

[ログイン](#)[新規登録](#)[トレンド](#)[質問](#)[◆ 100万記事感謝祭](#)[公式イベント](#)[公式コラム](#)[Organization](#)

i この記事は最終更新日から3年以上が経過しています。

📅 Kaggle Advent Calendar 2020



@chizuchizu

機械学習実験環境を晒す

Python データサイエンス Kaggle Hydra MLflow

最終更新日 2020年12月17日 投稿日 2020年12月15日

14日目はいのいさんの [【Kaggle】2020年に開催された画像分類コンペの1位の解法を紹介します](#)

です！

16日目は俵さんの [黒魔術への招待：Neural Network Stacking の探求](#) です！

Kaggleをやる私に必要なもの

こんにちは、皆さんはKaggleやってますか？ 正直なことをいうと、自分はあまりKaggleコンペに参加していないのでエアプ勢になってます。どちらかというとマイナーなコンペばかりに参加してて.....

Kaggle（広義）で勝つためにたくさんの実験を回しますが、何も考えていないと後々苦しみます

Qiitaにログインして、便利な機能を使ってみませんか？

♥ あなたにマッチした記事をお届けします

✓ 便利な情報をあとから読み返せます

[ログイン](#)

[新規登録](#)

[また後で](#)



248



217



諸々あってそろそろちゃんとした実験環境整えようかなという気になって自分のソースコードを整理してました。その頃にちょうどadvent calendarの時期になってたので折角の機会だしということで共有することにしました。

大事なこと↓

コード編

- 再現性
- デバッグしやすい
- 汎用性の高い関数

をデコレータで畳み掛ける！

おおまかなまとめ

- 乱数で実験を固定

これがないと何も始まらない。

- 再現したかったらgitのversionから実験名（乱数）を探す

ちゃんとcommitしようね。でも、面倒なので自動化しちゃおう。

- いつか必要になりそうな情報はちゃんと保存しておこうね

MLflowがあるよ

- コードの中には値を書かないでね

hydraを使ってconfigファイルを作ろう。実験管理がしやすくなるよ。

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録





Hydra | Hydra

<https://hydra.cc>

yamlファイルに保存されているパラメータを読み込み、pythonファイルで流し込むまでのプロセスをやってくれるライブラリです。

簡単な解説

1. yamlファイルを作る (config)
2. デコレータをつける

```
@hydra.main()
def main(cfg):
    return None
```

cfgは辞書型になるのであとは `cfg["parameter"]` と呼び出すだけです。

MLflow



Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248

217

MLflow | MLflow

<https://mlflow.org>

機械学習に関する実験管理ツールです。実験同士の比較も簡単なのでオススメです。

こちらは下記のリンクの解説がわかりやすいです。

[MLflow ～これで機械学習のモデル管理から API 作成まで楽にできるかも～](#)

GitPython

見たほうがわかりやすいです。コマンドラインで操作してたことをpythonコードでやるだけです。

[GitPythonを使う](#)

もちろん、`os.system()` のようなことでも代用可能です。

特徴量管理

takapyさんのスライド

データ分析コンペにおいて 特徴量管理に疲弊している全人類に伝えたい想い

<https://speakerdeck.com>

データ分析コンペにおいて
特徴量管理に疲弊している全人類に伝えたい想い
～学習・推論パイプラインを添えて～

ConneHito Inc. 野澤智朗
2019.11.05
ConneHito Marché vol.6 ～機械学習・データ分析市～

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248


217

2. 特徴量管理について

“列ごと”に特徴量をpickleファイルで管理する

survived_train.pkl pclass_train.pkl sex_train.pkl age_train.pkl embarked_train.pkl
pclass_test.pkl sex_test.pkl age_test.pkl embarked_test.pkl

Survived	Pclass	Sex	Age	Embarked
0	2	male	17	S
1	3	male	45	C
1	3	female	34	C
0	1	male	22	C
0	2	female	25	C
0	1	female	67	S
1	1	male	51	S

 mamari ママの一步を支える

takapyさんは下の記事を参考にしています。ここには具体的な実装例が書かれています。

[Kaggleで使えるFeather形式を利用した特徴量管理法](#)

自分はこれらの実装に少し手を加えてより使いやすいようにさせました。

列ごとの管理

使いたい特徴量を簡単に呼び出すことができます。また、列ごとにまとめておくことでメモリの効率化も図れます。（全部を呼び出してからカラム指定するのは非効率）

特徴量ごとにクラスを書くのですが、各々の値は干渉しないようになっているので管理も楽になってます。（バグが生まれにくい）

実践ottoコンペ

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



tree

軽く説明しておきます。

- data: 基本的にデータの保管庫
- config: hydraで読み込むためのyaml形式のconfig
- feature: あとで説明する。特徴量(.pkl)とその説明(.csv)が入る
- src: ソースコードを置く
- outputs: submitする用のファイルを置く

```
.
├── README.md
├── config # hydraで呼び出す
│   └── config.yaml # メインのconfig (ほかにもyamlを作ることも出来る)
├── data # dataset関連はここに集約
│   ├── sampleSubmission.csv
│   ├── test.csv
│   └── train.csv
├── features # 作った特徴量は列ごとにpickle形式で吐き出す
│   ├── _features_memo.csv # 特徴量のメモ
│   ├── base_data.pkl
│   └── pca.pkl
├── outputs # 提出用
│   ├── 118547.csv
│   ├── 736294.csv
│   └── 829643.csv
├── requirements.txt
├── src # ソース
│   ├── feature_engineering.py # 特徴量エンジニアリング
│   ├── inference.py # (Kaggle Notebookでの)推論用
│   ├── mlruns # MLflowのログ
│   │   ├── 0
│   │   ├── 1
│   │   ├── 2
│   │   ├── 3
│   │   └── 4
│   └── outputs # hydraのログ
```

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248

217

```
import hydra

@hydra.main(config_name="../config/config.yaml")
def main(cfg):
    run(cfg)
```

hydraを使うので、下のようにyaml形式でパラメータを書き、上のようにデコレータを置くことで cfg に辞書型のデータが渡されます。

```
../config/config.yaml

base:
  # 存在する特徴量だとしても上書きするか
  overwrite: true
  seed: 1234
  n_folds: 4
  # optunaを使うかどうか
  optuna: false
  num_boost_round: 1500

# LightGBMのパラメータ
parameters:
  objective: "multiclass"
  num_class: 9
  max_depth: 8
  learning_rate: .02
  metric: "multi_logloss"
  num_leaves: 31
  verbose: -1

# trainingに使用する特徴量
features: [
  "base_data",
  "pca",
]
```

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248

217

```
import hydra

@hydra.main()
def run():
    cwd = hydra.utils.get_original_cwd()
    data = pd.read_csv(cwd + "hogehoge.csv")
```

git関係

実行する前と実行したあとそれぞれのときにcommitします。

そうすることで、実行後に生成されるlogなどもgithubにアップロードをしたときにいつ生成されたファイルなのかわかるからです。（実行中に関係ないファイル操作をするとややこしくなりますが）

このデコレータは少し煩雑です。なぜなら、commit messageに含めるための実験名(rand)を引数として取る必要があるからです。

1. git_commits を呼ぶ
2. func_decorator() が呼ばれる
3. before running のcommitが行われる
4. func が実行される
5. after running のcommitが行われる
6. githubにpushされる

../utils.py

```
import git

def git_commits(rand):
    def func_decorator(my_func):
        print("experiment_name: ", rand)

    repo = git.Repo(str(Path(os.getcwd()).parents[0]))
```

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248

217


```
return decorator_wrapper
```

```
return func_decorator
```

万一datasetをpushしてしまったとき

エラーを吐かれます。変にいじっても更に複雑になって收拾がつかなくなるのでおとなしくしましょう。 `.gitignore` の設定をちゃんと忘れない事はもちろんですが、ことが起きたとしたら、安全なversionの番号をコピーして後ろに戻しましょう。そして問題のcommitを削除するなどして難を乗り越えましょう。

```
git reset "version number"
```

特徴量エンジニアリング

少し長くなりますが、具体例があるほうがわかりやすいと思うので紹介します。

お気持ちとしては、列ごとに管理してからconfigで使いたい列を指定して読み込むほうが実験や再現がしやすくなるよね〜というところです。

[Kaggleで使えるFeather形式を利用した特徴量管理法](#)

この天音さんの記事を参考にFeatureクラスを書きました。ベースとなるそのクラスは `utils.py` にあるのですが、あとで紹介します。（少し難解なので）

大事なポイントは

- クラス名が特徴量名となるクラスを作る
- `create_features()` 内の `self.data` を更新する
- `create_memo()` にその特徴量に関するメモを残す

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



```

from utils import Feature, generate_features, create_memo
from src.preprocess import base_data

import pandas as pd
import hydra
from sklearn.decomposition import PCA

# 生成された特徴量を保存するパス
Feature.dir = "features"
# trainとtestを結合して基本的な前処理を行ったデータと呼ぶ
data = base_data()

class Base_data(Feature):
    def create_features(self):
        self.data = data.drop(columns=["id"])
        create_memo("base_data", "初期")

class Pca(Feature):
    def create_features(self):
        n = 20
        pca = PCA(n_components=n)
        pca.fit(
            data.drop(
                columns=["train", "target", "id"]
            )
        )
        # カラム名
        n_name = [f"pca_{i}" for i in range(n)]
        df_pca = pd.DataFrame(
            pca.transform(data.drop(
                columns=["train", "target", "id"]
            )),
            columns=n_name
        )
        self.data = df_pca.copy()
        create_memo("pca", "pcaかけただけ")

```

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

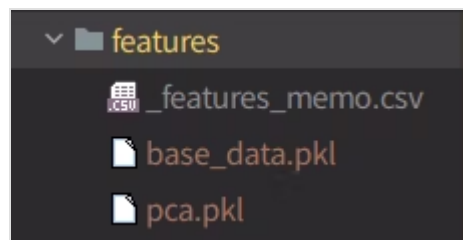
ログイン

新規登録



```
if __name__ == "__main__":  
    run()
```

実行すると feature の下に以下のようなファイルが生成されます。



また、_feature_memo.csv はgithub上で見ることもできます。 良い感じ。

2 lines (2 sloc)		40 Bytes
Search this file...		
1	base_data	初期
2	pca	pcaかけただけ

ちなみにですが、`cfg.base.overwrite` をtrueにすれば、既に実行し保存した特徴量も上書きするし、falseにすれば、保存されている特徴量は実行されません。
一回一回の計算が重いときは、基本falseにして、関数を上書きしたときには
`features/hoge.pkl` を削除してあげればfalseでも実行されるので良いと思います。

MLflow

今回の記事は書きたいことが盛り沢山なのでMLflow自体の解説は省きます。

そもそも論

実験はすべて乱数で管理しています。被ったときのことを考えてないのは内緒です。

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録

mlflowも色々なファイルを生成するのですが、hydraと併用しているとカレントディレクトリが変更されてしまうので上手くいきません。実行はされますが、mlflowが作成したファイル群はhydraのフォルダの中に保存されてしまって悲しくなります。

ちゃんと元あったパスを指定してあげましょう。

```
mlflow.set_tracking_uri("file://" + hydra.utils.get_original_cwd() + "/mlruns")
```

tracking

便利すぎる。これさえあればもう人間はのんびり暮らしていける。

```
mlflow.lightgbm.autolog()
```

- 実行したパラメータ
- best_iteration数
- 最終的なスコア
- パッケージ化された学習済みモデル
- importance

これらが勝手に保存されるようになっているので僕らは何もすることがありません。あと追加して保存したいものはコードに書きましょうというお気持ち。具体的にconfigや使った特徴量は保存したいよね。

```
cd src
mlflow ui
```

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



を実行して出てきたリンクを踏んで実験を選択すると下のようなページになります。

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248

217

fold_4 > 829643 ▾

Date: 2020-12-12 10:00:18

Source:  train.py

Git Commit:

5968c4ebd036a759d094be7940ec3d3a9baac706

User: yuma

Duration: 41.5s

Status: FINISHED

▼ Notes

None

▼ Parameters

Name	Value
categorical_feature	auto
early_stopping_rounds	100
feature_name	auto
keep_training_booster	False
learning_rate	0.02
max_depth	8
metric	multi_logloss
num_boost_round	1500
num_class	9
num_leaves	31
objective	multiclass
verbose	-1
verbose_eval	500

Qiitaにログインして、便利な機能を使ってみませんか？

♥ あなたにマッチした記事をお届けします

✔ 便利な情報をあとから読み返せます




ログイン

新規登録



248

217

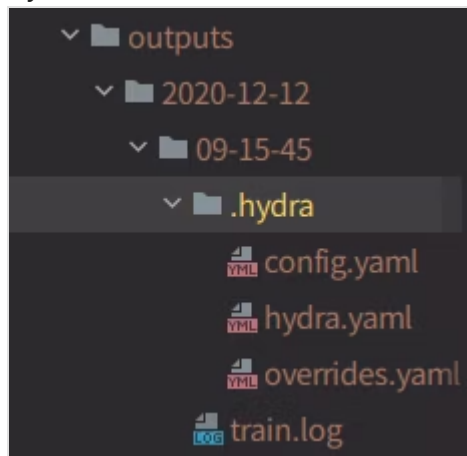
stopped_iteration 	1034
training-multi_logloss 	0.193
valid_1-multi_logloss 	0.483

少し改造しよう

追加で保存したいもの

- hydraの生成物(config、log等々)
- 特徴量
- submit用ファイル

hydraが生成するファイルは下のとおりです。



このとき、カレントディレクトリは 09-15-45 にあるので、以下のようにしてmlflowに保存します。最下行のコードは実行しているファイルから .py をとって .log をつけたログファイルを指定しています。

本音を言えばglobでこのディレクトリ下にあるファイルを全てぶち込むほうが綺麗に書けると思ったのですが、メモとして吐き出したとても大きなファイルがあった場合に困りそうだったので一つ一つ指定しました。

また、hydraを使っていれば以下のコードで示したようなyamlファイルやlogファイル

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録

```
def save_log(score_dict):
    mlflow.log_metrics(score_dict)
    mlflow.log_artifact(".hydra/config.yaml")
    mlflow.log_artifact(".hydra/hydra.yaml")
    mlflow.log_artifact(".hydra/overrides.yaml")
    # hydraでは実行したhoge.pyからhoge.logが生成されるのでそれも保存（少し煩雑）
    mlflow.log_artifact(f"{os.path.basename(__file__)[:-3]}.log")
    mlflow.log_artifact("features.csv") # 自分の場合は特徴量も吐き出しているの

score = {
    "rmse": 123, # 例
    "mae": 12, # 例
}

save_log(score)
```

次に特徴量です。念には念を入れて何らかの不手際があっても再現がとれなくなっても特徴量名から実装して何とかしようみたいなお気持ちです。

```
use_cols = pd.Series(train.columns)
use_cols.to_csv("features.csv", index=False, header=False)
mlflow.log_artifact("features.csv")
```

最後にsubmit用ファイルです。実験名（rand）をつけて保存することを心がけています。これをmlflowに結びつけておくことで mlflow ui 上でも変な予測値になってないかななど確認ができます。やっというて損はない。（と思ってる）

```
file_path = cwd / f"..../outputs/{rand}.csv"
ss.to_csv(file_path, index=False)
mlflow.log_artifact(file_path)
```

おまけ（log_artifact）

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録

で実行し、...、...を生成するようになっているかばかり気にする必要は...



248

217

結構駆け足なので何が便利なのかはちゃんと説明しておきます。もし要望があればottoコンペにnotebookをuploadしてそこでinferenceをするところまでやろうと思います。

便利なこと

- notebookにuploadする手間がなくなる
- inferenceに使うコードはconfigと学習済みモデルといった外部データに依存するので管理に頭を使う必要がない
- 何もせずともローカルで実行すれば 学習→終わる→upload→kaggle上で実行 までしてくれるのでちゃんと使えば効率化を図れる

Datasets

```
def add_datasets(rand):
    """upload to kaggle datasets
    hydraパス内で実行して
    notebooksの前に実行して
    """
    metadata = {
        "title": f"{rand}",
        "id": f"chizuchizu/{rand}",
        "licenses": [
            {
                "name": "CC0-1.0"
            }
        ]
    }

    data_json = eval(json.dumps(metadata))
    with open("dataset-metadata.json", "w") as f:
        json.dump(data_json, f)

    shutil.copy(".hydra/config.yaml", "config.yaml")
    os.system("kaggle datasets create -p .")
```

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



hydraパス内で実行して

```
:return: None
"""

meta = {
    "id": f"chizuchizu/{rand} inference",
    "title": f"{rand} inference",
    "language": "python",
    "kernel_type": "script",
    "code_file": str(cwd / "inference.py"),
    "is_private": "true",
    "enable_gpu": cfg.kaggle.enable_gpu,
    "dataset_sources": [
        f"chizuchizu/{rand}",
    ] + cfg.kaggle.data_sources,
    "competition_sources": cfg.kaggle.competitions,
}
data_json = eval(json.dumps(meta))
with open("kernel-metadata.json", "w") as f:
    json.dump(data_json, f)
os.system("kaggle kernels push -p .")
```

細かいところ

実験名をconfigに追記しておけばnotebooksでも読み込みが簡単になると思ったので実装してみました。

```
def add_experiment_name(rand):
    with open(".hydra/config.yaml", "r+") as f:
        data = yaml.load(f)

        data["experiment_name"] = str(rand)

        # f.write(yaml.dump(data))
    with open(".hydra/config.yaml", "w") as f:
        yaml.dump(data, f)
```

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録



```
add_experiment_name(rand=rand)
add_datasets(rand)
add_notebooks(rand, cwd, cfg)

return decorator_wrapper

return func_decorator
```

これから

技量を上げる

このようなまとめを書いたのはそれはそれで良くて、結局勝てなきゃエアプ勢にしかねないので2021年はコンペがんばります。

今cassavaコンペちょっとやろうとしてます。

クラウドでも動くように

どちらかというとクラウドでも作業ができるようになりたいというお気持ちです。最近の画像コンペはハイパワーGPUを必要とするので、家の8GBメモリのGPUだと力が弱すぎて.....

何にせよ、クラウドでも動くような（何の環境でも実験が可能な）コーディングを心がけていきたいです。

conda→Dockerへ

上のクラウドの話にも繋がりますが、今はanacondaでライブラリ等を管理していま

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録

より。



248

217



3



新規登録して、もっと便利にQiitaを使ってみよう

1. あなたにマッチした記事をお届けします
2. 便利な情報をあとで効率的に読み返せます
3. ダークテーマを利用できます

[ログインすると使える機能について](#)

新規登録

ログイン



@chizuchizu

フォロー



📌 今日のトレンド記事

📌 Qiita100万記事感謝祭！記事投稿キャンペーン開催のお知らせ



@uhyo

2025年01月19日

誤解されがちなnever型の危険性: 「存在しない」について

TypeScript

♡ 105



Qiitaにログインして、便利な機能を使ってみませんか？

- ♡ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録

Redis TypeScript Next.js Hono



248

217



@fumihiko_kimura

2025年01月18日

【完全網羅】第3回金融データ活用チャレンジ提出までの手引書

データ分析 AI Dataiku SIGNATE FDUA

♡ 33



@yossitech (yossi tech)

2025年01月18日

検索AIエージェント Feloがすごい

AI エージェント

♡ 34



🚩 Qiita100万記事感謝祭！記事投稿キャンペーン開催のお知らせ



@t-furusato (ふるさと@レアテック) in RareTECH 希少型エンジニア育成スクール

2025年01月20日

おい！そこのお前、俺の質問にちゃんと答えてくれ

ポエム 新人教育 新人プログラマ応援 新人エンジニア

♡ 19



トレンド一覧を見る

関連記事 Recommended by



argparseからhydraへの移植

by tanimutomo



mlflowを使ってデータ分析サイクルの効率化する方法を考える

by masa26hiro

Qiitaにログインして、便利な機能を使ってみませんか？

♡ あなたにマッチした記事をお届けします

✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248

217

電子帳簿保存法へとりあえずの対応をしていませんか？

PR 株式会社インテック

クレジットカード決済機能の実装ならPAY.JP

PR PAY株式会社

🔗 この記事は以下の記事からリンクされています

 **【Kaggle】2020年に開催された画像分類コンペの1位の解法を紹介します**
からリンク 4 years ago

コメント



@chizuchizu

2020-12-15 14:12 ...

gitのデコレータに不備があるので後で修正します（実行したあとのcommitが実行前に処理されてしまってる）



1



@yoshihomma (本間 喜明)

2020-12-16 20:08 ...

yamlファイルの読み込みですが、yacsというライブラリもおすすめです

<https://github.com/rbgirshick/yacs>

cfg.solver.base_lr みたいな感じで使えて、個人的には辞書型よりも扱いやすくて好きです。

Qiitaにログインして、便利な機能を使ってみませんか？

♥ あなたにマッチした記事をお届けします

✓ 便利な情報をあとから読み返せます

ログイン

新規登録



248

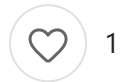
217

コメントありがとうございます！

一つ申し上げるとhydraのconfigのデータタイプは単純なdictではなくdictを拡張した
`omegaconf.dictconfig.DictConfig` という型なので `cfg.solver.base_lr` といった使い
方も出来ます！

yacsは初めて見ましたが、カレントディレクトリが変わらなかったりオーバーロード出
来たりなど優れてて良いですね！

今度使ってみます！



いいね以上の気持ちはコメントで

ログイン

新規登録

記事投稿キャンペーン開催中



Qiita100万記事感謝祭！記事投稿キャンペーン開催のお知らせ

2025/01/10~2025/01/31

詳細を見る

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✓ 便利な情報をあとから読み返せます

ログイン

新規登録

詳細を見る

すべて見る ➔

How developers code is here.

© 2011-2025 Qiita Inc.

ガイドとヘルプ

- About
- 利用規約
- プライバシーポリシー
- ガイドライン
- メディアキット
- ご意見・ご要望
- ヘルプ
- 広告掲載

コンテンツ

- リリースノート
- 公式イベント
- 公式コラム
- アドベントカレンダー
- Qiita 表彰プログラム
- エンジニア白書
- API

公式アカウント

- Qiita（キータ）公式
- Qiita マイルストーン
- Qiita 人気の投稿
- Facebook
- YouTube
- ポッドキャスト

Qiita 関連サービス

Qiita Team

運営

運営会社

Qiitaにログインして、便利な機能を使ってみませんか？

- ♥ あなたにマッチした記事をお届けします
- ✔ 便利な情報をあとから読み返せます

ログイン

新規登録

