AIエンジニアリング講義 第3回

松尾研究室
MATSUO LAB THE UNIVERSITY OF TOKYO

2025/4/30

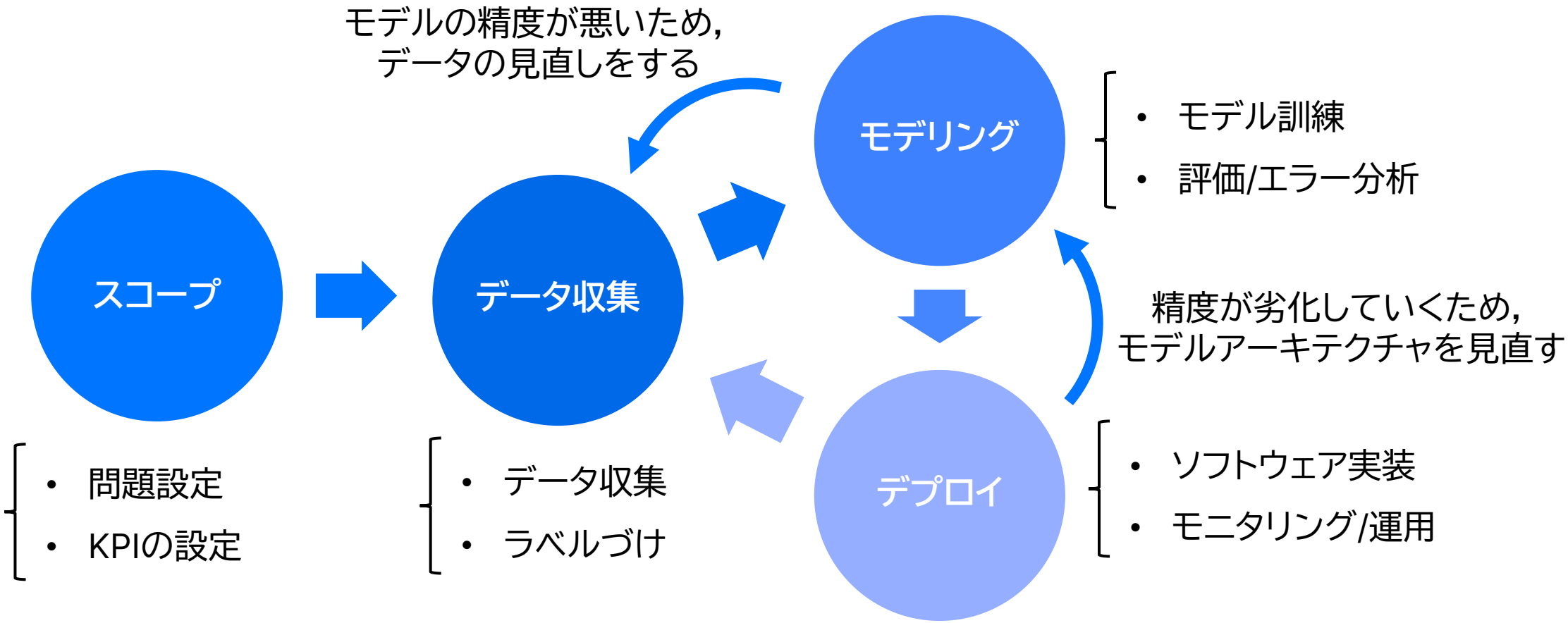
松尾・岩澤研特任助教 河野 慎

アジェンダ

- 事例を交えた開発サイクルの概要
- デプロイ
- モデル開発
- データ収集
- モデル改善

MLプロジェクトのライフサイクル

- 基本的には一定方向だが, 時には戻る必要あり



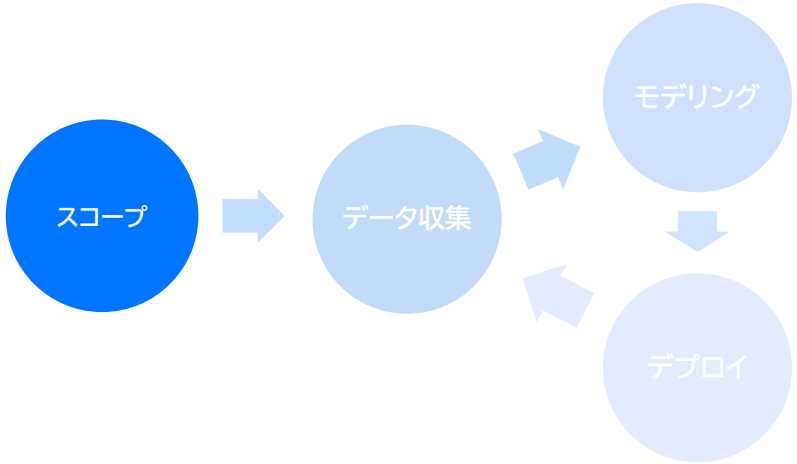
事例紹介: インフラ点検

- 車載カメラに映る道路表示の損傷を検出する[河野, 2017]

  - 点検は, 自治体の職員が別の用事があった時に目視で確認する程度
  - 自治体の管理区域全てを目視で確認するのは大変
  - 職員の高齢化が進んでおり, 年々人手不足問題

事例紹介: インフラ点検

- プロジェクトスコープを決める
  - 車載カメラ映像を用いて, 自動で損傷箇所を発見
  - 機械学習の問題としては, 物体検出問題として定義



x → y

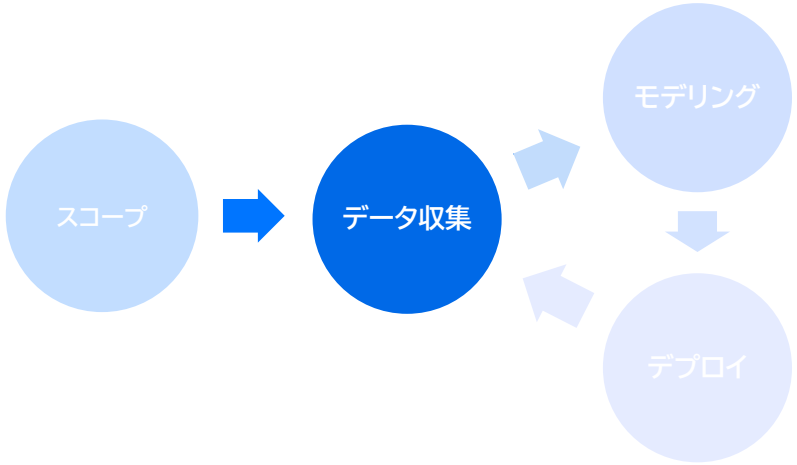
車載カメラ映像

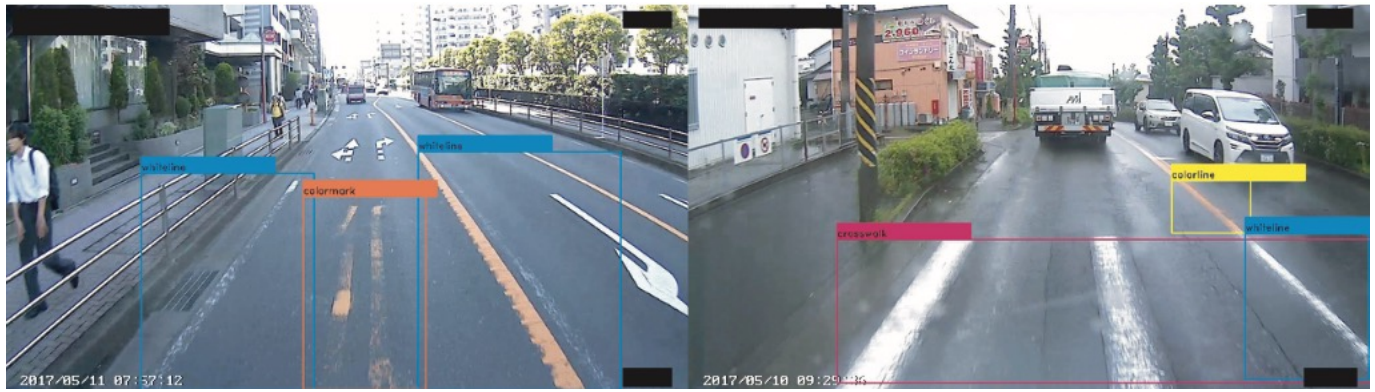
損傷箇所の
バウンディングボックス

- 通信費や計算機資源コストの観点から, エッジデバイスで実行
- 自車の走行速度や取り付けるカメラの画角等から1FPS以上の処理速度が必要

事例紹介: インフラ点検

- データ収集
  - 実際のドライブレコーダ映像10時間分(18FPS)
    - 25フレーム毎に1枚間引く
  - 対象とする道路標示は，5種類
  - アノテーションはバウンディングボックスで囲む

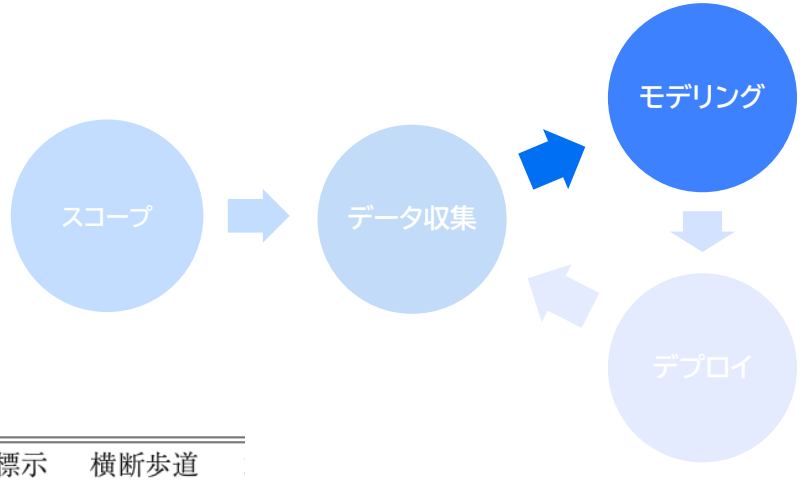




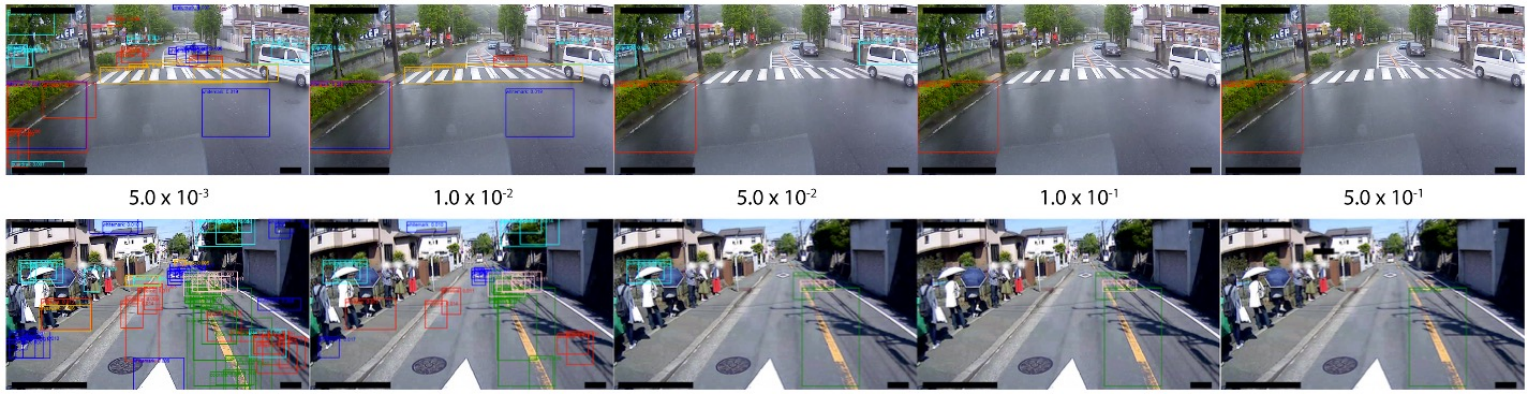
データセットの種類	画像数	白線	白色道路標示	黄色線	黄色道路標示	横断歩道
訓練用	1,807	963	726	249	37	209
テスト用	1,821	1,065	635	137	55	424

事例紹介: インフラ点検

- モデル開発
  - 当時, 物体検出器としてはSSD/YOLOv2の2強
    - 結果的に大きい対象を検出しやすいSSDを採用
  - データ拡張は, 色調変化と左右反転※を適用



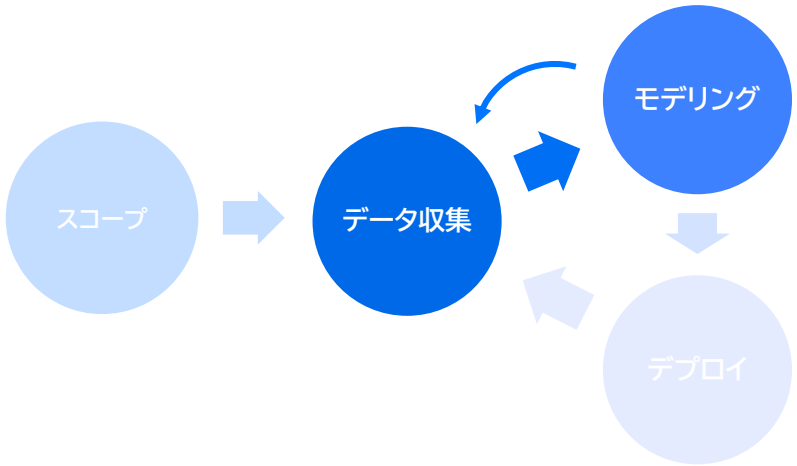
クラス閾値	mAP	白線	白色道路標示	黄色線	黄色道路標示	横断歩道
5.0×10^{-3}	48.80	49.12	51.71	33.77	51.31	63.04
1.0×10^{-2}	48.65	49.07	51.42	33.77	51.17	62.63
5.0×10^{-2}	46.95	48.57	49.74	32.33	50.50	57.16
1.0×10^{-1}	45.75	46.39	49.67	32.33	46.51	57.16
5.0×10^{-1}	39.13	41.02	40.31	29.02	34.87	50.68

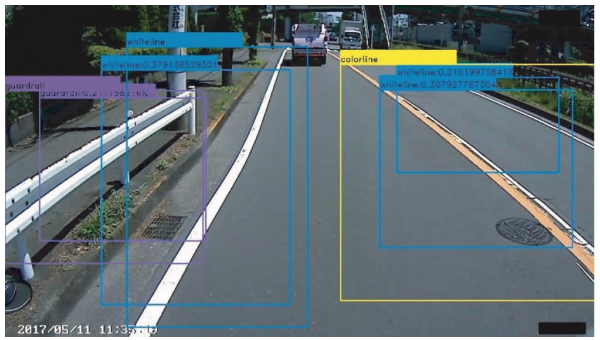


※今考えると, 日本の道路事情的に微妙だったかもしれない

事例紹介: インフラ点検

- モデル開発時にデータ収集(アノテーション)に戻る
  - ガードレールが白の区画線として検出されてしまう
  - 検出対象として, ガードレールを追加
  - もう一度, SSDの学習/評価を実施



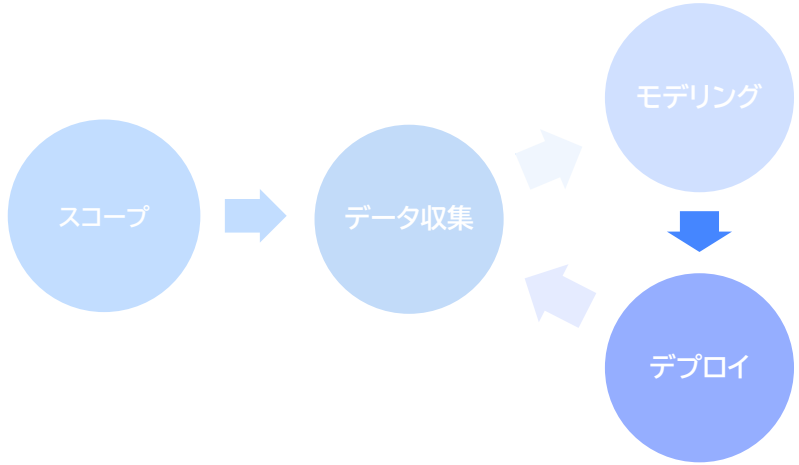


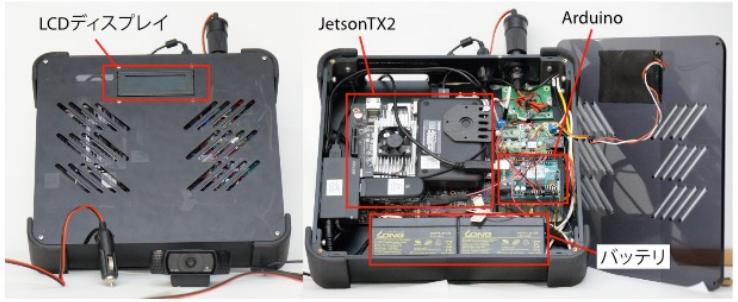
データセットの種類	画像数	白線	白色道路標示	黄色線	黄色道路標示	横断歩道	ガードレール
訓練用	1,807	963	726	249	37	209	709
テスト用	1,821	1,065	635	137	55	424	588

クラス閾値	mAP	白線	白色道路標示	黄色線	黄色道路標示	横断歩道	ガードレール
5.0×10^{-3}	48.80	49.12	51.71	33.77	51.31	63.04	43.85
1.0×10^{-2}	48.65	49.07	51.42	33.77	51.17	62.63	43.85
5.0×10^{-2}	46.95	48.57	49.74	32.33	50.50	57.16	43.42
1.0×10^{-1}	45.75	46.39	49.67	32.33	46.51	57.16	42.42
5.0×10^{-1}	39.13	41.02	40.31	29.02	34.87	50.68	38.88

事例紹介: インフラ点検

- デプロイ
  - JetsonでSSDが動作するようにシステムを設計・実装
    - ハードウェア・ソフトウェア
  - モニタリングシステムも同時に実装し，精度を目視で確認

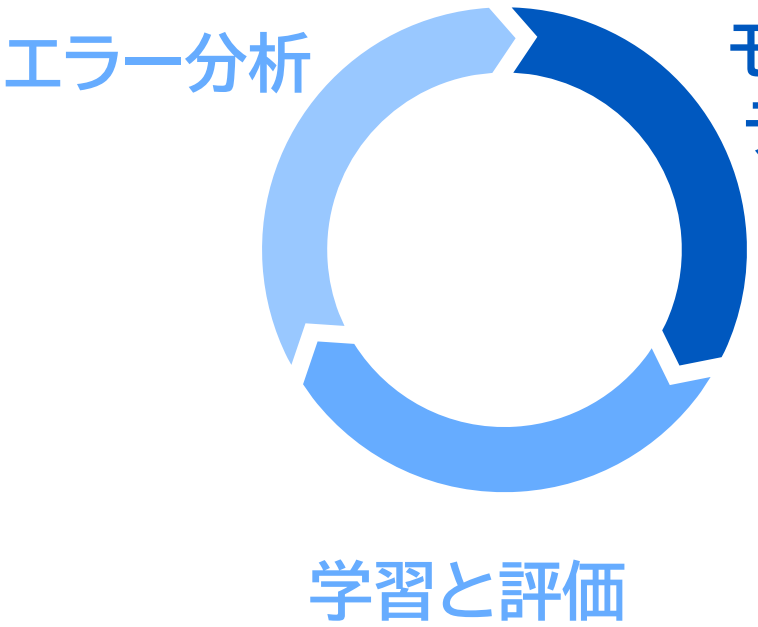


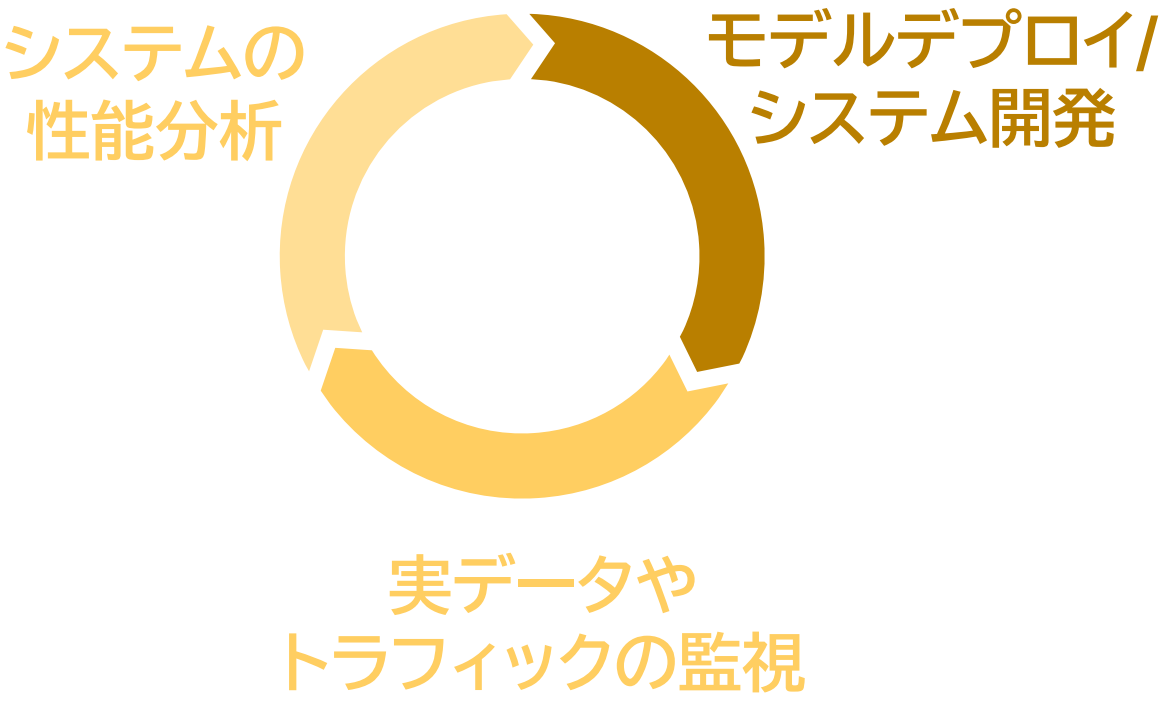


デプロイ

デプロイもPDCAサイクルが必要

- モデル開発も，1回目の学習評価で終わることはまずない
- →デプロイも同様に，1回のデプロイで終わることはない





ソフトウェア開発における課題

- 道路点検の場合
  - 推論はエッジで実行？サーバに動画を送信して実行？
    - バッチ処理？ストリーミング/リアルタイム処理？
    - スループット/レイテンシは？
      - 何ミリ秒以内に1フレーム処理をしないといけない？
  - 映像に映る個人情報はどうする？
  - 謎の条例への対応
    - ドライブレコーダのストレージの”半分”までしか保存してはいけない

これらの疑問/課題は, 大体のアプリケーションで当てはまる話

デプロイしたモデルの劣化

- システムを運用していくと，モデルの精度が下がっていってしまうことがある

モデルの学習に利用したデータ ≠ 運用時のデータ

データドリフト

xが変わってしまうこと

- 道路標示の損傷検出に使用していたカメラのメーカーを変更

コンセプトドリフト

$f: x \to y$が変わってしまうこと

- Deepseek-R1の登場により，NVIDIA株が急落

- **段階的な変化（流行語のようなもの）**
- **突然の変化（疫病や経済的事件）**

など様々な要因がありうる

モニタリングするべきものは？

ソフトウェア指標 リソースの使用量, 遅延, スループット

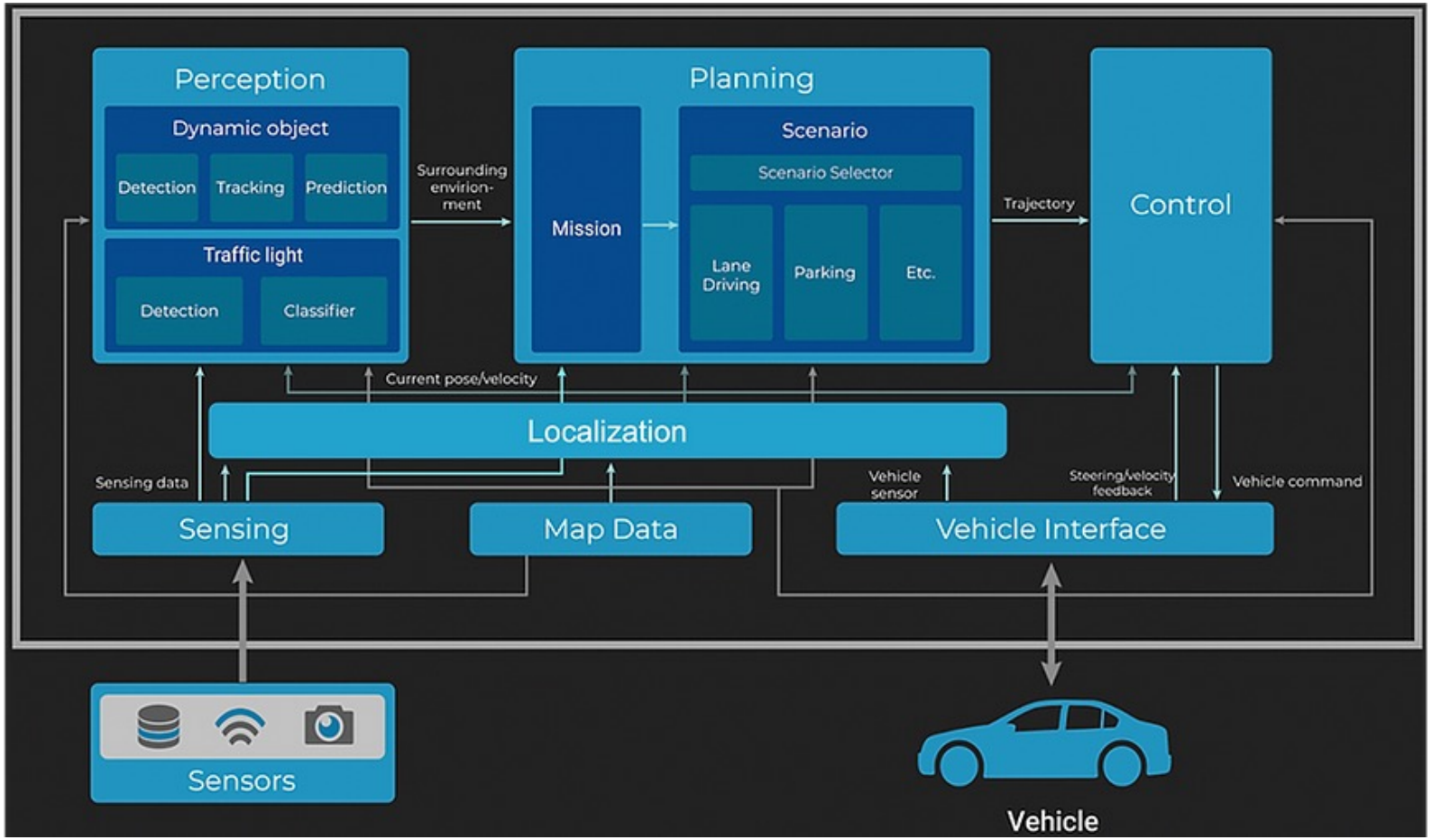
入力指標 入力長の平均, 入力量, 欠損値, 例外, 画像の明度など

出力指標 nullの回数, 再試行回数, 精度など

- ダッシュボード

MLモデルのパイプライン処理

- 自動運転システム(AD1.0※)



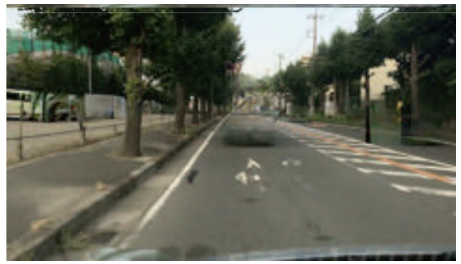
※最近は, 基盤モデルを用いたEnd-to-EndなAD2.0に移行が始まっている

MLモデルのパイプライン処理

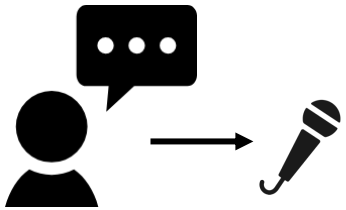
- 道路点検の例

匿名加工処理
モジュール

損傷箇所
検出モジュール

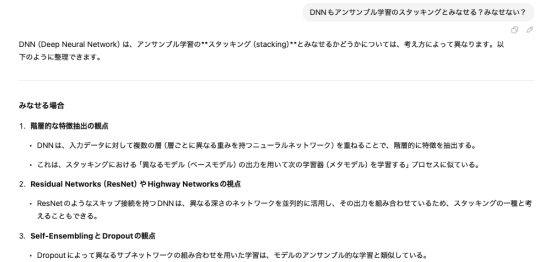


- チャットbot



Speech to Text
モデル/モジュール

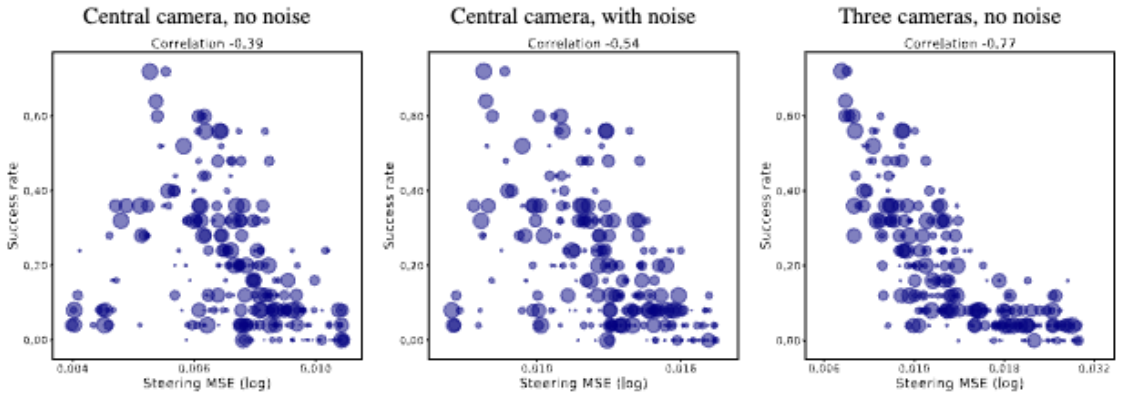
LLMなど
会話モジュール

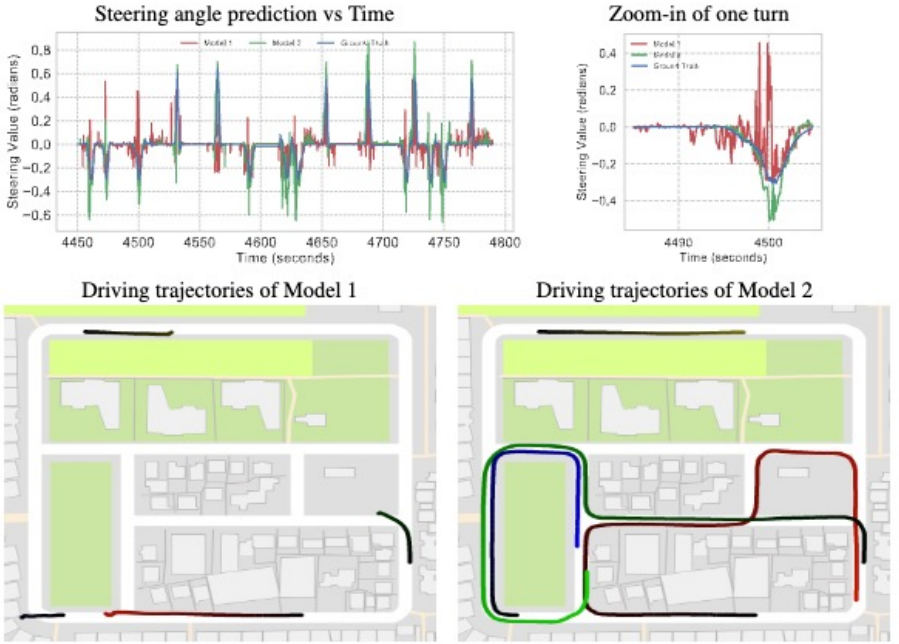


前のモジュールが変われば，データドリフトが起き，精度が変化する

実データで性能が出たとしてもシステムの性能としてはわからないケース

- 自動運転やロボットのプランニング
  - Open-loop vs Closed-loop 評価





- 異常検出
  - 1%の異常を検出することが可能なモデルができたとして…
  - 実運用していて，”その時”きちんと検出ができるのか？

モデル開発

AI/MLシステムをデプロイする

- AIシステム =

アルゴリズム
アーキテクチャ (+ソースコード)

PyTorch

TensorFlow

+

データ

huggingface_hub

論文などで研究・開発される技術

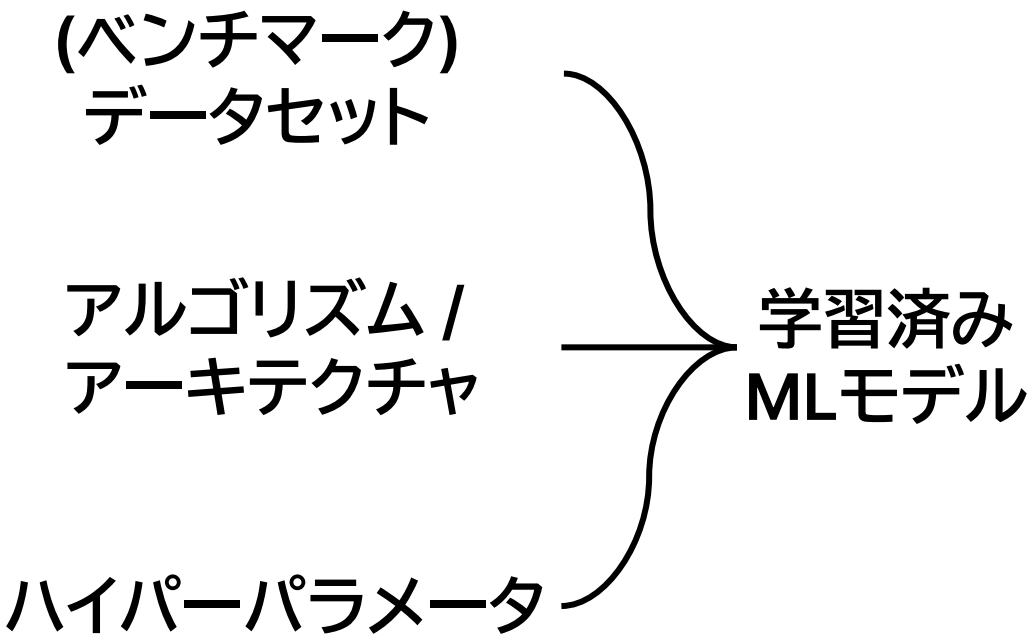
実応用を目指したシステム開発

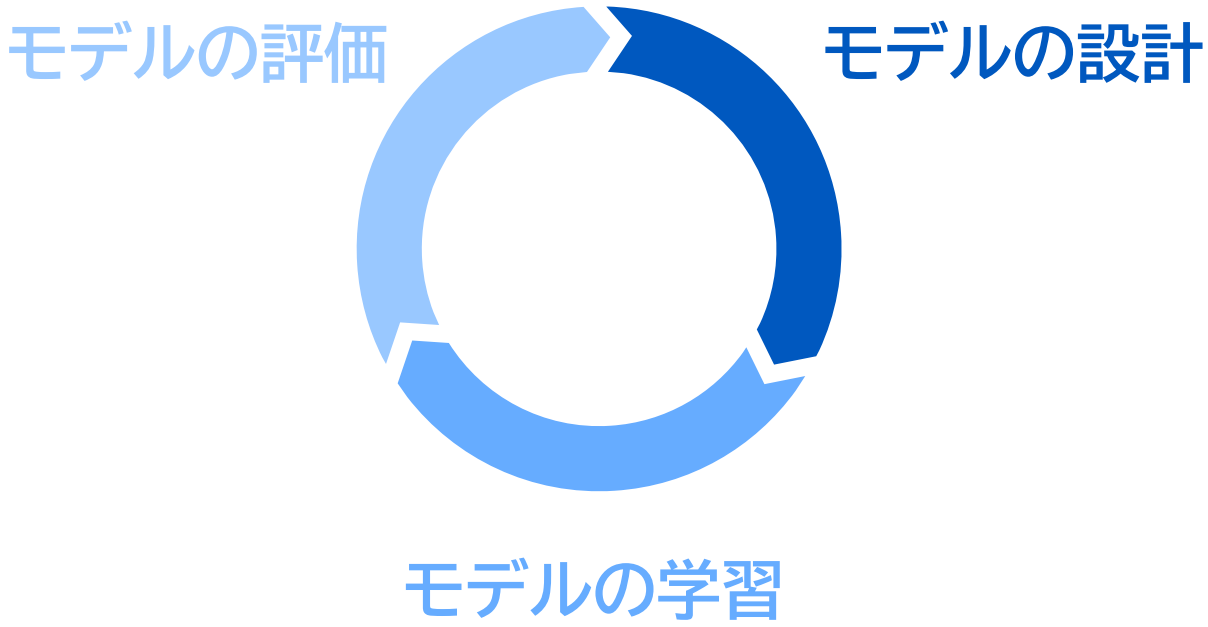
モデル中心開発

データ中心開発

MLモデルの開発は，サイクルを回して実施

- 1回の学習プロセスで完了することはない





データセットの部分集合における性能

- 特定の属性や特徴を用いたことによる評価指標の解釈も重要
- 例1:ローンの審査
  - 性別や宗教, 出身地など保護属性によって審査基準が設けられないようにする
- 例2:顔認識モデル
  - 年齢が高い被写体, 暗い照明条件下での画像, 特定の人種や性別グループでの精度検証

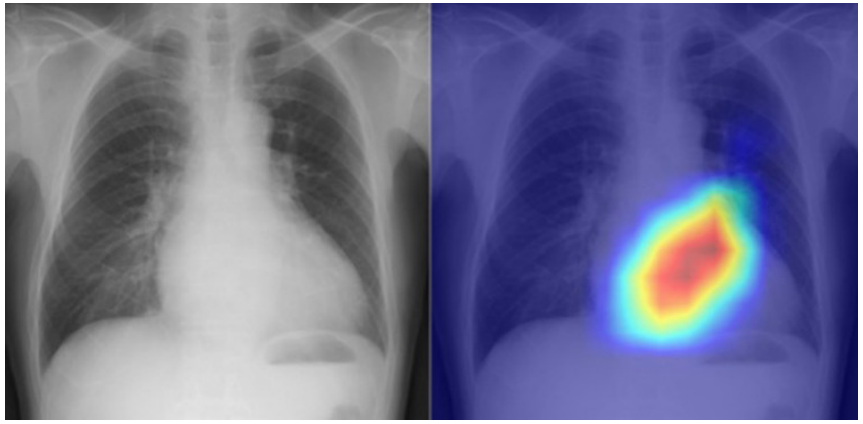
データセットの構成によって, モデルがバイアスを持っている可能性が高い

エッジケースや特定のユースケースでの性能を保証するためにも重要

偏りがあるデータセット

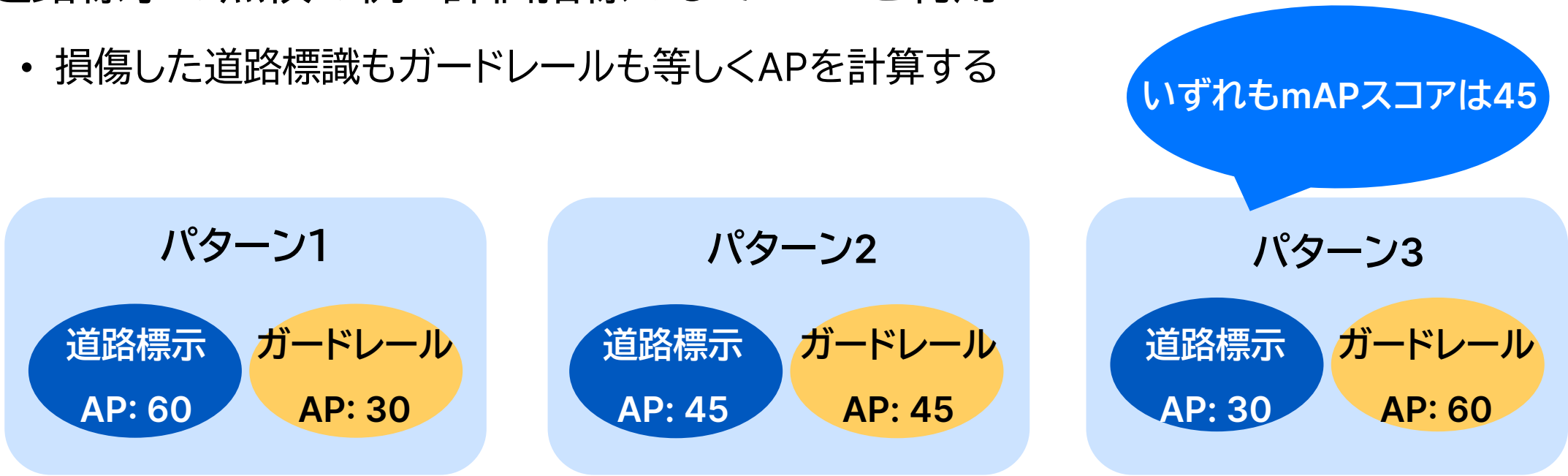
- 異常検出やレアクラスを含む問題も重要
  - 99%が”正常”クラス, 1%が”異常”となっているデータセット
    - モデルが全て”正常”クラスと出力すれば精度は99%
  - 胸部レントゲン画像から所見の発見
    - ヘルニアは稀であるため, 無視しがち

所見	データ数	性能
胸水貯留	10,000<	0.901
肺水腫		0.924
腫瘍		0.909
ヘルニア	<100	0.851



テストデータにおける評価指標の解釈

- 道路標示の点検の例：評価指標としてmAPを利用
  - 損傷した道路標識もガードレールも等しくAPを計算する



実際の利用状況を考えるとどれが正解なのだろうか？

→唯一解はなく，現場でのトライ/エラーで検証するしかない

実際によくある話として…

- よくある例1： 論文やプレスリリース

「弊社/本論文で開発した深層学習モデルが〇〇タスクにおいて，
精度98%のパフォーマンスを出すことに成功しました」

- 一見性能が高く，有用そうに思われる
- ただし，システムとして使うとうまくいかないことも多い
  - 評価指標とユースケースがあっていない
  - ベンチマークにoverfitされすぎている = 実データと異なりすぎている

実際によくある話として...

- よくある例2: プロダクトオーナーと開発者の会話

開発者:「新たなタスクのテストケースで90%出せました」

PO:「99%じゃないとプロダクトにできないよ」

開発者:「いやそもそも難しいタスクだから...」

どうやってそのタスク「難易度」について合意を得るか？

合理的なベースラインを設けることが重要

目標とするベースラインをどうやって設けるか？

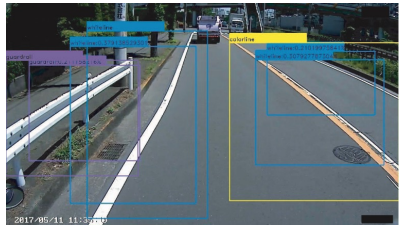
- 例：自動運転システム
  - (残念ながら)100%の誤検出や不適切な予測・計画が行われてしまう
  - →100kmあたりの事故率が人間より低かったら良い？
- 例：音声認識システム
  - 人間の認識精度と同等以上になれば良い？

利用環境	モデル精度	Human Level Performance	改善幅
静かな場所	94%	95%	1%
大通り	89%	93%	4%
カフェ	87%	89%	2%
工事現場	70%	70%	0%

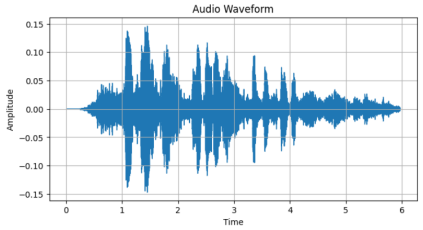
非構造データ v.s. 構造データ

非構造データ

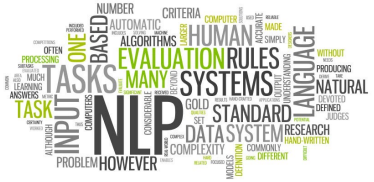
画像/動画



音声

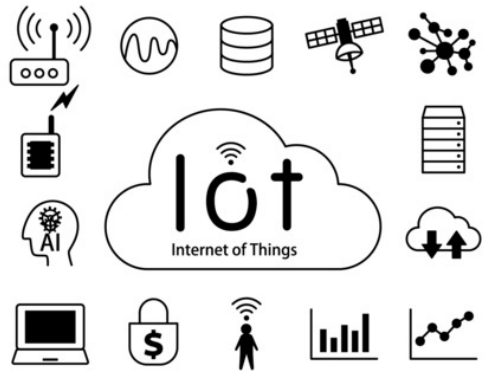


テキスト



構造データ

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	0	34.5	0	0	330911	7.8292	NaN	2
1	893	3	Wilkes, Mrs. James (Ellen Needs)	1	47.0	1	0	363272	7.0000	NaN	0
2	894	2	Myles, Mr. Thomas Francis	0	62.0	0	0	240276	9.6875	NaN	2
3	895	3	Wirz, Mr. Albert	0	27.0	0	0	315154	8.6625	NaN	0
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	1	22.0	1	1	3101298	12.2875	NaN	0
5	897	3	Svensson, Mr. Johan Cervin	0	14.0	0	0	7538	9.2250	NaN	0
6	898	3	Connolly, Miss. Kate	1	30.0	0	0	330972	7.6292	NaN	2
7	899	2	Caldwell, Mr. Albert Francis	0	26.0	1	1	248738	29.0000	NaN	0
8	900	3	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	1	18.0	0	0	2657	7.2292	NaN	1
9	901	3	Davies, Mr. John Samuel	0	21.0	2	0	A/4 48871	24.1500	NaN	0



ベースラインの設け方

- 大体どれくらいの精度が出せるか見積もるためにも必要
  - Human Level Performance
    - ベイズ誤り確率の大体の目安にもなる
  - SOTAやOSSの精度を見る
    - SOTA + ノイジーなデータ << シンプルな手法+綺麗なデータ
  - フルスクラッチ実装
    - シンプルなものならいいが, 最近のSOTA系はあまり動かないイメージ
  - 旧MLシステムの性能
    - すでに動いているものがあれば

評価方針

- 精度や公平性/バイアス, その他の問題を評価する必要がある
  - システムがうまくいっているか判断する指標の洗い出し
    - 一部のデータでの性能(性別や宗教, 特定のドメインなど)
    - FalsePositive/FalseNegative/F1スコアの解釈
    - 不均衡タスクでの性能
  - これらの問題を確認するためのサブセットと指標の定義
    - 異なる環境下での平均精度
    - 異なる属性値ごとの平均精度
    - アプリケーションでのユーザの利用状況(時間や頻度)

モデル開発で苦労しないために

- 一発で精度が出るモデルができることは少ない
  - 本質的な課題に直面しているのであれば, (問題はあるが)問題はない
  - 一方で, 自身で実装したコードやOSSを持ってきているときは要注意

解決策

- 小さいデータに過剰適合させる
- 例: 画像生成 -> 1枚だけ訓練させて, 生成させる
  - 訓練データを生成できるはず
  - もしできないならば, 学習コードもしくは可視化周りでバグを仕込んでいる可能性が高い

デプロイ時の制約の扱い

- モデルの選定/開発するときに，デプロイの時の要件をどこまで考えるか？
  - エッジデバイスに乗るのか/スループットはどれくらいか？
  - 個人情報/プライバシー保護はどうするのか？

➡ 考えるべき

- すでにベースラインとしてある程度動くことがわかっていて，デプロイが目標の場合

➡ 考えなくて良い

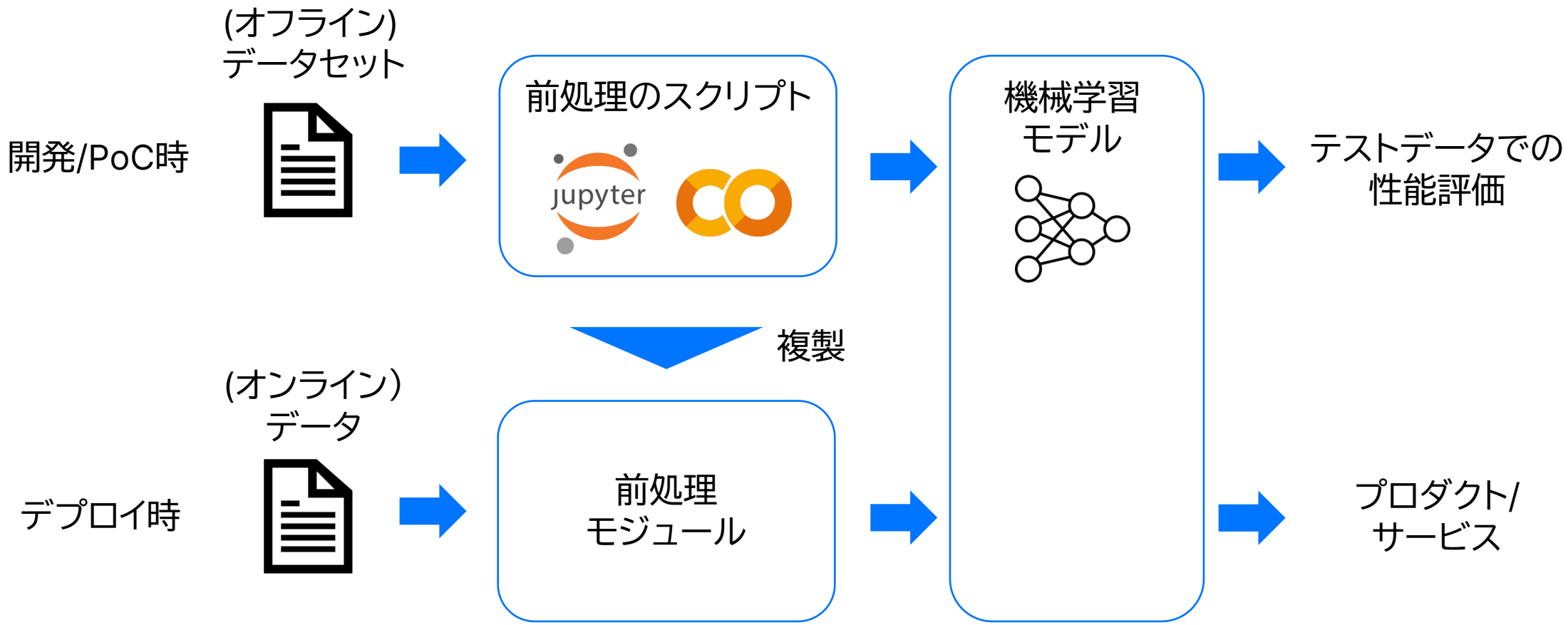
- まだベースラインができていない場合

各段階でのポイント

- 開発/Proof of Concept時
  - **目的** その「タスク」が実現可能であること，そして十分に性能が出ることを示す
  - **体制** とにかくプロトタイピングに注力する
  - **コツ** 前処理も綺麗に書かなくても良い（ただし後々見返すためにコメントは残す）

- デプロイ時
  - **体制** 設計からきちんと見直し，リファクタリングに努める
  - **コツ** メンテナンスされている/プロダクションレディなツールを採用する

各段階でのデータパイプライン



データ収集

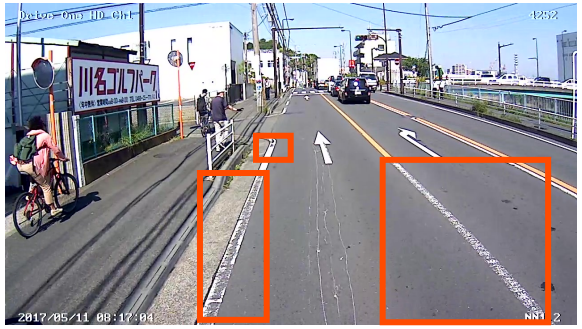
データ定義

- 基本的に，すでに「データセット」ができていることはない
  - 自分自身でデータセットを構築することが大事
  - 入力データxは？
    - 非構造データなら：
      - カメラ解像度 / 明るさ / 画角
      - 表記揺れ / 系列長
    - 構造データなら：
      - 含めるべき / 外すべき属性値は？
  - 正解ラベルyは？
    - どうやって，アノテータに一貫したラベルをつけてもらうか

道路標示の場合

ラベル付けのルール：

「損傷した道路標示を枠で囲む」

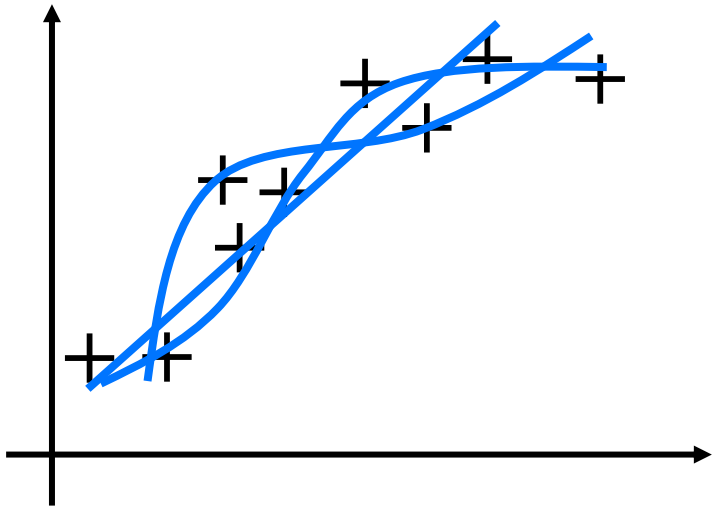


○

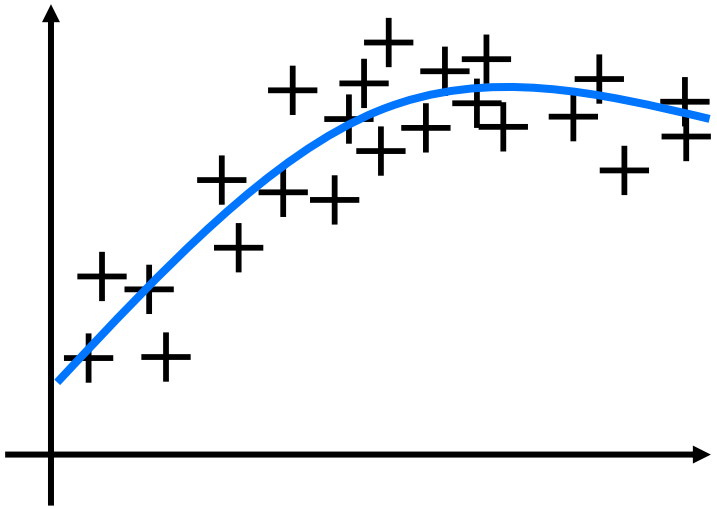
○

×

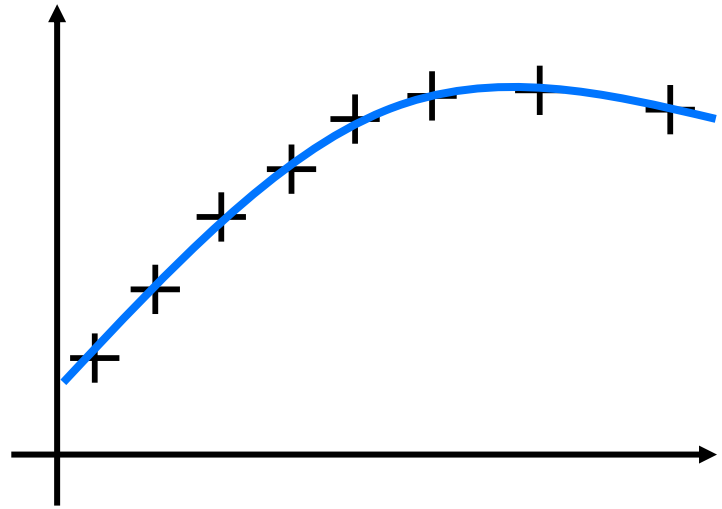
ラベルの一貫性の重要性



- 少数データ
- ノイズありラベル



- 大量データ
- ノイズありラベル



- 少数データ
- ノイズなしラベル

ラベル一貫性の向上

- 複数人のアノテータ間で合意形成を得るようにする
  - 数人でやる時は, 特定のデータについてしっかりと議論する
  - もしくは多数決でラベルを決定させる
- 表記揺れ:統一にする
- クラス:ラベルの種類を増やす
  - 0 / ボーダーライン / 1

データにおけるよくある問題

	非構造データ	構造データ	
少量	点検データ	家賃データ	<=10000
多量	Webのテキストデータ	ECサイトの購買履歴	10000<

綺麗なデータを
作ることが重要

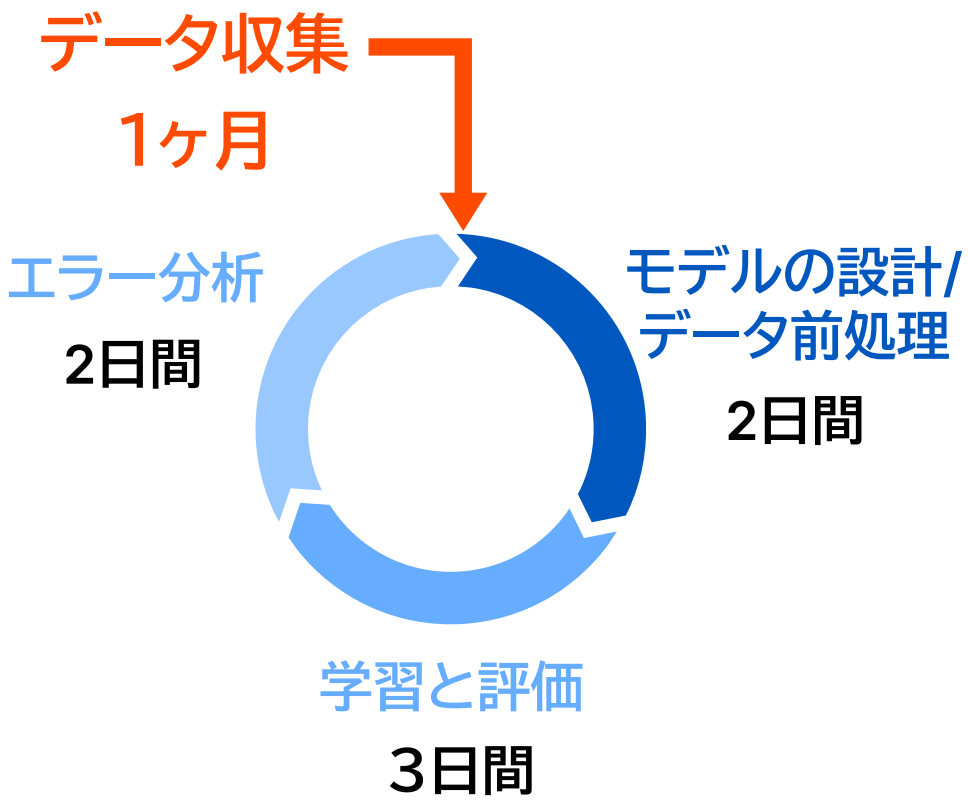
データ収集時点で
気を付ける

ラベル付けしやすい(HLP)
データ拡張しやすい

データ合成や
データ拡張しにくい

データの収集期間の見積もり

- 可能な限り早く，サイクルを回せるようになることが重要



- サイクルとしては7日間でいける
- (必要量の)データ収集に1ヶ月必要

勿体無い

「m個のデータを集めるのにどれくらいかかるか？」

「k日データを収集したらどれくらい集まるか？」

アノテーションのトレードオフ

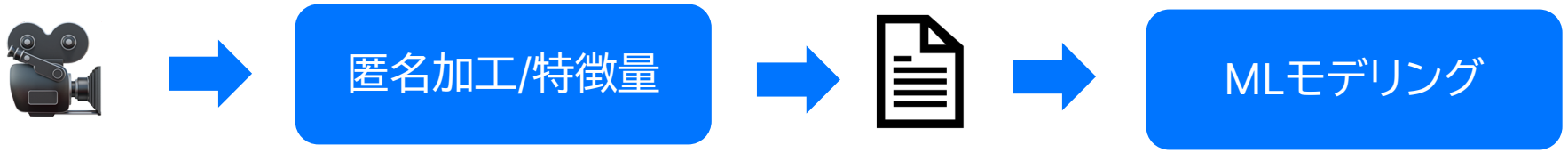
- 時間と費用のバランスが重要
  - データがテキストや画像なのか, 動画なのかでも変わる
    - 動画だとアノテーションするために再生が大変
  - 他の要因もある：データの質, プライバシーの観点, 規制周りの観点など
    - 専門家に依頼するかどうかもかなり重要

ソース	データ量	時間	費用
自前/自作	1,000	3日	0円
クラウドソーシング	10,000	14日	10万円
業者ラベリング依頼	3,000	7日	7万円
データ購入	10,000	1日	15万円

早くサイクルに入るためには,
自前/購入が良さそう

生データからデータセットへ

- データのフォーマット
  - 動画(.mp4) / (.hdf5) / numpy.array (.npy)
  - テキスト(.txt) / json (.jsonl) / csv(.csv)
- ファイル構造
  - 用途 > クラス > ファイル / クラス > 用途 > ファイル / メタデータ + ファイル
- 事前処理
  - 匿名化加工 / コーデック / 特徴量変換(抽出)



データセットの分割

- 2値分類問題： 100サンプルのうち, 30サンプルが陽クラスとする

訓練:検証:評価 = 60%:20%:20%

大規模データセットの場合は
あまり気にしなくて良い
(ことが多い)

悪い例

各分割に含まれる陽クラス

21:2:7 ➡ 55%:10%:35%

良い例

各分割に含まれる陽クラス

18:6:6 ➡ 30%:30%:30%

`sklearn.train_test_split` では, 引数 `stratify=y` を使うことで分布を維持できる

モデル改善

モデルのエラー分析

- 道路標示検出
  - 一部の画像に映り込みが誤検出を誘引
    - 帽子屋やワイパーなど
  - 雨の日の検出精度が悪い
    - 晴れの日に比べるとデータ件数が少ない
- 音声認識
  - 車の走行音や街の騒音
  - (他の)人の声が混ざってしまっている
  - 声が小さい

エラー分析を踏まえた次の施策をどうするか？

- モデルの改善をした時に，一番インパクトがあるのは何か検証する

利用環境	モデル精度	Human Level Performance	改善幅	データの比重
静かな場所	94%	95%	1%	0.6%
大通り	89%	93%	4%	0.16%
カフェ	87%	89%	2%	0.6%
工事現場	70%	70%	0%	0%

「大通り」より，「静かな場所」「カフェ」での精度改善に取り組むのが良い

次の施策の決め方

- どのカテゴリに注力していくか？
  - 改善幅がどれくらいあるか
  - そのカテゴリの出現頻度はどれくらいか？
  - 精度改善は容易に可能か？

- 特定のカテゴリにおけるデータの追加・改善
  - もっとデータを収集する
  - データ拡張を利用する
  - ラベルやデータの精度・質を高める

データ拡張

目標

(1)人間(やその他のベースライン)がうまくいく
(2)モデルがうまくいかない

} 本物っぽいデータを作る

チェックリスト

- 本当に本物っぽいデータになっているか？
- タスクとして成立しているか = $x \rightarrow y$の写像は明らかか
  - 人間が本当にできるのか

道路標示の例

元の画像

○

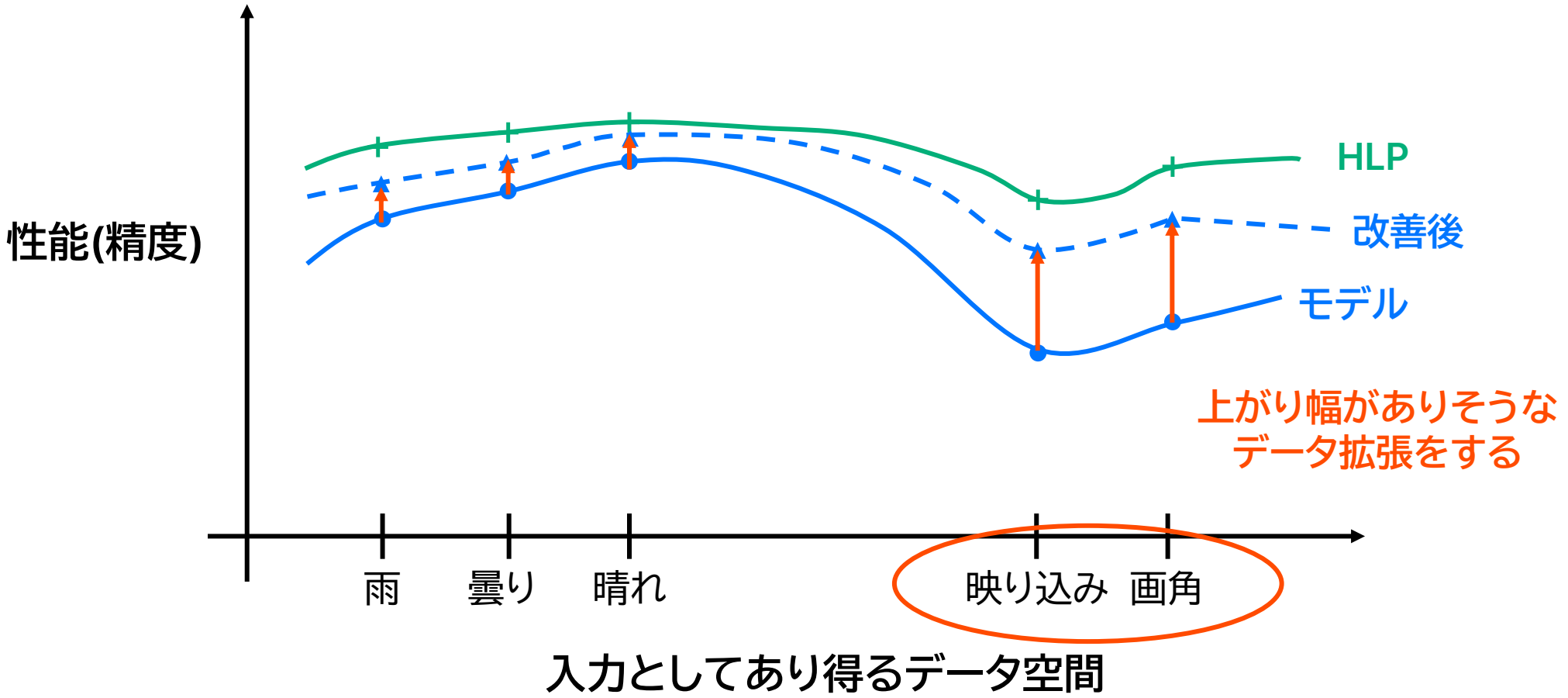
○

×

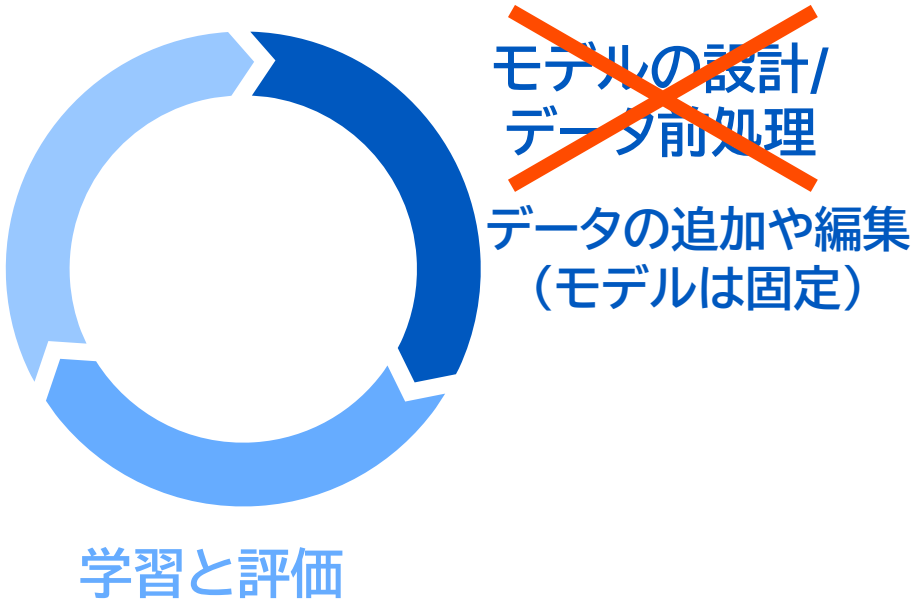
画像加工

損傷を追加

道路標示におけるデータ拡張戦略



データ中心開発

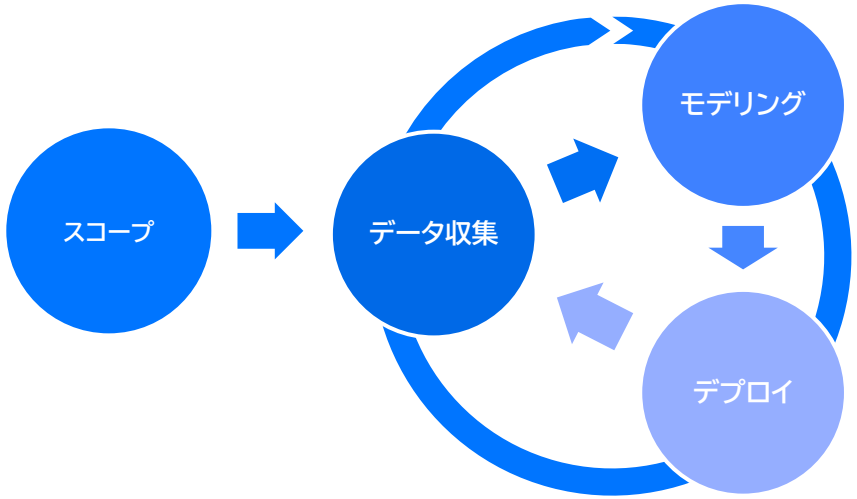


- モデルを中心に開発
  - 論文ではベンチマークを利用するため

- データを中心に開発
  - モデルは固定
  - データ拡張をしたときの性能を検証
  - サイクルを回して改善していく

講義まとめ

- MLプロジェクトにおいて，大規模データは大事だが，その質も大事
  - MLプロジェクトの各段階で一貫性のある高品質データを用意すること



- 高品質データとは
  - 入力される可能性があるデータを網羅すること
    - 入力xのカバレッジをあげる
  - 一貫性のある定義
    - ラベルyのルールにおける曖昧性の排除
  - デプロイ後の即時フィードバック
    - データ/コンセプトドリフトの検知

実データに注目して，サイクルを回すことがプロジェクト成功の秘訣