#18 Turing × atmaCup solution

2024 11/14〜11/24

目次

EDA

やること

- 実際の走行シーンのカメラ画像と、信号検出結果、車両の状態データから車両の位置を
  6つの点から予測する必要がありそれぞれ (x, y, z) の3つの座標を持ちます。ですので結局ある走行時情報から18個の値を予測する問題

データ

- ['/analysis/data/raw/images',
- '/analysis/data/raw/test_features.csv',
- '/analysis/data/raw/atmacup18_dataset.zip',
- '/analysis/data/raw/train_features.csv',
- '/analysis/data/raw/traffic_lights']

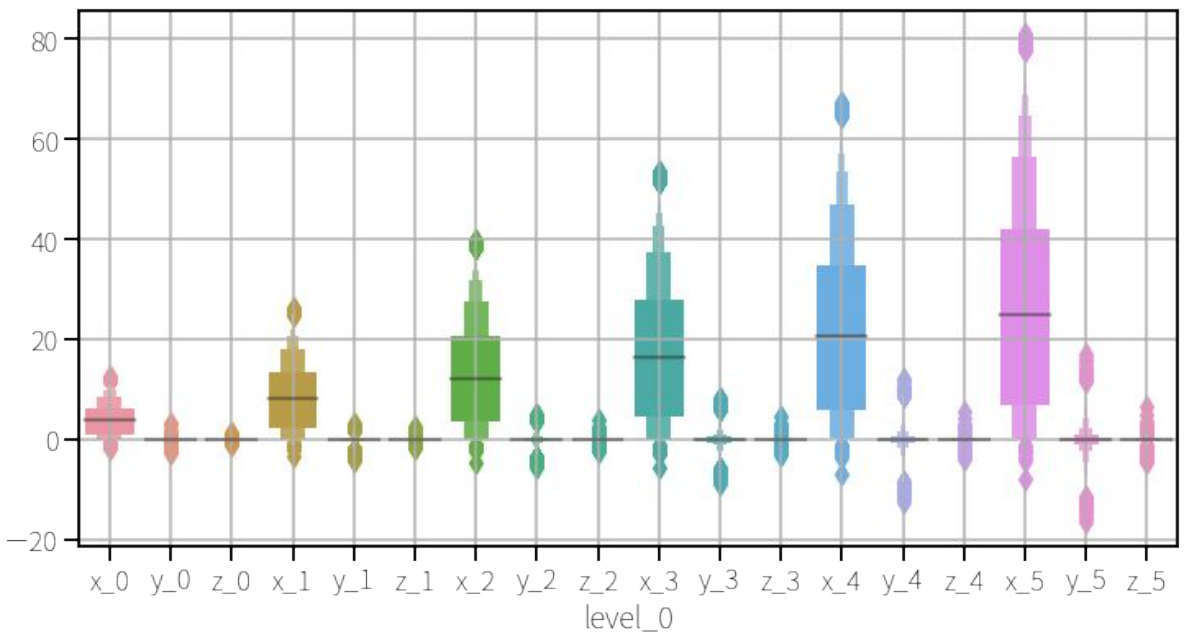
評価

- 絶対値誤差の平均

	0
ID	00066be8e203
vEgo	5.701526
aEgo	1.538456
steeringAngleDeg	-2.165777
steeringTorque	-139.0

予測の分布

- * 値が大きく変わるのは`x` 系の値
- * `x` の値の変化は線形に見える
  これは車は動いたらすぐ止まれないからっぽ
- * `y` の値はプラスとマイナスに同じぐらい分布。
  これは右に曲がるケースと左に曲がるケースが
  同じぐらいあるからと言えそ
- * `z` の値はほかの2つと比べて一番動きの幅が小さい
  よーわからない



train_features.csv

- - ID: 走行 ID.
- - ID は **`{シーンID}_{シーンのデシ秒数}`**で構成
- - 例えば **`"0a7d64c1e7fda174fc26c7cedd49d7"`** であればシーン ID **`0a7d64c1e7fda174fc26c7ce`** の 32 秒経過時の走行情報
- - vEgo: 車の速度。単位は [m/s]。
- - aEgo: 車の加速度。単位は [m/s^2]。
- - steeringAngleDeg: ハンドル角度. 単位は度数で半時計回りが正。
- - steeringTorque: ハンドルのトルク。単位はN・mで半時計回りが正です。
- - brake: ブレーキペダルのふまれ具合。0 ~ 1 の間の値で一番奥までふまれている場合が1。
- - brakePressed: ブレーキペダルがふまれているかどうか。
- - gas: アクセルペダルのふまれ具合。0 ~ 1 の間の値で一番奥までふまれている場合が1。
- - gasPressed: アクセルペダルがふまれているかどうか。
- - gearShifter: シフトノブの状態
- - leftBlinker: 左のウィンカーがついているかどうか
- - rightBlinker: 右のウィンカーが付いているかどうか

traffic_lights: 信号機の情報

- あるシーンの画像から検出された信号機の情報が**`json`** 形式で含まれています。
- - index: 検出結果の番号
- - class: 検出された信号機の種類
- - **`green`**: 青色信号
- - **`yellow`**: 黄色信号
- - **`red`**: 赤色信号
- - **`straight`**: 矢印信号機の直進
- - **`left`**: 矢印信号機の左折
- - **`right`**: 矢印信号機の右折
- - **`empty`**: 点滅が観測できない信号機
- - **`other`**: それ以外の信号機
- - bbox:
- - 検出された領域の bounding box 座標. 順に [左上の x 座標 / 左上の y 座標 / 右下の x 座標 / 右下の y 座標]

images: カメラの画像

- 走行時に撮影された画像が含まれているディレクトリです。時刻**`t`** を基準として **`t-1秒`**, **`t-0.5秒`**, **`t秒`** の 3 つの時刻で撮影された画像が含まれます。単眼カメラのRGB 画像で幅 128pixel, 高さ 64 pixel

56 th solution & baseline

方針(Public: 0.1935 / Private: 0.2017)

- 全データ使うためカメラ情報をCNNで扱い、その予測値とテーブルデータをLightGBMでstackingする。
- 類似コンペの[1st and Future - Player Contact Detection](https://www.kaggle.com/competitions/nfl-player-contact-detection)のパクリ

- カメラ画像(CNNパート)
  - 過去の類似コンペ「Player Contact Detection」での手法を参考に、t-1.0秒・t-0.5秒・t秒の3フレーム画像をチャンネル結合して入力し、CNN（ResNet）で学習・推論。
  - 学習時はK交差検証を行い、検証データに対する**Out-Of-Fold(OOF)**予測を作成し、テストデータに対しても推論。
  - このCNNパートの出力（各ターゲット座標の予測値）をファイルに書き出して保存します(oof_cv.csv, submission_oof.csv など)。
- テーブルデータ(LightGBMパート)
  - 速度やアクセル操作など、元々のテーブルデータを特徴量とし、さらに上記のCNNで得た予測値も特徴量として加え
  - シーンIDごとのグループK折などを設定し、LightGBMを用いて回帰タスクとして最終予測モデルを学習します。
  - その結果としてLightGBMのOOFスコアやテストデータの最終予測を出力します。

1 th solution

1st Place Solution (CV: 0.1792 / Public: 0.1885 / Private: 0.1754)

方針概要

- scene単位で学習＆推論（IDではなく、sceneの連なりを時系列として扱う）
- 1stステージ: CNN + BiLSTM
  - (画像 + テーブル情報) を結合
  - HorizontalFlip・CoarseDropout等の拡張
  - ターゲット正規化 / 補助目的(aux target)導入で精度UP
- 2ndステージ: GBDT
  - 1stステージ出力(main + aux予測)を特徴量化
  - 車両ダイナミクスモデル等の出力も組込み
- アンサンブル
  - ResNet / EffNet / Swinなど多数のCNNモデルを加重平均
  - モデル数を増やすほど精度向上し、最終 11モデルを採用

主な改善効果

- ID➡sceneに切り替え: CV 0.2259 ➡ 0.1981
- HorizontalFlip推論: CV 0.1963 ➡ 0.1906
- 補助目的(aux target) 追加: CV 0.2312 ➡ 0.2288

「テーブル＋CNN＋時系列（BiLSTM）」でsceneレベルの文脈を取り込み、多様なモデルを最終的にアンサンブルすることで高精度を実現。

2 th solution

2st Place Solution (CV: 0.1826 / Public: 0..1890 / Private: 0.1791)

1. 1stステージ: 画像 + テーブル特徴 のNN

1. **高解像度化した画像**を利用
   - Stable Diffusion系の超解像モデル(CompVis/ldm-super-resolution-4x-openimages)を用いて、(64×128)→(256×512)に拡大
   - Swin TransformerなどTransformer系バックボーンで、精度・安定性を向上
2. **画像エンコーダ出力 + テーブル特徴**を複数層MLPに通す構成
   - テーブルMLPの次元数を大きくし、画像出力との結合後は1層MLPを追加
3. **様々なバリエーションモデルを多数生成**
   - 入力画像の組み方(Depth単独, 信号機マップ追加 3枚を結合・別々入力)
   - バックボーン(Swin/ConvNeXt/MaxVit/EfficientNet/ResNetなど)
   - 結果の傾向がモデルごとにバラつく アンサンブルでリスク分散

この1stステージで計20個ほどのモデルを作成し、**Foldごとの出力もそれぞれ保存** しておきます。

2. 2ndステージ: GBDT (LightGBM / XGBoost)

- **1stステージで得た予測値 (各Fold別) + テーブル特徴**を特徴量に利用
- 特に
  1. LightGBMとXGBoostの両方を試してモデル化
  2. 「1stステージ各Foldの推定値」をそのまま2ndステージに入れる
     - Fold平均ではなくFold別の値を使うことで、
       学習データ/検証データの分布差を小さく
- モデル間の組み合わせを複数作成し、最終的に多数のGBDTモデルを作成

3. アンサンブル戦略

- 最終提出は、**2ndステージのGBDTモデル**たちの予測を平均
  - 1stステージからのシングルモデル×Fold数(5fold)×GBDT(LightGBM / XGBoost)
  - さらに「複数の1stステージモデルをまとめたGBDT」も用意
  - 重要モデルには大きめウェイトをかけ、最終的にすべてをブレンド

まとめ

- **「画像＋テーブル NN」×「GBDT」×「大量アンサンブル」** が核
- 画像は超解像で解像度を上げ、多様なバックボーンを試行
- Fold別予測を2段目に活用しつつ、多数のモデルを加重平均
- 大量の学習と推論を要するが、ばらつきを利用したアンサンブルで最終的に精度を高めている

3 th solution

3rd Place Solution (CV: 0.1844 / LB: 0.1904 / Private: 0.1795)

全体方針

1. NNパート
2. Stacking Part1 (LightGBM & CatBoost)
3. Stacking Part2 (Ridge回帰)

この三段構成で最終予測を行っています。

1. NNパート

(A) モデルバリエーション（計5パターン × バックボーン違いで14モデル）

1. 画像 + テーブル特徴
2. 画像 + 信号機 + Depth(DeepAnythingV2) + テーブル特徴
3. 上記(1)〜(2)の各モデルに対して、速度・加速度(微分値)を補助損失(aux loss)に追加
4. (2)のモデル出力(予測軌跡)を画像に重ね書き→ t時刻の入力画像として再学習

注意: t-1.0, t-0.5の画像と軌跡オーバーレイ画像のサイズ調整(iffやshift ... デシ秒が大きく離れるケースは欠損扱いとしNN投入前に標準化→0埋め

2. Stacking Part1

(1) LightGBM (fair loss, 各次元単回帰)

- 速度・加速度のdiff/shiftなども特徴量に加えている

(2) CatBoost (MultiRMSE loss, 多次元回帰)

- こちらも速度・加速度のdiff/shiftを追加

3. Stacking Part2

- LightGBM & CatBoost で得られた予測値を入力し、Ridge回帰を各次元ごとに最終スタッキング
- これにより最終的な予測を出力

うまくいかなかった施策

- 次scene画像をNNの追加入力にする
- 予測軌跡のみを単独チャネルとして追加
- LSTMを使ったスタッキングなど

4 th solution

4th Solution (CV: 0.1865 / Public: 0.1977 / Private: 0.1806)

全体像

1. CNNで画像をベースに予測
    - カメラ画像+ Depth Map + Optical Flow(RAFT利用) を入力
    - さらに線形回帰との誤差を学習したモデルや3フレーム(t-1.0, t-0.5, t秒)を縦に繋げた特殊画像を使ったモデルなど複数を作成
    - 合計7つのCNNモデルを採用
2. LightGBMでスタッキング
    - x, zは「線形回帰との誤差」を予測しyは直接予測
    - ターゲット(x, y, z)毎に使う特徴量を切り替え
    - あるターゲットの推測結果を別のターゲットの特徴量として利用

1. CNNパート詳細

- DepthMap: 事前学習済みモデルから推定
- OpticalFlow: RAFTを使用して連続フレームの動き推定
- 複数のCNNモデルを用意
    - 例: 線形回帰予測との差分を学習 3フレームをまとめて1枚に / vEgo・aEgoなどテーブル情報も同時に予測 次時刻の画像も活用
- 最終的に7モデルをアンサンブル

2. LightGBMパート詳細

- x, zは「(実値 - 線形回帰予測値)」を学習し、推論時に線形回帰予測を足し戻す
- yはそのまま学習
- ターゲット別に投入する特徴量を変更
    - 例: xのLightGBMは「過去のx予測」を含める、y/zは「xの予測結果」を含める
- 追加特徴量
    - CNN出力予測
    - 該当時刻の移動量や速度(vEgo, aEgo から計算)
    - DepthMapから算出した前方オブジェクトとの距離
    - 同一シーンの前後時刻情報
    - 複数種(線形/リッジ/ラッソ/ElasticNet)の回帰モデルによる予測
- 最終スコア: CV=0.1865, Public=0.1977, Private=0.1806

まとめ

- カメラ画像＋Depth＋Optical Flow に加えて様々な追加予測を特徴量化
- x/zは差分予測方式、yは直接予測などターゲット別に戦略を変える
- 複数手法を組み合わせたCNN→LightGBMの2段構成で最終精度を高めている

6 th solution

6th Place Solution (CV: 0.1826 / Public: 0.1928 / Private: 0.1818)

全体の流れ

1. LightGBM Stacking (2段構成)
2. CNN + Metadata (画像とテーブルを同時入力)
3. MLP Stacking (メタ特徴だけをNNでスタッキング)
4. 上記モデル群をターゲットごとに重み調整してアンサンブル

1. LightGBM Stacking

- 最初の公開ノートブックとほぼ同じ手法
- 追加対応として、同一scene内で**デシ秒の飛び(200以上)**がある場合のshift/diff特徴量を導入→ CV +0.0015
- t-1段目のCNN予測などを含めた多数特徴量を用いて再度LightGBMで回帰

2. CNN + Metadata

- 画像 + テーブル特徴を同一モデルに入力するNN
  - backbone: resnet34d, legacy_seresnext26_32x4d, tf_efficientnetv2_s.in21k_ft_in1k など
  - 画像のチャンネル数にはDepthMapを追加（inoichanさんのdiscussionを参照）
  - MLP層でテーブル特徴を変換し、CNN出力と結合→最終線形層で座標を出力
- これで画像とメタ情報を一括学習するモデルを作成

3. MLP Stacking

- 2段目を LightGBMではなくMLPでスタッキングしたところ、精度が大幅向上
- 主に以下を実施
  - OOFなどから得た特徴量(各種モデルの予測値 + テーブル情報など) をMLPへ投入
  - 複数層のGELU + LayerNorm構成で深めのMLPを構築
-
- これをチューニングした結果、CV 0.1897 まで改善

4. アンサンブル

- ターゲットごとに (CNN+Metadata, LGBM Stacking, MLP Stacking) の予測精度を比較すると、
  - 短い将来(t+0.5秒, t+1秒) は LGBM stacking が良い
  - より長い将来 は NN が良い
- そこでターゲット別に重みを変え、Nelder-Mead で最適化
- 最終アンサンブル : CV: 0.1826 / Public: 0.1928 / Private: 0.1818

まとめ

- LightGBM + CNN の2段構成をベースに、「 CNN + メタ情報同時学習」モデルや **「MLPによるスタッキング」**を追加
- ターゲットごとにモデルの強みが異なるため、重み付けして最終的に高精度を実現した

7 th solution

7th Place Solution (CV: 0.1915 / Public: 0.1923 / Private: 0.1822)

全体アプローチ

1. CNNで差分を予測 → LightGBMでスタッキング
2. CNN → LGBM の構成を 3種類 つくり、それぞれの予測を最終的にアンサンブル（加重平均）

1. 差分予測の採用

- 座標を直接予測するのではなく、x0(そのまま),x1−x0,x2−x1,... のように連続フレーム間の差分を予測する形に変更

  $x0(\text{そのまま}), x1-x0, x2-x1, \dots \quad x\_0(\text{そのまま}), \quad x\_1 - x\_0, \quad x\_2 - x\_1, \dots$

- 学習後に差分を累積和で戻して座標値を求める
- この工夫で LB が 0.0011 改善

2. CNNパート

1. 画像情報のみで学習した CNN (backbone: ResNet200d)
2. 画像・Depth・テーブルを合わせた CNN + MLP (backbone: ResNet18)
   - 各フレーム(t-1.0, t-0.5, t秒)を1モデルに入れる形
   - 前後シーンの一部情報を補完しながら使う

※ ほかにも Augmentation やフレーム処理の工夫を加え、多数モデルを作成

3. LightGBMパート

- ラグ特徴量の作り方
  - シーン内で時刻が飛んでいる(200デシ秒空く)箇所は前後の値を平均補完 → shift → 欠損除去
- 重要なカテゴリ情報(例えば「highway」「intersection」等) をOne-hot化で追加
- 差分予測を生かした特徴量
  - 予測済みの [x0,y0,z0],[x1−x0,...] を次以降の予測に利用

    [x0,y0,z0],[x1−x0,...][x_0,y_0,z_0], [x_1 - x_0, ...]
  - 0.5秒間隔の移動距離・速度・加速度を算出→さらにその差分も加える
  - これが単体モデルとして最も効いた(LB: 0.0041 改善)

4. アンサンブル戦略

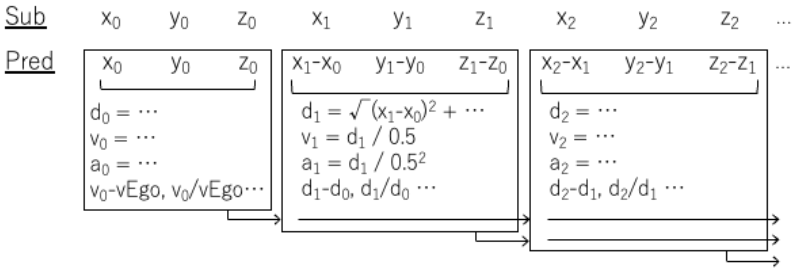
- 3種類のCNN→LGBMモデルを加重平均
- Publicの結果が良い2モデルアンサンブル/ CVが良い3モデルアンサンブルを比較・試行
- 最終的にCV 0.1915 / Public 0.1923 / Private 0.1822

その他トライアル

- 画像出力に対するVLMのembedding次元圧縮特徴
- Shiftレンジ変更
- 信号機特徴やシーン全体の信号出現回数
- 車線情報の抽出(VLM出力や信号機の座標活用)
- CNNのAugmentation変更
- (後日実験) “t-0.5時点の速度・加速度” を予測し、LGBMに再度スタッキング → 最終的にCV 0.1904 / Private 0.1815 へ追加改善

まとめ

- 座標を差分予測するというアイデアを軸に、CNN + LGBM の2段構成を3パターン作成し加重平均
- 時系列の飛びを補完し、連続的に差分を活用した特徴量を充実させることで精度を上げている



discussion

とくになし

- https://www.kaggle.com/competitions/nfl-player-contact-detectionがプレイヤー接触検出のが類似コンペなこと