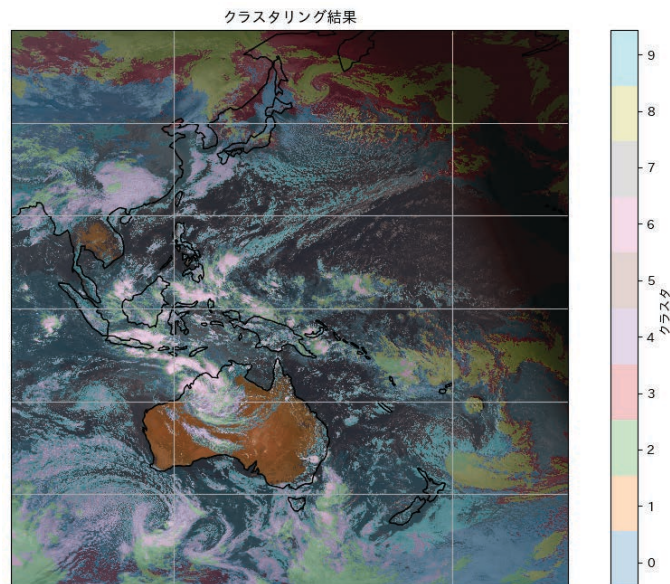


Figure 29. クラスタリング結果と RGB 画像の重ね合わせ

その結果、Figure 29 のような結果が得られます。例えば、赤道周辺の雲頂の高い雲はクラスター番号6に割り当てられ、日本の冬の雲はクラスター番号0や3などに割り当てられていることがわかります。

4.3.4 雲ではないクラスターの除去

Figure 29 を見ていると、どうやら海や地面が見えているところも何かしらのクラスターに分類されていることがわかります。これは、雲を対象としたクラスタリングの場合は表示しなくてもよいクラスタとなりますから、これを除去するようにプログラムを書きましょう。今回は、クラスターを除去しプロットをする関数を提供します。Python に詳しい方は中身を見て

何をやっているかを理解すると良いでしょう。初心者の方は、関数が使えるだけで十分です。¹⁸

```
1  import warnings
2  from matplotlib.colors import ListedColormap
3  import matplotlib.pyplot as plt
4  import xarray as xr
5  import numpy as np
6  import cartopy.crs as ccrs
7
8
9  def visualize_clustering(
10     ds: xr.Dataset,
11     original_cluster_num: int,
12     title: str = "クラスタリング結果",
13     cluster_var: str = "cluster",
14     exclude_clusters: list = None,
15     rgb_var: str = "rgb",
16     figsize: tuple = (12, 8),
17     dpi: int = 100,
18     alpha: float = 0.25,
19     custom_colors: dict = None,
20     plot_cluster: bool = True,
21     plot_rgb: bool = True,
22     plot_grid: bool = True,
23     grid_spacing: int = 10,
24     grid_color: str = "white",
25     grid_style: str = "--",
26     grid_linewidth: float = 0.5,
27     plot_coast: bool = True,
28     coast_color: str = "lightgreen",
29     coast_linewidth: float = 0.5,
30     save_path: str = None,
31     subplot_index: tuple = (1, 1, 1),
32 ) -> tuple:
33     """クラスタリング結果を可視化する関数
34
35     Args:
36         ds (xr.Dataset): 入力データセット
37         original_cluster_num (int): オリジナルのクラスタ数
38         title (str): プロットのタイトル
39         cluster_var (str): クラスタリング結果の変数名
40         exclude_clusters (list, optional): 除外するクラスタのリスト
41         rgb_var (str, optional): RGB 画像の変数名
42         figsize (tuple, optional): 図のサイズ
43         dpi (int, optional): 図の解像度
```

¹⁸とりあえず可視化できて論文に書ければ良いわけですから。

```

44     alpha (float, optional): クラスタリング結果の透明度
45     custom_colors (dict, optional): カスタムカラーマップの色指定
46     plot_cluster (bool, optional): クラスタリング結果をプロットするかどうか
47     plot_rgb (bool, optional): RGB 画像をプロットするかどうか
48     plot_grid (bool, optional): グリッド線をプロットするかどうか
49     grid_spacing (int, optional): グリッド線の間隔 (度)
50     grid_color (str, optional): グリッド線の色
51     grid_style (str, optional): グリッド線のスタイル
52     grid_linewidth (float, optional): グリッド線の幅
53     plot_coast (bool, optional): 海岸線をプロットするかどうか
54     coast_color (str, optional): 海岸線の色
55     save_path (str, optional): 図を保存するパス
56     subplot_index (tuple, optional): サブプロットのインデックス, (1, 1, 1)の時、
57                                     すなわち 1 つの図の可視化以外なら保存などはでき
58                                     ない
59
60     Returns:
61     tuple: (fig, ax) - 作成した図と axes オブジェクト
62     """
63     if custom_colors is None:
64         custom_colors = {
65             0: "#1f77b4", # 青
66             1: "#ff7f0e", # オレンジ
67             2: "#2ca02c", # 緑
68             3: "#d62728", # 赤
69             4: "#9467bd", # 紫
70             5: "#ffff00", # 黄色
71             6: "#7f7f7f", # グレー
72             7: "#bcbd22", # 黄緑
73             8: "#e377c2", # ピンク
74             9: "#ff9896", # 薄赤
75             10: "#98df8a", # 薄緑
76             11: "#ffbb78", # 薄オレンジ
77             12: "#aec7e8", # 薄青
78             13: "#c5b0d5", # 薄紫
79             14: "#c49c94", # 薄茶色
80             15: "#17becf", # 水色
81             16: "#f7b6d2", # 薄ピンク
82             17: "#c7c7c7", # 薄グレー
83             18: "#dbdb8d", # 薄黄緑
84             19: "#9edae5", # 薄水色
85             20: "#393b79", # 濃青
86             21: "#b35806", # 濃オレンジ
87             22: "#006d2c", # 濃緑
88             23: "#a50f15", # 濃赤
89             24: "#54278f", # 濃紫

```

```

89         25: "#843c39", # 濃茶色
90         26: "#c51b8a", # 濃ピンク
91         27: "#525252", # 濃グレー
92         28: "#b2df8a", # 黄緑 2
93         29: "#a6cee3", # 水色 2
94     }
95     # クラスタの再マッピング
96     if exclude_clusters:
97         cluster_remapped = ds[cluster_var].copy()
98         cluster_remapped = xr.where(
99             ds[cluster_var].isin(exclude_clusters),
100            np.nan, # 除外するクラスタを NaN に設定
101            cluster_remapped,
102        )
103
104     # 残ったクラスタに新しいラベルを振る
105     new_label = 0
106     for old_label in range(int(ds[cluster_var].max())):
107         if old_label not in exclude_clusters:
108             cluster_remapped = xr.where(
109                 ds[cluster_var] == old_label,
110                 new_label,
111                 cluster_remapped,
112             )
113             new_label += 1
114         else:
115             cluster_remapped = ds[cluster_var]
116
117     # 有効なクラスタ数の計算
118     n_valid_clusters = original_cluster_num - len(exclude_clusters)
119
120     # カラーマップの作成
121     valid_colors = {c: custom_colors[c] for c in range(n_valid_clusters)}
122     custom_cmap = ListedColormap([valid_colors[i] for i in
123                                     range(len(valid_colors))])
124
125     # プロット
126     fig = plt.figure(figsize=figsize, dpi=dpi)
127     ax = fig.add_subplot(*subplot_index,
128                          projection=ccrs.PlateCarree(central_longitude=180))
129
130     # RGB 画像を表示 (存在する場合)
131     if plot_rgb and rgb_var in ds:
132         ax.pcolormesh(ds.lon, ds.lat, ds[rgb_var].values,
133                      transform=ccrs.PlateCarree())

```

```

132     # クラスタリング結果を表示
133     if plot_cluster:
134         im = cluster_remapped.plot(
135             ax=ax,
136             transform=ccrs.PlateCarree(),
137             alpha=alpha,
138             cmap=custom_cmap,
139             add_colorbar=False,
140             vmin=0,
141             vmax=n_valid_clusters,
142         )
143
144     # カラーバーの追加
145     cbar = plt.colorbar(im, label="クラスタ")
146     cbar.set_ticks(np.linspace(0.5, n_valid_clusters - 0.5,
147                             n_valid_clusters))
147     cbar.set_ticklabels(range(n_valid_clusters))
148
149     # 地図の装飾
150     if plot_coast:
151         ax.coastlines(color=coast_color, linewidth=coast_linewidth,
152                     resolution="10m")
153     if plot_grid:
154         gl = ax.gridlines(
155             draw_labels=True,
156             linestyle=grid_style,
157             linewidth=grid_linewidth,
158             color=grid_color,
159         )
160         gl.top_labels = False
161         gl.right_labels = False
162         gl.xlocator = plt.FixedLocator(np.arange(-180, 181, grid_spacing))
163         gl.ylocator = plt.FixedLocator(np.arange(-90, 91, grid_spacing))
164         gl.xlabel_style = {"size": 10}
165         gl.ylabel_style = {"size": 10}
166     if title:
167         plt.title(title)
168
169     if subplot_index != (1, 1, 1):
170         warnings.warn("Multi plot mode, save_path is ignored!")
171     else:
172         if save_path:
173             plt.savefig(save_path)
174         else:
175             plt.show()

```

```
176     return fig, ax
```

クラスタリングの図 (Figure 29) から、クラスター番号 1, 3, 5, 7 は海面か地表面であると判断しました。そこで、`exclude_clusters` を以下のように定義し、関数を可視化する `visualize_clustering` を実行します。

```
1 exclude_clusters = [1, 3, 5, 7] Python  
2 visualize_clustering(ds_analysis, exclude_clusters=exclude_clusters, figsize=(12,  
    8), dpi=100)
```

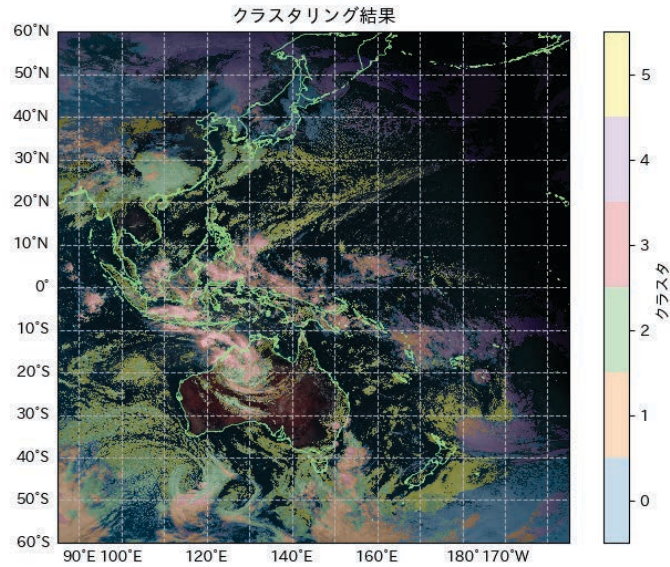


Figure 30. クラスタリング結果の可視化

その結果、Figure 30 のような画像が得られます。目標通り、海や地面が見えていところはクラスタリングの結果から除外されていますね。

4.3.5 領域を指定してクラスタリングの結果を見てみよう

前章での可視化の方法を応用し、領域を指定してクラスタリングの結果を見てみましょう。領域を指定するには、`sel` メソッドを用いるのでした。

例えば、日本の周辺のみを可視化したいときは、以下のように実行しましょう。

```
1 lats = slice(50, 18)
2 lons = slice(126, 150)
3 ds_target = ds_analysis.sel(lat=lats, lon=lons)
4 visualize_clustering(ds_target, title="日本周辺の雲", exclude_clusters=[1, 3, 5, 7], figsize=(12, 8), dpi=100)
```

Python

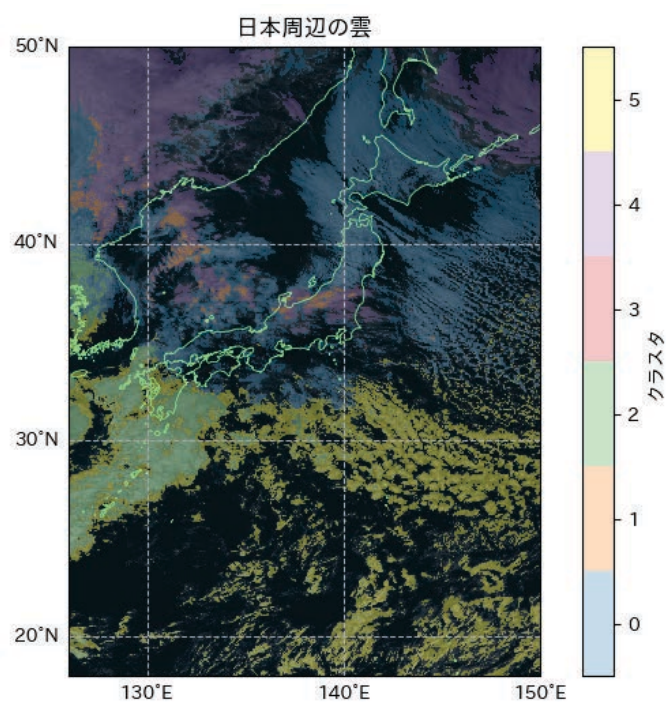


Figure 31. 日本周辺の雲のクラスタリング結果

日本では冬型の気圧配置によって、西側から日本海を通過して東側に向かう寒気の流れが強く、その寒気の流れに沿って筋状に雲が流れていることがわかります（クラスター0）。

トンガの噴火なら以下のようにします。

```
1 lats = slice(-15, -28)
2 lons = slice(178, 190)
3 ds_target = ds_analysis.sel(lat=lats, lon=lons)
4 visualize_clustering(ds_target, title="トンガ噴火", exclude_clusters=[1, 3, 5, 7], figsize=(12, 8), dpi=100)
```

Python

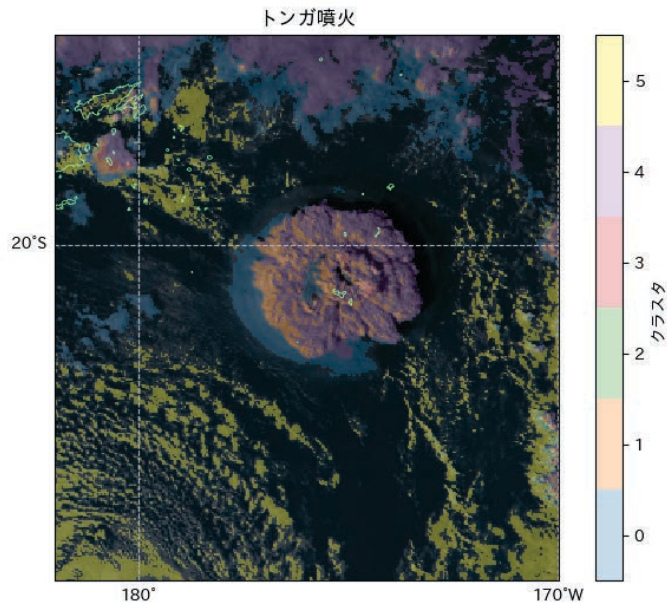


Figure 32. トンガ噴火のクラスタリング結果

噴火の中心から同心円状に噴煙が広がっていることがわかります。噴火中心から、クラスター6、クラスター1、クラスター0の順に噴煙が広がっていることがわかります。

4.4 学習済みモデルを他のデータの推論に利用しよう

ここまでは、モデルの学習について学びました。これによって、クラスターの中心が求まりました。クラスターの中心を表示してみましょう。

```

1 kmeans.cluster_centers_
2 """
3 array([[ 0.23891513,  0.28289476,  0.2624784 ,  0.27328315,  0.37611988,
4          0.37685713, -0.3973798 , -0.39910913, -0.39372048, -0.44160601,
5          -0.1128546 , -0.58330894, -0.52220404, -0.45597965, -0.41507277,
6          -0.5264737 ],
7        [-0.03717491, -0.46235052, -0.41034526,  0.37735146,  1.5855058 ,
8          1.1409559 ,  1.9691147 ,  1.9153811 ,  1.7758998 ,  1.5668467 ,
9          2.2223542 ,  0.98329026,  0.9582461 ,  0.9936646 ,  1.882145 ,
10         2.0109212 ],
11        [ 1.5200335 ,  1.5286137 ,  1.5337228 ,  1.4737186 ,  0.73846143,
12          1.2053272 , -1.4125123 , -1.4279001 , -1.4462693 , -1.3587466 ,
13         ...
14 shape=(10, 16)のデータ
15 """

```

このように、クラスターの中心は、16次元のベクトルで表され、クラスター数分（ここでは $K = 10$ ）のベクトルが用意されます。これら10個のクラスターとデータの距離によって、データがどのクラスターに属するのかが判定されます。

ここからは、学習済みモデルを用いて、新しいデータのクラスタリングを行ってみましょう。もう学習する必要はないので、fitメソッドは使用しません。その代わりに、predictメソッドを用いて、新しいデータのクラスタリングを行います。dsの、学習に使っていない時刻、2022年1月15日03時（UTC）のデータを用いて、予測を行ってみましょう。¹⁹ データの選択から、予測用のデータに変換するところまでを一度にやってしまいましょう。基本的には、学習で行ったプロセスと同じことをすれば良いだけです。

```
1 ds_test = ds.sel(time="2022-01-15 03:00:00")
2
3 # データセットを dask array に変換し、lat, lon をスタック
4 stacked_data = da.vstack(
5     [ds_test[var].stack(pixels=("lat", "lon")).data.astype("float32") for var in
6     ds_test.data_vars]
7 )
8 # (9000000, 16)の形に転置
9 X_test = stacked_data.T
10
11 # チャンクを適切に設定
12 # 1 軸目 (サンプル) は 100000 ごと、2 軸目 (特徴量) は全体を 1 チャンクに
13 X_test = X_test.rechunk((100000, -1))
```

ここで、データの前処理が必要になります。モデルの学習時と同じ処理、すなわち標準化を適用することで、データのスケールを揃えます。このとき、標準化に使用する平均と標準偏差は、学習時に用いたものをそのまま用います。

以前定義した scaler が、学習時の平均と標準偏差をそのまま保存していますから、この scaler をそのまま流用します。

```
1 X_test_scaled = scaler.transform(X_test)
```

これで、データの前処理が完了しました。これらのデータを用いて、クラスタリングの結果を予測します。

```
1 y_pred = kmeans.predict(X_test_scaled)
```

学習と異なり、推論の処理はあっという間に終わってしまうと思います。それでは、以下を実行して、予測結果を ds_test に追加し、可視化までやってしまいましょう。

```
1 ds_test["cluster"] = (
2     ("lat", "lon"),
3     y_pred.reshape((ds_test.lat.shape[0], ds_test.lon.shape[0])),
4 )
5 ds_test["cluster"].plot()
```

¹⁹このテキストでは、5時の情報で学習して3時を予測するという行っています。則ち、未来の情報を用いて過去を予測していることとなります。これは現実ではあり得ないことなのですが、あくまで実習という事でご理解ください。

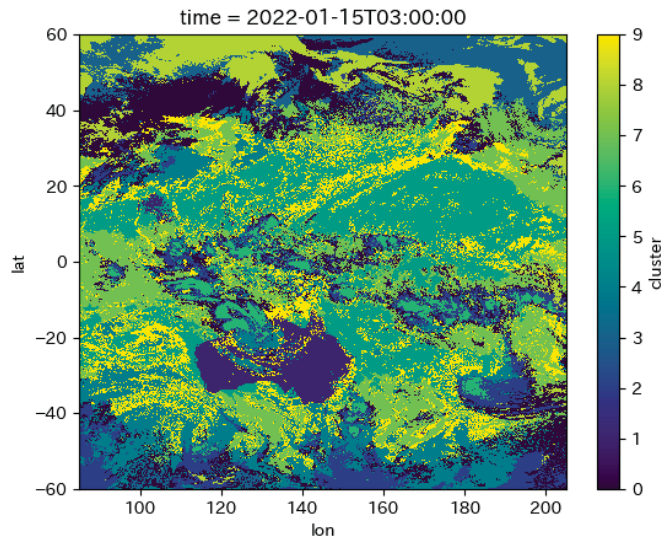


Figure 33. 2022年1月15日03時のクラスタリング結果

その結果、Figure 33 この画像は、学習に用いたデータのクラスタリング結果と同じです。更に綺麗に書いてみましょう。このとき、除外するクラスターは学習時と同じでないといけないので注意しましょう。²⁰

```

1 rgb_test = make_rgb(ds_test)
2 rgb_test_gamma = gamma_correction(rgb_test, 1.4)
3 ds_test["rgb"] = (("lat", "lon", "rgb_channel"), rgb_test_gamma)
4 visualize_clustering(ds_test, exclude_clusters=exclude_clusters, figsize=(12, 8),
  dpi=100)

```

Python

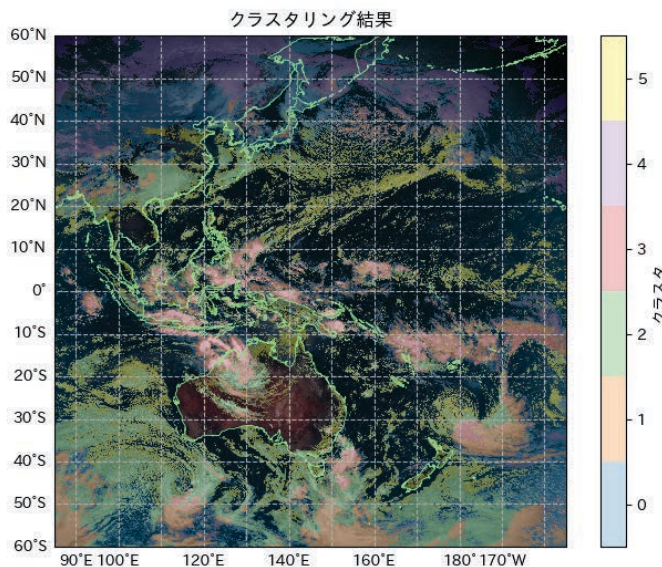


Figure 34. 2022年1月15日03時のクラスタリング結果（予測）

上のように表示されれば成功です！

²⁰学習時に雲以外として除外したクラスターは、予測時にも雲以外として除外しないとおかしいですね。

クイズ

2022年1月15日03時の日本の周辺を可視化をしてみよう

sel メソッドを使って領域を絞りましょう。みなさんの大学の周辺にはどんな雲がありますか？

2022年1月15日03時のトンガ噴火の可視化をしてみよう

噴火時刻が日本時刻の1時頃とされていますが、この直前の火山はどうみえますか？

4.5 モデルの保存と読み込み

モデルを学習した後、そのモデルを保存しておくことで、後で再利用することができます。Google Colab を

```
1 import pickle
2
3 # モデルの保存
4 with open("kmeans.pkl", "wb") as f: # Google Drive
5     pickle.dump(kmeans, f)
6
7 # モデルのロード
8 with open("kmeans.pkl", "rb") as f:
9     kmeans = pickle.load(f)
```

5.

考察を深めよう

この章のポイント


- クラスタリングの結果と各バンドで取得されたデータの比較
- 地上からの雲画像データとの比較
- 良いクラスターとは？ どうやったら精度が良くなるの？

5.1 解析手法いろいろ

5.1.1 クラスタリング結果と各バンド画像の比較

学習に使ったバンドとクラスタリングの結果を比較してみましょう。ひまわりには16のバンドがあることは、第1章で紹介をしましたが、それぞれのバンドがクラスタリング結果にどのような影響を及ぼしているのかを考えてみましょう。以下のコードを実行して、クラスタリング結果と16枚の画像を一度に表示してみます。²¹

```
1 lats = slice(-15, -28)
2 lons = slice(178, 190)
3 row = 3
4 col = 6
5
6
7 total_subplots = row * col
8
9 NUM_TIR = 10
10 NUM_SIR = 2
11 NUM_VIS = 3
12 EXT = 1
13
14 ds_target = ds_analysis.sel(lat=lats, lon=lons)
15 fig, ax = visualize_clustering(
16     ds_target,
17     original_cluster_num=K,
18     title="トンガ噴火",
19     exclude_clusters=exclude_clusters,
```

 Python

²¹このコードは matplotlib の subplot を用いて、一枚の図面に複数枚の図を書くことを行っています。

```

20     figsize=(16, 6),
21     dpi=100,
22     subplot_index=(row, col, 1),
23 )
24
25 subplot_num = 2
26
27 ax = fig.add_subplot(row, col, subplot_num,
28     projection=ccrs.PlateCarree(central_longitude=180))
29 ds_target["ext.01"].sel(lon=lons, lat=lats).plot(
30     ax=ax,
31     transform=ccrs.PlateCarree(),
32     cmap="gist_stern",
33 )
34 ax.set_title("ext.01")
35 subplot_num += 1
36
37 for band in range(1, NUM_VIS + 1):
38     ax = fig.add_subplot(row, col, subplot_num,
39         projection=ccrs.PlateCarree(central_longitude=180))
40     ds_target["vis." + str(band).zfill(2)].sel(lon=lons, lat=lats).plot(
41         ax=ax,
42         transform=ccrs.PlateCarree(),
43         cmap="gist_stern",
44     )
45     ax.set_title("vis." + str(band).zfill(2))
46     subplot_num += 1
47
48 for band in range(1, NUM_TIR + 1):
49     ax = fig.add_subplot(row, col, subplot_num,
50         projection=ccrs.PlateCarree(central_longitude=180))
51     ds_target["tir." + str(band).zfill(2)].sel(lon=lons, lat=lats).plot(
52         ax=ax,
53         transform=ccrs.PlateCarree(),
54         cmap="gist_stern",
55         # vmin=190, # カラーマップの最小値を設定
56         # vmax=300 # カラーマップの最大値を設定
57     )
58     ax.set_title("tir." + str(band).zfill(2))
59     subplot_num += 1
60
61 for band in range(1, NUM_SIR + 1):
62     ax = fig.add_subplot(row, col, subplot_num,
63         projection=ccrs.PlateCarree(central_longitude=180))
64     ds_target["sir." + str(band).zfill(2)].sel(lon=lons, lat=lats).plot(

```

```

62     ax=ax,
63     transform=ccrs.PlateCarree(),
64     cmap="gist_stern",
65     # vmin=190, # カラーマップの最小値を設定
66     # vmax=300 # カラーマップの最大値を設定
67 )
68 ax.set_title("sir." + str(band).zfill(2))
69 subplot_num += 1
70 plt.show()
71 plt.close()

```

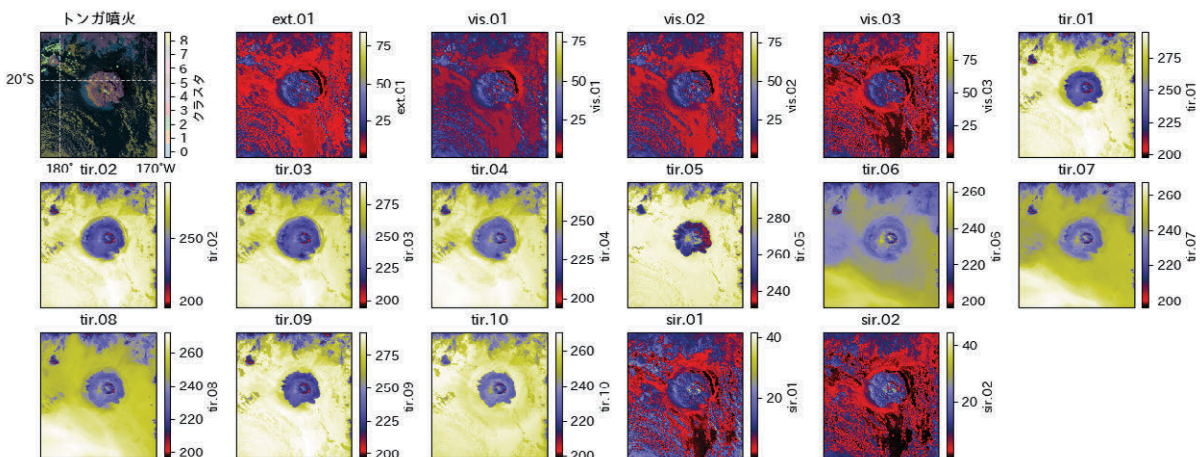


Figure 35. クラスタリング結果と各バンド画像の比較

このコードを実行すると、Figure 35 のような画像が得られます。それぞれのバンドの主な観測項目や可視化の結果を対応させながら検討してみると考察深まるかもしれません。また、このコードの可視化領域を変更すれば、更に様々な考察を行うことができるでしょう。

5.1.2 そのクラスター、本当に良いクラスターなの？

クラスタリングがうまくできたかどうかの確認を行う手法には、エルボー法やシルエット分析などがあります。今回は後者を用いて、クラスタリングがうまくできたかどうかの確認をしてみましょう²²。

シルエット法は、あるクラスターに属するサンプルと、同一クラスター内のサンプルとの距離が近いほど（凝集しているほど）、最も近い他のクラスター内のサンプルとの距離が遠いほど（乖離しているほど）、そのクラスターは良いクラスターであると判断するという判断の基準を用いています。

C_I をあるサンプル x_i が属するクラスターとし（則ち $x_i \in C_I$ ）、クラスターの凝集してなさの度合い a_i を以下のようなスコアで表します。これは、 x_i 自身を除く、 C_I 内のすべての点との距離の平均です。

²²これは著者経験ですが、エルボー法は大抵うまくいかない気がします。また、シルエット分析を用い定量的にスコアを算出したとしても、結局最後は見た目での判断も大事だと思われます。

$$a_i = \frac{1}{|C_I| - 1} \sum_{x_j \in C_I, i \neq j} \|x_i - x_j\|^2 \quad (8)$$

次に、点 x_i と、最も隣接したクラスター C_J との平均非類似度 b_i を、 x_i から C_J 内のすべての点までの距離の平均として定義します。各データ $x_i \in C_I$ に対して、 b_i は以下のように定義されます。

$$b_i = \min_{C_J \neq C_I} \frac{1}{|C_J|} \sum_{x_j \in C_J} \|x_i - x_j\|^2 \quad (9)$$

この時、 x_i のシルエット係数 s_i は以下のように定義されます。

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \quad (10)$$

このシルエット係数 s_i は、-1 から 1 の間の値を取り、1 に近いほど良いクラスターであると判断されます。各クラスターのシルエット係数の平均値を計算することで、クラスタの良さを評価することができます。さらに、各サンプルについてスコアを出した後、クラスターごとにまとめ、係数が大きいものからソートしたデータを棒グラフで可視化した結果シルエットプロットといい、Figure 36 のように可視化します。これは以下のコードを使って計算、描画されます。

```

1 from sklearn.metrics import silhouette_samples
2 import matplotlib.pyplot as plt
3 import matplotlib.cm as cm
4 import numpy as np
5
6 # サンプリングしたデータでシルエット分析を行う
7 sample_size = 1000 # サンプルサイズ
8 random_indices = np.random.choice(X_scaled.shape[0], sample_size, replace=False)
9
10 # サンプリングしたデータを取得
11 X_sample = X_scaled.compute()[random_indices]
12 cluster_labels = kmeans.labels_.compute()[random_indices]
13
14 # シルエット係数を計算
15 silhouette_vals = silhouette_samples(X_sample, cluster_labels)
16
17 # クラスタ数
18 n_clusters = kmeans.n_clusters
19
20 # シルエット図の作成
21 from sklearn.metrics import silhouette_samples
22 import matplotlib.pyplot as plt
23 import matplotlib.cm as cm
24 import numpy as np
25
26 # サンプリングしたデータでシルエット分析を行う

```



```

27 sample_size = 10000 # サンプルサイズ
28 random_indices = np.random.choice(X_scaled.shape[0], sample_size, replace=False)
29
30 # サンプリングしたデータを取得
31 X_sample = X_scaled.compute()[random_indices]
32 cluster_labels = kmeans.labels_.compute()[random_indices]
33
34 # シルエット係数を計算
35 silhouette_vals = silhouette_samples(X_sample, cluster_labels)
36
37 # クラスタ数
38 n_clusters = kmeans.n_clusters
39
40 # シルエット図の作成
41 plt.figure(figsize=(12, 8))
42 y_lower, y_upper = 0, 0
43
44 # 各クラスタのシルエット係数をプロット
45 for i in range(n_clusters):
46     # i 番目のクラスタのシルエット係数を取得
47     ith_cluster_silhouette_vals = silhouette_vals[cluster_labels == i]
48     ith_cluster_silhouette_vals.sort()
49
50     size_cluster_i = ith_cluster_silhouette_vals.shape[0]
51     y_upper = y_lower + size_cluster_i
52
53     # カラーマップの色を取得
54     color = cm.tab10(float(i) / n_clusters)
55
56     # シルエット係数をプロット
57     plt.fill_betweenx(
58         np.arange(y_lower, y_upper),
59         0,
60         ith_cluster_silhouette_vals,
61         facecolor=color,
62         edgecolor=color,
63         alpha=0.7,
64     )
65
66     # クラスタラベルを追加
67     plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
68
69     # 次のクラスタの y_lower を更新
70     y_lower = y_upper + 0
71
72 # 平均シルエット係数の線を追加

```

```

73 avg_silhouette = np.mean(silhouette_vals)
74 plt.axvline(x=avg_silhouette, color="red", linestyle="--")
75 plt.text(
76     avg_silhouette + 0.02,
77     plt.ylim()[0] + 0.05 * (plt.ylim()[1] - plt.ylim()[0]),
78     f"平均シルエット係数: {avg_silhouette:.3f}",
79     color="red",
80 )
81
82 # グラフの設定
83 plt.title("シルエットプロット", fontsize=16)
84 plt.xlabel("シルエット係数", fontsize=14)
85 plt.ylabel("サンプル番号", fontsize=14)
86 plt.tight_layout()
87 plt.show()

```

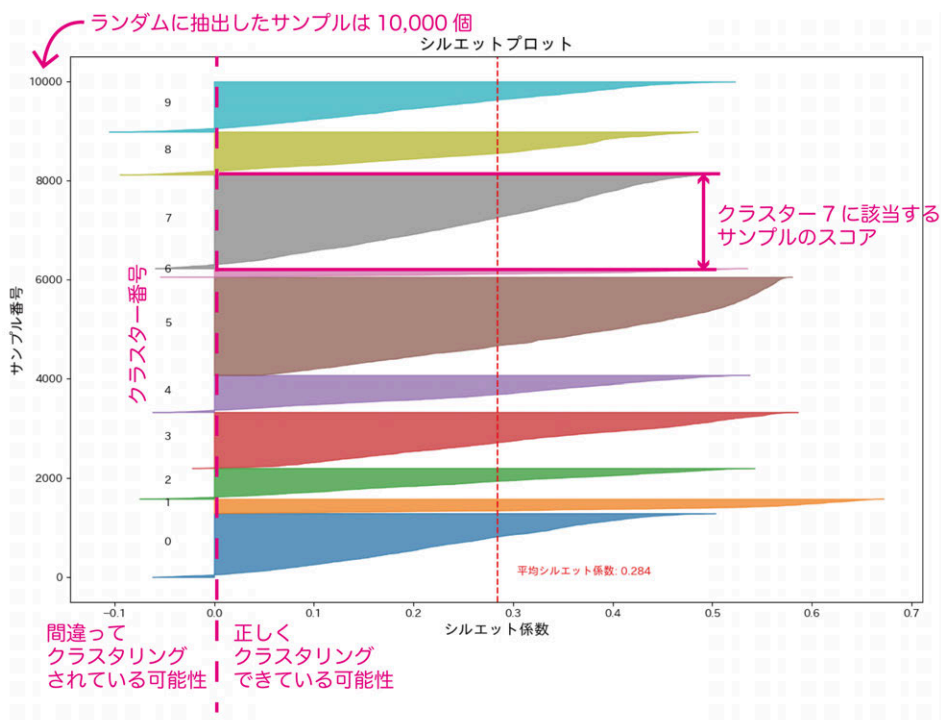


Figure 36. シルエットプロット

シルエットプロットは、横軸にシルエット係数、縦軸にサンプル番号をとったプロットです。この各クラスターの厚みから、分類されたクラスターの偏りを知ることができます。また、シルエット係数がマイナスのサンプルがどの程度あるのか、各クラスターのシルエット係数の平均はどの程度かなどを確認することができます。

この計算では、 $K = 10$ で計算していますが、これを変化させることで、シルエットプロットがどうなるのか変化を見ながら、最適な K を探ることができます（できるかもしれません）。

5.1.3 地上からの観測データと比較してみよう

A-SKYのデータは、CEReSの入江研究室が主導する、国際地上リモートセンシング観測網です。この中には、千葉大学で撮影された雲のデータ(可視光)も入っています。過去からずっとアーカイブされていますから、これを用いて、クラスタリングの結果と比較してみましょう。

A-SKY (<https://a-sky.irie-lab.jp/>) にアクセスし、右上のDATAから千葉のデータにアクセスしてみましょう。沢山の観測項目が見つかるはずです。その中から、Skyviewを見つけ、過去のデータを探しましょう。



Figure 37. A-SKY

5.1.4 SNS から雲画像を探してみる

SNSなどを使って当時の雲写真を探してみるのも面白いかもしれません。著者も探してみましたが、あまりそれらしい画像は見つかりませんでした……。うーん。

5.1.5 他の現象でも見てみよう

他の日付のデータセットを以下に準備しました。時間のある方は試してみて、クラスタリングの結果がどう変わるのか、学習済みモデルが正しく評価できているかなどを試してみましょう。分類のパターンを見ることで、だんだんと知見が得られるかもしれません。

追加データのダウンロード先は以下です。 (https://drive.google.com/drive/folders/1yJv2OXPnsdfDRM19Z0pPiwnfCC_Bf1G-?usp=sharing)

5.2 どうやったら精度が良くなるの？

いままでの例では、クラスタリングをし、それを綺麗に可視化するところまでを取り組みました。一方、まだ精度が十分に出ないところや、科学的な解釈が難しい箇所があるかもしれません。例えば、海の領域と雲の領域が上手く分離されていなかったり、同じ性質かもしれない雲が違うクラスターに分類されていたり、我々の解釈では上手く納得いかない箇所があります。これをなるべく解釈可能な分類とするために、以下のような工夫をしてみましょう。

5.2.1 学習データの追加

今は1つのタイムステップのデータで学習していますが、これが例えば複数の時刻にまたがるデータであったりすると、更に情報が増えて、学習の結果がより良くなるかも知れません。


5.2.2 クラスタ数の調整

クラスタ数によって、クラスタリングの精度は変わります。クラスタ数を変えて、精度を確認してみましょう。シルエットプロットなどを確認しながら検証すると良いですね。この時、クラスタ数が多すぎると、オーバークラスタリングが起きてしまい、不必要なクラスターに過剰に分類されてしまいますので、気を付けましょう。

5.2.3 説明変数の選択


現在は、全てのバンドのデータをそのまま学習器に入れています。しかし、これらのバンドの中でも、雲の存在を示すバンドとそうでないバンドがあります。そこで、雲の存在を示すバンドのみを学習器に入れることで、精度の向上が期待できます。xr.open_mfdataset で読み込んでいるデータセットは、16のバンドのデータを含んでいます。この中から、不要なバンドを削除してみましょう。例えば以下の例では、影の影響が出てしまう可視バンドを削除する例です。

```
1 ds = ds.drop(["ext.01", "vis.01", "vis.02", "vis.03"])
```

 Python

また、変数に重み付けをすることによって、変数ごとの重要さを調整し学習することができます。以下では、可視バンドを補助的な情報として扱うために、重み付けをしています。

```
1 WEIGHT = 0.8 # 可視変数の重み
2 ds[["ext.01", "vis.01", "vis.02", "vis.03", "sir.01", "sir.02"]] = ds[["ext.01",
  "vis.01", "vis.02", "vis.03"]] * WEIGHT
```

 Python

5.2.4 空間情報や時間情報の入力

今回の実習では、単一のグリッドの16バンドの情報を参照し学習を行いました。しかし、これでは雲の時間変化や周辺情報を全く学習できていません。これらを学習するために、16次元のベクトルに加えて、空間情報や時間情報を入力することで、より精度の高い学習が期待できます。

5.2.5 モデルの変更

モデルには、それぞれ特徴があります。得意なこと、不得意なこと、様々な数理的背景に目を向けて、モデルを選択しなくてはなりません。今回は、簡単のため、K-means を使いましたが、このモデルが暗に前提とされているのは、各クラスターが超球面に分布していること、クラスターのサンプルサイズがだいたい同じということです。これは、厳密な仮定というわけではないのですが、計算の性質上このようなデータの方が学習が上手くいきやすいという事が考えられるのです。世の中には様々なモデルがありますが、dask_ml に実装されているものも沢山ありますから、検討してみると良いでしょう。

おわりに

衛星データと機械学習について、入門コースとして、このような教材を作成しましたが、いかがでしたでしょうか。すんなりと受け入れられる人と、そうでない人がいると思います。私は後者なので、博士課程のときに、なかなか研究が思うように進まず（今も大変ですけど）苦労した記憶があります。

この、機械学習の分野でとても重要なのは、とにかく手を動かして実装してみることだと思います。やっていく内に、数理的背景を学習し、はじめて間違いに気付いたり、はじめて実装のコツを体験したりするのです。それからまた軌道を修正して、というのを繰り返す、こんなアルゴリズムが大事なんだと思います。これは、機械学習のモチベーションと同じところがあるのかなと考えています。機械学習のアルゴリズムが開発された背景にも、「とりあえず～を仮定してやってみる」「とりあえず計算して誤差があるからそれを修正する」などなど、学習している内に、とりあえずやってみることが大事なのだ気付くと思います。動かない限りは、何も進まないのですよね。停滞が一番の悪だから、とりあえず動かしてみる、というのは、他人から言われて学ぶよりも、自分でモデルの数理的背景を学習した方が、腑に落ちる気すらします。

昨今、自然言語処理が大きく発展し、私たちと会話できるAI（なのかな）が、私たちの生活に一気に浸透しました。聞けば色んなことを教えてくれるし、色んなことをやってくれます。ドラえもんの世界とまではいかないまでも、どんどんSFの世界に近づいているなどというワクワク感と不安感があります²³。これは、できることが増えていく反面、理解が私たちの手元から離れていってしまうのではないかという不安です。軽く質問して返ってきたテキストを、判断する術がなくなるのではないか、本当は自分の個性を出さなくてはならないところで、楽な道を選んでしまわないか、そんなことを思っています。今回の課題もAIに聞けば全部答えてくれるでしょう。しかも筆者よりも良い答えをくれるかもしれません。

この時に、私たちの深い理解はAIを使う助けになると思っていますが、一方で、最後にプログラムの善し悪しを判断するのは、数理的背景を判断するのは、そしてそれらに責任を持つのは私たちです。こんな時代だからこそ、泥臭くやっていくことは、他人と差別化する上でも大事なのかなと思っています。これは一種の個性であるとも思っていて、その人なりの判断、その人なりの考えの、上振れ、下振れがあるからこそ、面白い研究ができるのかなと考えています。もっとも、それはたぶん効率が悪いし、研究に個性はいらないのだけど、なんだかそう言うのを実は期待しているのかもしれない。

ひまわり9号は2029年までの運用予定で、まだまだ頑張ってくれる予定です。そして、2029年からはひまわり10号が観測を開始する予定となっています。冒頭に説明したとおり、私の研究の歴史は、ひまわり8号とともにスタートしました。ですので、衛星が代替わりする

²³筆者はPSYCHO-PASSとかそういうSFがすきなのです。

ことは感慨深く、自分²⁴も歳をとっていくものだと実感しています。2029年までも新たな地球に関する知見が得られることと思いますし、新しいセンサーの搭載されたひまわり10号からは、更に壮大な情報を手に入れることができることになるでしょう。これらのデータを使ってどんなことができるのか、どんなことがわかるのか、期待感が湧いてきますし、我々もその流れに乗らなくてはなりません。「衛星データ×機械学習」のキックオフとして、この教材が皆さんの研究人生にちょっとでも彩りを添えられたら幸いです。

さて、この実習は多くの学生の皆さんとスタッフの皆さんに支えられています。本郷研究室の林くん、入江研究室の溝渕君、小槻・岡崎研究室の、毛束くん、齋藤さん、宮澤くん、この実習のテキストの確認、プログラムの確認、そして実習に対する的確なアドバイスを大変にありがとうございました。飛び入り参加でTAに加わってくれた小槻・岡崎研究室の白石君もありがとう！私自身もとっても勉強になりました。またCEReSのVLスタッフのみなさま、このようなイベントの企画と運営をありがとうございました。また、小槻・岡崎研究室の技術補佐員の曾我さんにおかれましては、表紙のデザインをかなり相談させていただきました。ちょっと配置を変えるだけでも全く印象が異なることに驚きました。ありがとうございました。

最後になりましたが、今回、この実習に参加いただきましたが皆さまに、改めまして御礼を申し上げますと共に、今後の分野の発展を願い、このテキストを書き終えたいと思います。ありがとうございました。

²⁴リプレイスはしないでほしいです。

宇宙から色々な雲を見つけよう！
～衛星画像×機械学習入門～

2025年 3月13日 第2版 発行

2025年 3月 3日 初版 発行

発行者 千葉大学環境リモートセンシング研究センター

著者 金子凌・小槻峻司

連絡先 rkaneko@chiba-u.jp

© Ryo Kaneko, Shunji Kotsuki, 2025