



コンペ支援ライブラリ **nyagggle** の設計

2020/02/08 CA × atmaCup#3 データコンペ振り返り LT発表

Nomi

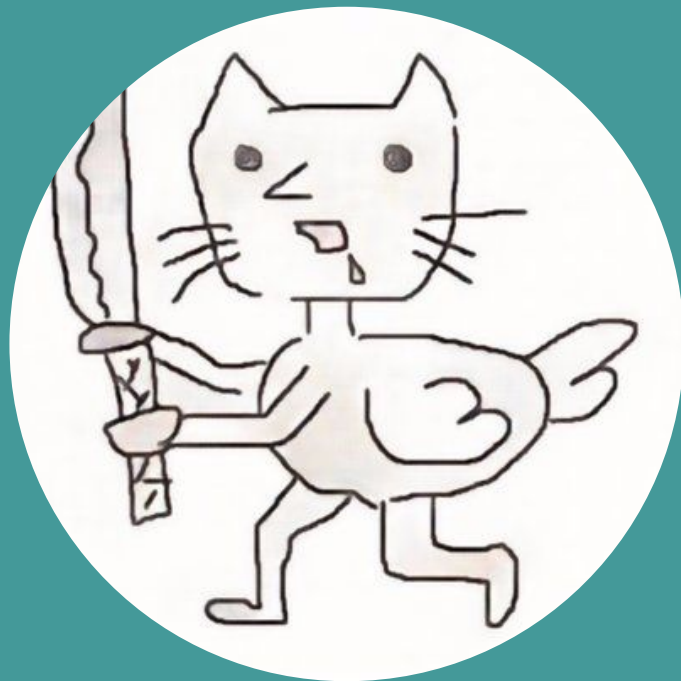
Kaggle: nyanpn

Twitter: nyanp

GitHub: nyanp

ここから200mくらいの会社で
働いています。

趣味: 特徴量エンジニアリング



今日の内容:

データ分析コンペに準備するコードは人それぞれ。
自分が**atmaCup#3**に向けて、
どんな考えで、何を作ったのか紹介します。



きっかけ

コンペの度に大体同じようなコード
書いてる気がする



大体こんな事を繰り返すので

- 特徴量を作ってfeatherで保存
- CVを評価
- 評価結果＋ログ＋色々な生成物を保存しておく
 - OOF予測値, テスト予測値, 学習済モデルetc



大体こんなのをいつも書いてる

```
oof = np.zeros(...)
test = np.zeros(...)

# CV
for train_index, valid_index in kf.split(X, y):
    X_train, y_train = X.iloc[...], y.iloc[...]
    X_valid, y_valid = X.iloc[...], y.iloc[...]
    ...

    model.fit(...)
    oof[valid_index] = model.predict_proba(X_valid)
    test += model.predict_proba(X_test)

# 予測値とかモデルとか importanceとか評価値とかを記録しておく
np.save(...)
logger.info(f'score: {roc_auc_score(...)}')
```



課題

- 毎回同じことをする無駄
- うっかりログを忘れるかも
 - チームメイト「スタッキングするのでOOF予測値ください」
- ちょっとアルゴリズムを変える、が意外とできない
 - 微妙なAPIの差、カテゴリ変数の扱いの差
 - ハイパラを忘れてる
 - コンペ中はつい汎用性を意識せずに書きちゃう
- 実験結果の一覧性に乏しい

nyagggle

Code for Data Science Competition

<https://github.com/nyanp/nyagggle>

nyagggle.experiment

```
from nyagggle.experiment import run_experiment
```

```
# モデルパラメータを決めて
```

```
params = { 'max_depth': 8 }
```

```
# X, yと一緒に渡せば (Test Dataはoptional)
```

```
result = run_experiment(params, X_train, y, X_test)
```

```
# 結果が一式返ってきて、フォルダに成果物が保存される
```

```
result.oof_prediction
```

```
result.models
```

```
...
```



(成果物)

output/

20200120123456/

params.txt

metrics.txt

oof_prediction.npy

test_prediction.npy

20200120123456.csv

importance.png

models/

デフォルトはymdHMS(変更可)

実験パラメータ

CV評価値

Out-of-Fold予測値

Test予測値

提出用CSV

importance plot

fold毎のモデル



简单！



やらないこと

- 特徴量は自動で作らない
- モデルは基本的に指定した通りのパラメータで学習
 - Early Stoppingだけは自動適用
- つまり、ただのロギング付きCross Validation

いろいろなモデル

```
# sklearn Estimator
```

```
result = run_experiment(params, X_train, y, X_test,  
                        algorithm_type=LogisticRegression)
```

```
# CatBoost
```

```
result = run_experiment(params, X_train, y, X_test,  
                        algorithm_type='cat')
```

```
# XGBoost
```

```
result = run_experiment(params, X_train, y, X_test,  
                        algorithm_type='xgb')
```

いろいろなバリデーション

5-Fold

```
result = run_experiment(params, X_train, y, X_test, cv=5)
```

Stratified 5-Fold

```
result = run_experiment(params, X_train, y, X_test,  
                        cv=StratifiedKFold(5))
```

時系列

```
from nyaggles.validation import TimeSeriesSplit  
result = run_experiment(params, X_train, y, X_test,  
                        cv=TimeSeriesSplit(...))
```

(Option) Optunaによるハイパーパラメータ最適化

```
# LightGBM Tunerを使ってチューニングしてからCV  
result = run_experiment(params, X_train, y, X_test,  
                        with_auto_hpo=True)
```

Submission fileのフォーマット

指定したカラム構成に従ってファイルを生成

```
sample_df = pd.read_csv('sample_submission.csv')
```

```
result = run_experiment(params, X_train, y, X_test,  
                        sample_submission=sample_df)
```


mlflowとの連携

```
# with_mlflow = True を指定するだけ  
result = run_experiment(params, X_train, y, X_test,  
                        with_mlflow=True)
```

```
# > mlflow ui でUIを起動すると...
```

Default

Experiment ID: 0 Artifact Location: file:///Users/nomi/atma3/src/mlruns/0

▼ Notes [🔗](#)

None




Search Runs:

State: Active ▼ Search

Filter Params:

Filter Metrics: Clear

Showing 100 matching runs Compare Delete Download CSV 📄

<input type="checkbox"/>	Date	User	Run Name	Source	Versi...	Tags	Parameters	Overall	Metrics
<input type="checkbox"/>	2020-01-25 17:51:46	nomi	../output/7...	 test.py	3a9fd8		features: [2, 4, 6, 100, 1... fit_params: {'early_stoppin... gbdt_type: cat model_params: {'learning_rate':... num_features: 46	0.918	Fold 1: 0.947 Fold 2: 0.945 Fold 3: 0.915 Fold 4: 0.885 Fold 5: 0.939 Fold 6: 0.857 Fold 7: 0.944 Fold 8: 0.921
<input type="checkbox"/>	2020-01-25 17:50:52	nomi	../output/7...	 test.py	3a9fd8		features: [2, 4, 6, 100, 1... fit_params: {'early_stoppin... gbdt_type: cat model_params: {'learning_rate':... num_features: 46	0.93	Fold 1: 0.947 Fold 2: 0.962 Fold 3: 0.923 Fold 4: 0.925 Fold 5: 0.954 Fold 6: 0.871 Fold 7: 0.956 Fold 8: 0.903
<input type="checkbox"/>	2020-01-25 17:50:01	nomi	../output/7...	 test.py	3a9fd8		features: [2, 4, 6, 100, 1... fit_params: {'early_stoppin... gbdt_type: cat	0.927	Fold 1: 0.956 Fold 2: 0.962 Fold 3: 0.918

Default > Comparing 4 Runs

Run ID:	a99923a41ed64c098b9f8f7745122132	ce639b7f0b4c4f0292f66ed761125120	641d481c43c74bdb8ae03a522...	80401606009e4278b9ef83e4bfb...
Run Name:	../output/77_cat_skf_seed1	../output/77_cat_skf_seed3	../output/77_cat_meta	../output/76_cat_wo_te
Start Time:	2020-01-25 17:46:06	2020-01-25 17:47:27	2020-01-25 17:40:10	2020-01-25 17:13:58

Parameters

features	[2, 4, 6, 100, 101, 200, 300, 9, 11, 13, 2...	[2, 4, 6, 100, 101, 200, 300, 9, 11, 13, 2...	[2, 4, 6, 100, 101, 200, 300, 9, 1...	[2, 4, 6, 100, 101, 200, 300, 9, 11, ...
fit_params	{'early_stopping_rounds': None}	{'early_stopping_rounds': None}	{'early_stopping_rounds': None}	{'early_stopping_rounds': None}
gbdt_type	cat	cat	cat	cat
model_params	{'learning_rate': 0.1, 'eval_metric': 'AUC', 'l...	{'learning_rate': 0.1, 'eval_metric': 'AUC', 'l...	{'learning_rate': 0.1, 'eval_metric': ...	{'learning_rate': 0.1, 'eval_metric': 'l...
num_features	46	46	47	44

Metrics

Fold 1	0.937	0.961	0.989	0.887
Fold 2	0.972	0.93	0.964	0.877
Fold 3	0.937	0.934	0.977	0.864
Fold 4	0.934	0.924	0.996	0.975
Fold 5	0.965	0.964	0.964	0.898
Fold 6	0.869	0.84	0.923	0.876
Fold 7	0.941	0.951	0.991	0.935
Fold 8	0.915	0.936	0.982	0.839
Overall	0.933	0.931	0.969	0.89



その他いろいろ

- `nyaggle.feature_store`
 - feather形式で特徴管理をするヘルパ
- `nyaggle.features`
 - sklearn互換の特徴生成器
- `nyaggle.validation`
 - sklearn互換のValidation Splitter
 - Adversarial Validation
- `nyaggle.hyper_parameters`
 - 過去の金圈解法から集めたハイパラのカタログ



nyaggleは単機能APIの詰め合わせ

- pipelineでもframeworkでもない(つもり)
 - end-to-endじゃない
 - 設定ファイルを持たない
 - 特定のクラス構造を使い手に要求しない
 - 制御の反転が無い

※pipeline: 分析に必要な一連のデータ変換、学習のステップを繋いで自動化したもの、くらいの気持ち

※制御の反転: ライブラリのコードがユーザーの書いたコードを呼び出すような形のこと。フレームワークの定義として使われることもある。普通のライブラリだってCallbackはあるし、実際ははっきりした境界はないと思う



Q: 何でパイプラインにしないの？

A: 抽象化には漏れがある。



きっかけ
コンペの度に**大体**同じようなコード
書いてる気がする

だい たい

大体

①ほとんど全体に及んでいるさま。
細部は別にして、主要な部分はそうであるさま。たいてい。

三省堂 大辞林 第三版



細部は毎回違う

コンペの性質

- バリデーションが特殊
- 評価指標が特殊
- 時系列性が強い
- リーク
- Test Dataが死ぬほどでかい
- 2-stage Code Competition
- マルチモーダル
- 日本語、ロシア語、ベンガル語

...

やりたくなること

- モデリング後に後処理
- 外部データの利用
- 日本語NLP
- Pseudo-Labelling
- チームメイトとStacking
- 特殊な欠損埋め
- メタ特徴のためのモデリング
- データの一部だけでモデリング

...



理想: この3つが全部欲しい

スコープ

あらゆる作業をカバー

シンプルさ

覚えやすいAPI

応用力

幅広いコンペで通用



あらゆるコンペで使える**End-to-End**のコード資産
→コードが複雑化、書くのも使うのも大変

スコープ

あらゆる作業をカバー

シンプルさ

覚えやすいAPI

応用力

幅広いコンペで通用

End-to-End かつシンプルなAPI

→ よほど上手く設計しないとコンペに通用しにくい



スコープ

あらゆる作業をカバー

シンプルさ

覚えやすいAPI

応用力

幅広いコンペで通用



シンプルで実用的だが、スコープは狭い
→ これなら自分でも書けそうだし、少しは役に立ちそう

スコープ

あらゆる作業をカバー

シンプルさ

覚えやすいAPI

応用力

幅広いコンペで通用



nyaggleの設計原則

- コンペで通用すること
- すぐ使えること
- すぐやめられること



コンペで通用すること



コンペで通用すること

NG: nyagggleがこのテーブルコンペでは使えない

NG: nyagggleを使うことで精度が犠牲になる



例: nyagggle.experiment

最初はfold数をintで受け取るだけだった

```
def experiment_gbdt(...  
    nfolds: int = 5,  
    ...)
```



例: nyagggle.experiment

Stratified KFold対応しとこ。lgb.cvにもあるし

```
def experiment_gbdt(...  
    nfolds: int = 5,  
    stratified: bool = False,  
    ...)
```



例: nyagggle.experiment

チームでSeed合わせることあるな→Seed追加

```
def experiment_gbdt(...  
    nfolds: int = 5,  
    stratified: bool = False,  
    seed_split: int = 0,  
    ...)
```



例: nyagggle.experiment

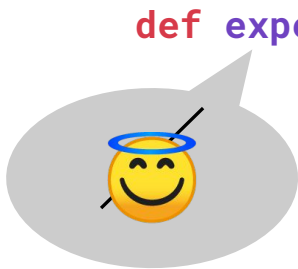
Group KFoldもいるよね

```
def experiment_gbdt(...  
    nfold: int = 5,  
    stratified: bool = False,  
    seed_split: int = 0,  
    groups: Optional[Series] = None,  
    ...)
```



例: nyagggle.experiment

あ、あと時系列でSplitもできないとね



```
def experiment_gbdt(...  
    nfold: int = 5,  
    stratified: bool = False,  
    seed_split: int = 0,  
    groups: Optional[Series] = None,  
    ...)
```



例: nyagggle.experiment

sklearnのcross_validateに倣って、cv引数にまとめる

- ・バリデーションの実装を分離することで柔軟に(関心の分離)
- ・実装もsklearnのcheck_cvという関数を呼ぶだけ

```
def experiment_gbdtd(...  
    cv: Union[int, BaseCrossValidator] = 5,  
    groups: Optional[Series] = None,  
    ...)
```



すぐ使えること



すぐ使えるまでの壁をなくす

環境構築の壁

- デフォルトの依存パッケージを減らす
- `pip install nyaggle`で入るようにしておく
- CIをちゃんと通す



すぐ使えるまでの壁をなくす

ライブラリ利用の壁

- なるべくsklearnなどの既知のAPIに寄せる
- APIドキュメントを書く
- “約束事”を最小限にする
 - API Docに“assuming that...”と書き始めたら黄信号
- 小さく始められるAPIにする
 - 例: mlflowはオプション扱い



心構え

- 自分の記憶力をあてにしない
 - LightGBMのPython APIページ、何回見た？
 - 未来の自分のため＝まだ見ぬユーザーのため

● すぐやめられること



すぐやめられること

- ブラックボックスなすごい仕組みにのっかると、後で降りるときに大変
- ダメだと思ったら使うのをいつでもやめたい



ではなく



を目指す



すぐやめられること

- UIは自作せず、他の仕掛けと連動
 - 他のものにも乗り換えても、過去の実験結果が引き継げる
- 各機能をシンプルに
- 高レベルAPIの下に低レベルAPIの層を作る
 - 高レベルAPIが使えないときに、低レベルAPIに降りる
 - Keras - TF, Lightning - PyTorchみたいなイメージ

で、使ってみてどうだったか



atmaCup#3でデビュー→準優勝

- 特徴量エンジニアリングに集中できた
 - "ちょっとCatBoost試そう"がノータイムでできた
 - Mlflow ui上で結果が見られる安心感
-
- 劇的な効果は無いけれど、それなりに役立った



分析用コードの整備は楽しい

- 求めるものは人それぞれ。正解は無い
 - 自作する時は自分が何を優先したいか、何を解決したいのか意識することをおすすめします
- (みんなのコンペ用パイプライン見るの楽しいので、みんな書いてみてほしい)



Thanks!

Source Code:

<https://github.com/nyanp/nyaggle>

Install:

```
> pip install nyaggle
```