

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. 野澤哲照

2019.11.05

Connehito Marché vol.6 ～機械学習・データ分析市～

こんにちは！

いきなりですが

データ分析コンペ（Kaggle , SIGNATEなど）
聞いたことある人～👋

データ分析コンペに参加したことある人～👋

**特徴量の管理ってどうしてますか？
(テーブルデータにおいて)**

よくあるパターン（実体験）

よくあるパターン (.ipynb)

- ・ 特徴量作る

`col = [y, A, B, C, D] → [y, A, B, C, D, E, F ...]`

```
[12] # e.g)
train['A'] = train['A'].fillna(0)
train['B'] = np.log1p(train['B'])
train['E'] = train['A'] + train['B']
df_group = train.groupby('D')['E'].mean()
train['F'] = train['D'].map(df_group)
```

・
・
・

よくあるパターン (.ipynb)

・
・
・

- ・ 使う特徴量のカラムだけ指定する

```
[93] # e.g)
      feat_col = ['A', 'C', 'D', 'E', 'F', 'J']
      x_train = train[feat_col]
      y_train = train['y']
```

- ・ 学習させる

```
[97] # e.g)
      clf.fit(x_train, y_train)
```

よくあるパターン (.ipynb)

・
・
・

- ・ 使う特徴量のカラムだけ指定する **あれ**

```
[93] # e.g)
      feat_col = ['A', 'C', 'D', 'E', 'F', 'G']
      x_train = train[feat_col]
      y_train = train['y']
```

‘F’ってどんな特徴量だっけ？

- ・ 学習させる

```
[97] # e.g)
      clf.fit(x_train, y_train)
```

よくあるパターン (.ipynb)

```
[12] # e.g)
train['A'] = train['A'].fillna(0)
train['B'] = np.log1p(train['B'])
train['E'] = train['A'] + train['B']
df_group = train.groupby('D')['E'].mean()
train['F'] = train['D'].map(df_group)
```



```
[93] # e.g)
feat_col = ['A', 'C', 'D', 'E', 'F', 'J']
x_train = train[feat_col]
y_train = train['y']
```

```
[97] # e.g)
clf.fit(x_train, y_train)
```

よくあるパターン (.ipynb)

```
[12] # e.g)
train['A'] = train['A'].fillna(0)
train['B'] = np.log1p(train['B'])
train['E'] = train['A'] + train['B']
df_group = train.groupby('D')['E'].mean()
train['F'] = train['D'].map(df_group)
```



見つけた！
(notebookの上の方)

```
[93] # e.g)
feat_col = ['A', 'C', 'D', 'E', 'F', 'J']
x_train = train[feat_col]
y_train = train['y']
```

```
[97] # e.g)
clf.fit(x_train, y_train)
```

よくあるパターン (.ipynb)

特徴量が少ない場合はまだマシだが、
多くなってくるとどんな計算で求めた
特徴量だったかをいちいち考える (探す)

のは結構大変だし、時間がかかる😓

```
[93] # e.g.)
      feat_col = 'D'
      x_train = train[feat_col]
      y_train = train['y']
```

```
[97] # e.g.)
      clf.fit(x_train, y_train)
```

よくあるパターン その2 (実体験)

よくあるパターン その2 (.ipynb)

よっしゃあ！めっちゃ良いスコアでたぜ~~~~😭😭😭

このnotebookをDuplicateして、もっと良いモデル作っちゃうぞ！



一方、notebookの中身は…

よくあるパターン その2 (.ipynb)

notebookの中身

```
[1] import numpy as np  
import pandas as pd
```

•
•
•

```
[193] submission.to_csv('submission.csv', index=False)
```

よくあるパターン その2 (.ipynb)

notebookの中身

```
[1] import numpy as np  
import pandas as pd
```

•

•

•

•

•

•

```
[193] submission.to_csv('submission.csv', index=False)
```

よくあるパターン その2 (.ipynb)

notebookの中身

```
[1] import numpy as np
import pandas as pd
```

・
・
・

```
[193]
```

```
submission.to_csv('submission.csv', index=False)
```



よくあるパターン その2 (.ipynb)

notebookの中身

同じ計算を何度もやらないといけない



よくあるパターン その2 (.ipynb)

度重なるDuplicateにより、notebook地獄に陥る可能性も…

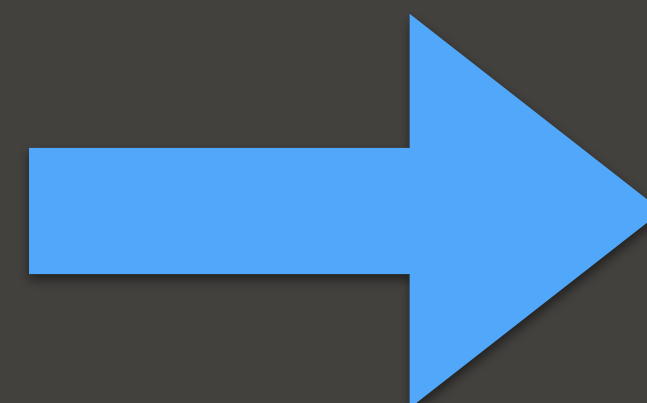
```
~/notebook/LightBGM_score_0902.ipynb  
~/notebook/LightBGM_score_0910.ipynb  
~/notebook/LightBGM_score_0911.ipynb  
~/notebook/LightBGM_score_0914.ipynb  
~/notebook/LightBGM_score_0916.ipynb  
~/notebook/LightBGM_score_0919.ipynb  
~/notebook/LightBGM_score_0924.ipynb  
~/notebook/LightBGM_score_0924.ipynb  
~/notebook/LightBGM_score_0924.ipynb  
~/notebook/LightBGM_score_0956.ipynb  
~/notebook/LightBGM_score_0966.ipynb  
~/notebook/LightBGM_score_0966.ipynb  
~/notebook/LightBGM_score_0966.ipynb  
.....
```



よくあるパターン その2 (.ipynb)

度重なるDuplicateにより、notebook地獄に陥る可能性も…

~/notebook/LightBGM_score_0902.ipynb
~/notebook/LightBGM_score_0910.ipynb
~/notebook/LightBGM_score_0911.ipynb
~/notebook/LightBGM_score_0914.ipynb
~/notebook/LightBGM_score_0916.ipynb
~/notebook/LightBGM_score_0917.ipynb
~/notebook/LightBGM_score_0918.ipynb
~/notebook/LightBGM_score_0924.ipynb
~/notebook/LightBGM_score_0924.ipynb
~/notebook/LightBGM_score_0956.ipynb
~/notebook/LightBGM_score_0966.ipynb
~/notebook/LightBGM_score_0966.ipynb
~/notebook/LightBGM_score_0966.ipynb
.....



今日話すこと

**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思い
～学習・推論パイプラインを添えて～**

アジェンダ

1. 自己紹介

2. 特徴量管理について

- ▶ 列ごとにpickleファイルで特徴量を管理
- ▶ 特徴量生成時、同時にメモファイルも生成

3. 学習・推論パイプラインについて

- ▶ コマンド一発で学習→Submitファイル作成までを実行
- ▶ 学習に使用した特徴量やモデルパラメータはlogと一緒に保存
- ▶ shapを用いて特徴量の貢献度を可視化し、次回学習時の勘所を見つける

アジェンダ

1. 自己紹介

2. 特徴量管理について

玄人の知恵をお借りしたら

めっちゃよかった^(※1) っていう話をします

3. 学習・推論パイプラインについて

(※1) あくまで主観です

- ▶ コマンド一発で学習→Submitファイル作成までを実行
- ▶ 学習に使用した特徴量やモデルパラメータはlogと一緒に保存
- ▶ shapを用いて特徴量の貢献度を可視化し、次回学習時の勘所を見つける

1. 自己紹介

1. 自己紹介

名前：野澤 哲照 (Nozawa Takanobu)

所属：コネヒト株式会社

 : たかぱい@takapy0210 



- ・ 2019.03～コネヒトにMLエンジニアとしてJOIN
- ・ 機械学習（NLP、推薦システム）をメインにやりつつインフラ（AWS）も勉強中
- ・ Kaggleしたり、ブログ（<https://www.takapy.work>）書いたり、野球したり、ラーメン食べたりしています

2. 特徴量管理について

- ▶ 列ごとにpickleファイルで特徴量を管理
- ▶ 特徴量生成時、同時にメモファイルも生成

※下記記事を参考にさせていただきました。

・ Kaggleで使えるFeather形式を利用した特徴量管理法

<https://amalog.hateblo.jp/entry/kaggle-feature-management>

最初にイメージを共有します

- ▶ 列ごとにpickleファイルで特徴量を管理
- ▶ 特徴量生成時、同時にメモファイルも生成

※下記記事を参考にさせていただきました。

・ Kaggleで使えるFeather形式を利用した特徴量管理法

<https://amalog.hateblo.jp/entry/kaggle-feature-management>

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

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	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

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
pclass_test.pkl

sex_train.pkl
sex_test.pkl

age_train.pkl
age_test.pkl

embarked_train.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

2. 特徴量管理について

特徴量生成時、同時に特徴量メモを作成する

2. 特徴量管理について

特徴量生成時、同時に特徴量メモを作成する

 takapy0210 mod

4c3b389 19 seconds ago

1 contributor

16 lines (15 sloc) 494 Bytes

Raw

Blame

History







Search this file...

1	特徴量	メモ
2	Survived	生存フラグ。今回の目的変数。
3	PassengerId	搭乗者ID。
4	Pclass	チケットのクラス。1st, 2nd, 3rdの3種類
5	Name	名前
6	Sex	性別
7	Age	年齢
8	Age_mis_val_median	年齢の欠損値を中央値で補完したもの
9	SibSp	兄弟/配偶者の数
10	Parch	親/子供の数
11	Ticket	チケットナンバー
12	Fare	運賃
13	Fare_mis_val_median	年齢の欠損値を中央値で補完したもの
14	Cabin	キャビン番号
15	Embarked	乗船した港

2. 特徴量管理について

特徴量生成時、同時に特徴量メモを作成する

16 lines (15 slots) 494 Bytes

Raw Blame History

検索

1	特徴量	
2	Survived	生存フラグ。今回の目的変数。
3	PassengerId	搭乗者ID。
4	Pclass	チケットのクラス。1st, 2nd, 3rdの3種類
5	Name	名前
6	Sex	性別
7	Age	年齢
8	Age_mis_val_median	年齢の欠損値を中央値で補完したもの
9	SibSp	兄弟/配偶者の数
10	Parch	親/子供の数
11	Ticket	チケットナンバー
12	Fare	運賃
13	Fare_mis_val_median	年齢の欠損値を中央値で補完したもの
14	Cabin	キャビン番号
15	Embarked	乗船した港

結構大変そう・・・🤔

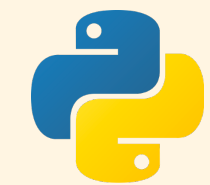
2. 特徴量管理について

特徴量生成時、同時に特徴量メモを作成する

でも、pythonスクリプトを1つ実行するだけ。

takapy0210 mod		4c3b389 19 seconds ago
1 contributor		
16 lines (15 sloc) 494 Bytes		Raw Blame History
特徴量	メモ	
2	Survived	生存フラグ。今回の目的変数。
3	PassengerId	搭乗者ID。
4	Pclass	チケットのクラス。1st, 2nd, 3rdの3種類
5	Name	名前
6	Sex	性別
7	Age	年齢
8	Age_mis_val_median	年齢の欠損値を中央値で補完したもの
9	SibSp	兄弟/配偶者の数
10	Parch	親/子供の数
11	Ticket	チケットナンバー
12	Fare	運賃
13	Fare_mis_val_median	年齢の欠損値を中央値で補完したもの
14	Cabin	キャビン番号
15	Embarked	乗船した港

2. 特徴量管理について



hoge.py をコマンドラインから実行するだけ

```
class Pclass(Feature):
    def create_features(self):
        self.train['Pclass'] = train['Pclass']
        self.test['Pclass'] = test['Pclass']
        create_memo('Pclass', 'チケットのクラス。1st, 2nd, 3rdの3種類')

class Sex(Feature):
    def create_features(self):
        self.train['Sex'] = train['Sex']
        self.test['Sex'] = test['Sex']
        create_memo('Sex', '性別')

class Age(Feature):
    def create_features(self):
        self.train['Age'] = train['Age']
        self.test['Age'] = test['Age']
        create_memo('Age', '年齢')

class Age_mis_val_median(Feature):
    def create_features(self):
        self.train['Age_mis_val_median'] = train['Age'].fillna(train['Age'].median())
        self.test['Age_mis_val_median'] = test['Age'].fillna(test['Age'].median())
        create_memo('Age_mis_val_median', '年齢の欠損値を中央値で補完したもの')
```


2. 特徴量管理について

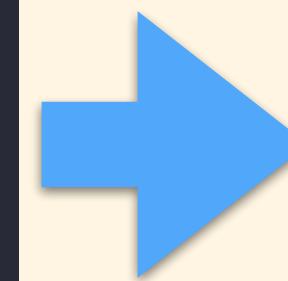
 hoge.py をコマンドラインから実行するだけ

```
class Pclass(Feature):
    def create_features(self):
        self.train['Pclass'] = train['Pclass']
        self.test['Pclass'] = test['Pclass']
        create_memo('Pclass', 'チケットのクラス。1st, 2nd, 3rdの3種類')

class Sex(Feature):
    def create_features(self):
        self.train['Sex'] = train['Sex']
        self.test['Sex'] = test['Sex']
        create_memo('Sex', '性別')

class Age(Feature):
    def create_features(self):
        self.train['Age'] = train['Age']
        self.test['Age'] = test['Age']
        create_memo('Age', '年齢')

class Age_mis_val_median(Feature):
    def create_features(self):
        self.train['Age_mis_val_median'] = train['Age'].fillna(train['Age'].median())
        self.test['Age_mis_val_median'] = test['Age'].fillna(test['Age'].median())
        create_memo('Age_mis_val_median', '年齢の欠損値を中央値で補完したもの')
```



_features_memo.csv

その他

- age_mis_val_median_test.pkl
- age_mis_val_median_train.pkl
- age_test.pkl
- age_train.pkl
- cabin_test.pkl
- cabin_train.pkl
- embarked_mis_val__s_test.pkl
- embarked_mis_val__s_train.pkl
- embarked_test.pkl
- embarked_train.pkl
- fare_mis_val_median_test.pkl
- fare_mis_val_median_train.pkl
- fare_test.pkl
- fare_train.pkl
- name_test.pkl
- name_train.pkl
- parch_test.pkl
- parch_train.pkl
- pclass_test.pkl
- pclass_train.pkl
- sex_test.pkl
- sex_train.pkl
- sib_sp_test.pkl
- sib_sp_train.pkl
- survived_test.pkl
- survived_train.pkl
- ticket_test.pkl
- ticket_train.pkl

2. 特徴量管理について

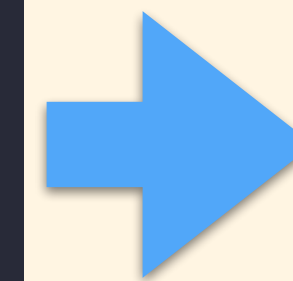
 hoge.py をコマンドラインから実行するだけ

```
class Pclass(Feature):  
    def create_features(self):  
        self.train['Pclass'] = train['Pclass']  
        self.test['Pclass'] = test['Pclass']  
        create_memo('Pclass', '客室のクラス。1st, 2nd, 3rdの3種類')
```

```
class Sex(Feature):  
    def create_features(self):  
        self.train['Sex'] = train['Sex']  
        self.test['Sex'] = test['Sex']  
        create_memo('Sex', '性別')
```

```
class Age(Feature):  
    def create_features(self):  
        self.train['Age'] = train['Age']  
        self.test['Age'] = test['Age']  
        create_memo('Age', '年齢')
```

```
class Age_mis_val_median(Feature):  
    def create_features(self):  
        self.train['Age_mis_val_median'] = train['Age'].fillna(train['Age'].median())  
        self.test['Age_mis_val_median'] = test['Age'].fillna(test['Age'].median())  
        create_memo('Age_mis_val_median', '年齢の欠損値を中央値で補完したもの')
```



_features_memo.csv

その他

- age_mis_val_median_test.pkl
- age_mis_val_median_train.pkl
- age_test.pkl
- age_train.pkl
- cabin_test.pkl
- cabin_train.pkl
- embarked_mis_val_s_test.pkl
- embarked_mis_val_s_train.pkl
- embarked_test.pkl
- embarked_train.pkl
- fare_mis_val_median_test.pkl
- fare_mis_val_median_train.pkl
- fare_test.pkl
- fare_train.pkl
- name_test.pkl
- name_train.pkl
- parch_test.pkl
- parch_train.pkl
- pclass_test.pkl
- pclass_train.pkl
- sex_test.pkl
- sex_train.pkl
- sib_sp_test.pkl
- sib_sp_train.pkl
- survived_test.pkl
- survived_train.pkl
- ticket_test.pkl
- ticket_train.pkl

各特徴量

2. 特徴量管理について

 hoge.py をコマンドラインから実行するだけ


```
class Pclass(Feature):
    def create_features(self):
        self.train['Pclass'] = train['Pclass']
        self.test['Pclass'] = test['Pclass']
        create_memo('Pclass', 'チケットのクラス。1st, 2nd, 3rdの3種類')

class Sex(Feature):
    def create_features(self):
        self.train['Sex'] = train['Sex']
        self.test['Sex'] = test['Sex']
        create_memo('Sex', '性別')





























class Age(Feature):
    def create_features(self):
        self.train['Age'] = train['Age']
        self.test['Age'] = test['Age']
        create_memo('Age', '年齢')

class Age_mis_val_median(Feature):
    def create_features(self):
        self.train['Age_mis_val_median'] = train['Age'].fillna(train['Age'].median())
        self.test['Age_mis_val_median'] = test['Age'].fillna(test['Age'].median())
        create_memo('Age_mis_val_median', '年齢の欠損値を中央値で補完したもの')
```

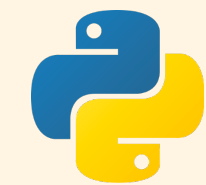
特徴量メモファイル

 _features_memo.csv

その他

-  age_mis_val_median_test.pkl
-  age_mis_val_median_train.pkl
-  age_test.pkl
-  age_train.pkl
-  cabin_test.pkl
-  cabin_train.pkl
-  embarked_mis_val__s_test.pkl
-  embarked_mis_val__s_train.pkl
-  embarked_test.pkl
-  embarked_train.pkl
-  fare_mis_val_median_test.pkl
-  fare_mis_val_median_train.pkl
-  fare_test.pkl
-  fare_train.pkl
-  name_test.pkl
-  name_train.pkl
-  parch_test.pkl
-  parch_train.pkl
-  pclass_test.pkl
-  pclass_train.pkl
-  sex_test.pkl
-  sex_train.pkl
-  sib_sp_test.pkl
-  sib_sp_train.pkl
-  survived_test.pkl
-  survived_train.pkl
-  ticket_test.pkl
-  ticket_train.pkl

2. 特徴量管理について



create_memoの処理概要

```
class Pclass(Feature):  
    def create_features(self):  
        self.train['Pclass'] = train['Pclass']  
        self.test['Pclass'] = test['Pclass']  
        create_memo('Pclass', 'チケットのクラス。1st, 2nd, 3rdの3種類')
```

2. 特徴量管理について

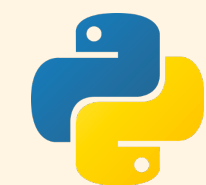
create_memoの処理概要

```
class Pclass(Feature):  
    def create_features(self):  
        self.train['Pclass'] = train['Pclass']  
        self.test['Pclass'] = test['Pclass']  
        create_memo('Pclass', 'チケットのクラス。1st, 2nd, 3rdの3種類')
```

特徴量メモcsvファイル作成

```
def create_memo(col_name, desc):  
  
    file_path = Feature.dir + '/_features_memo.csv'  
    if not os.path.isfile(file_path):  
        with open(file_path, "w"):pass  
  
    with open(file_path, 'r+') as f:  
        lines = f.readlines()  
        lines = [line.strip() for line in lines]  
  
    # 書き込もうとしている特徴量がすでに書き込まれていないかチェック  
    col = [line for line in lines if line.split(',')[0] == col_name]  
    if len(col) != 0: return  
  
    writer = csv.writer(f)  
    writer.writerow([col_name, desc])
```

2. 特徴量管理について



create_memoの処理概要

```
class Pclass(Feature):  
    def create_features(self):  
        self.train['Pclass'] = train['Pclass']  
        self.test['Pclass'] = test['Pclass']  
        create_memo('Pclass', 'チケットのクラス。1st,
```

特徴量メモcsvファイル作成

```
def create_memo(col_name, desc):  
  
    file_path = Feature.dir + '/_features'  
    if not os.path.isfile(file_path):  
        with open(file_path, "w"):pass  
  
    with open(file_path, 'r+') as f:  
        lines = f.readlines()  
        lines = [line.strip() for line in lines]  
  
    # 書き込もうとしている特徴量がすでに書き  
    col = [line for line in lines if  
           if len(col) != 0: return  
  
    writer = csv.writer(f)  
    writer.writerow([col_name, desc])
```

takapy0210 mod		4c3b389 19 seconds ago
1 contributor		
16 lines (15 sloc) 494 Bytes		Raw Blame History
Search this file...		
1	特徴量	メモ
2	Survived	生存フラグ。今回の目的変数。
3	PassengerId	搭乗者ID。
4	Pclass	チケットのクラス。1st, 2nd, 3rdの3種類
5	Name	名前
6	Sex	性別
7	Age	年齢
8	Age_mis_val_median	年齢の欠損値を中央値で補完したもの
9	SibSp	兄弟/配偶者の数
10	Parch	親/子供の数
11	Ticket	チケットナンバー
12	Fare	運賃
13	Fare_mis_val_median	年齢の欠損値を中央値で補完したもの
14	Cabin	キャビン番号
15	Embarked	乗船した港

CSV形式で保存しておくとGithubから参照しやすい (もちろん、ExcelやNumbersといったアプリケーションからでも綺麗に見える)

```
class Pclass(Feature):
    def create_features(self):
        self.train['Pclass'] = train['Pclass']
        self.test['Pclass'] = test['Pclass']
        create_memo('Pclass', 'チケットのクラス。1st, 2nd, 3rdの3種類')
```

特徴量メモcsvファイル作成

```
def create_memo(col_name, desc):
```

```
    file_path = Feature.dir + '/_features'
    if not os.path.isfile(file_path):
        with open(file_path, "w"):pass
```

```
    with open(file_path, 'r+') as f:
        lines = f.readlines()
        lines = [line.strip() for line in lines]
```

書き込もうとしている特徴量がすでに書き

```
    col = [line for line in lines if line.startswith(col_name)]
    if len(col) != 0: return
```

```
    writer = csv.writer(f)
    writer.writerow([col_name, desc])
```

 takapy0210 mod

4c3b389 19 seconds ago

1 contributor

16 lines (15 sloc) | 494 Bytes

Raw Blame History

Search this file...

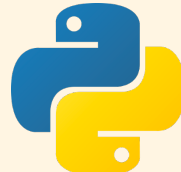
1	特徴量	メモ
2	Survived	生存フラグ。今回の目的変数。
3	PassengerId	搭乗者ID。
4	Pclass	チケットのクラス。1st, 2nd, 3rdの3種類
5	Name	名前
6	Sex	性別
7	Age	年齢
8	Age_mis_val_median	年齢の欠損値を中央値で補完したもの
9	SibSp	兄弟/配偶者の数
10	Parch	親/子供の数
11	Ticket	チケットナンバー
12	Fare	運賃
13	Fare_mis_val_median	年齢の欠損値を中央値で補完したもの
14	Cabin	キャビン番号
15	Embarked	乗船した港

2. 特徴量管理について

新しい特徴量を作成する場合

2. 特徴量管理について

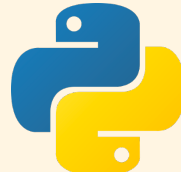
新しい特徴量を作成する場合

 hoge.py に新しい特徴量生成処理を記述

```
class Family_Size(Feature):  
    def create_features(self):  
        self.train['Family_Size'] = train['Parch'] + train['SibSp']  
        self.test['Family_Size'] = test['Parch'] + test['SibSp']  
        create_memo('Family_Size', '家族の総数')
```

2. 特徴量管理について

新しい特徴量を作成する場合

 hoge.py に新しい特徴量生成処理を記述

```
class Family_Size(Feature):  
    def create_features(self):  
        self.train['Family_Size'] = train['Parch'] + train['SibSp']  
        self.test['Family_Size'] = test['Parch'] + test['SibSp']  
        create_memo('Family_Size', '家族の総数')
```

>> python hoge.py

```
» python titanic_fe.py  
titanic main start  
titanic data load end  
survived was skipped  
passenger_id was skipped  
pclass was skipped  
name was skipped  
sex was skipped  
age was skipped  
age_mis_val_median was skipped  
sib_sp was skipped  
parch was skipped  
[family__size] start  
[family__size] done in 0 s  
ticket was skipped  
fare was skipped  
fare_mis_val_median was skipped  
cabin was skipped  
embarked was skipped  
embarked_mis_val__s was skipped
```


2. 特徴量管理について

新しい特徴量を作成する場合

 hoge.py に新しい特徴量生成処理を記述

```
class Family_Size(Feature):  
    def create_features(self):  
        self.train['Family_Size'] = train['Parch'] + train['SibSp']  
        self.test['Family_Size'] = test['Parch'] + test['SibSp']  
        create_memo('Family_Size', '家族の総数')
```

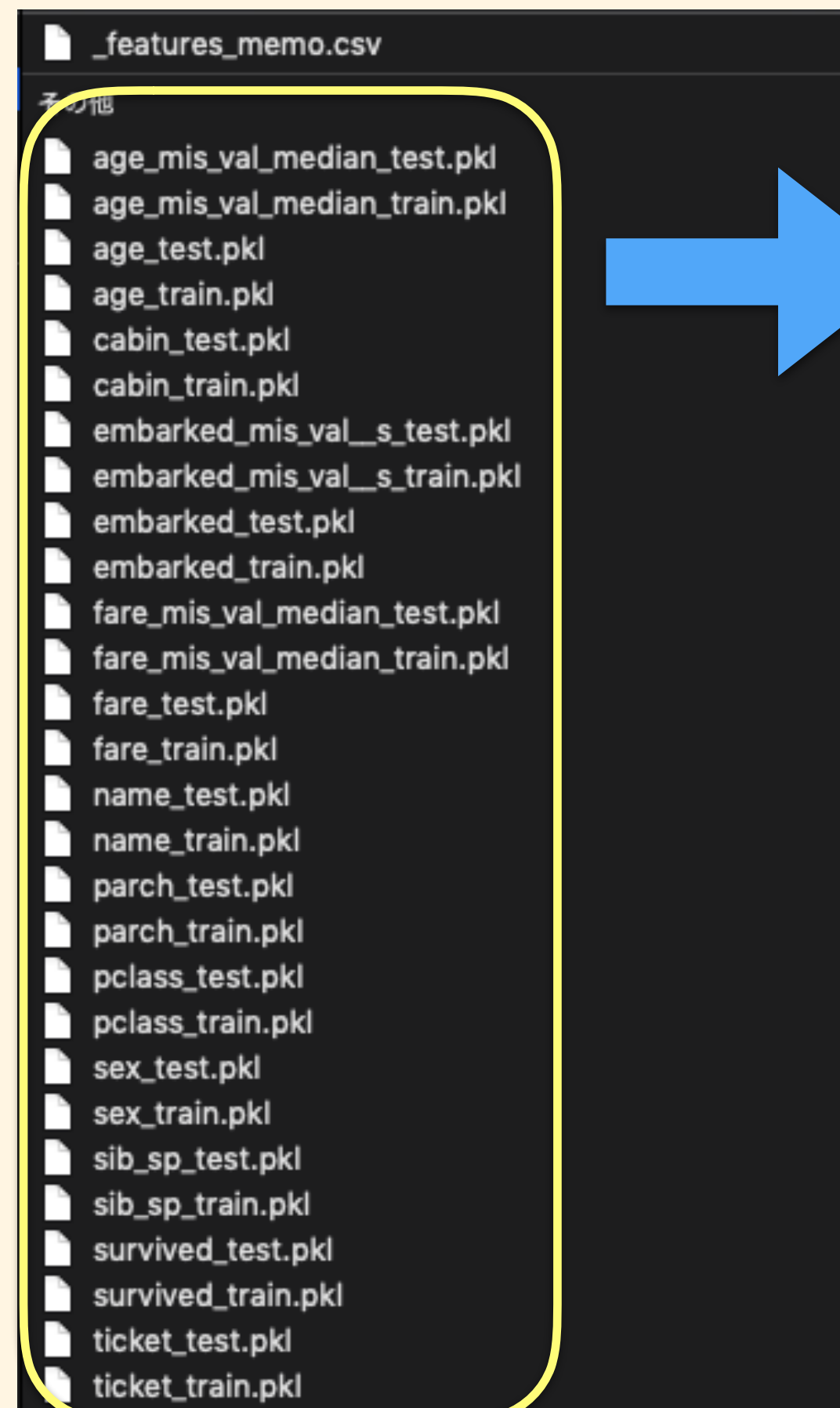
>> python hoge.py

```
» python titanic_fe.py  
titanic main start  
titanic data load end  
survived was skipped  
passenger_id was skipped  
pclass was skipped  
name was skipped  
sex was skipped  
age was skipped  
age_mis_val_median was skipped  
sib_sp was skipped  
parch was skipped  
[family_size] start  
[family_size] done in 0 s  
ticket was skipped  
fare was skipped  
fare_mis_val_median was skipped  
cabin was skipped  
embarked was skipped  
embarked_mis_val_s was skipped
```

新しい特徴量のみ生成

2. 特徴量管理について

データを読み込む際は、特徴量を指定してロードするだけ



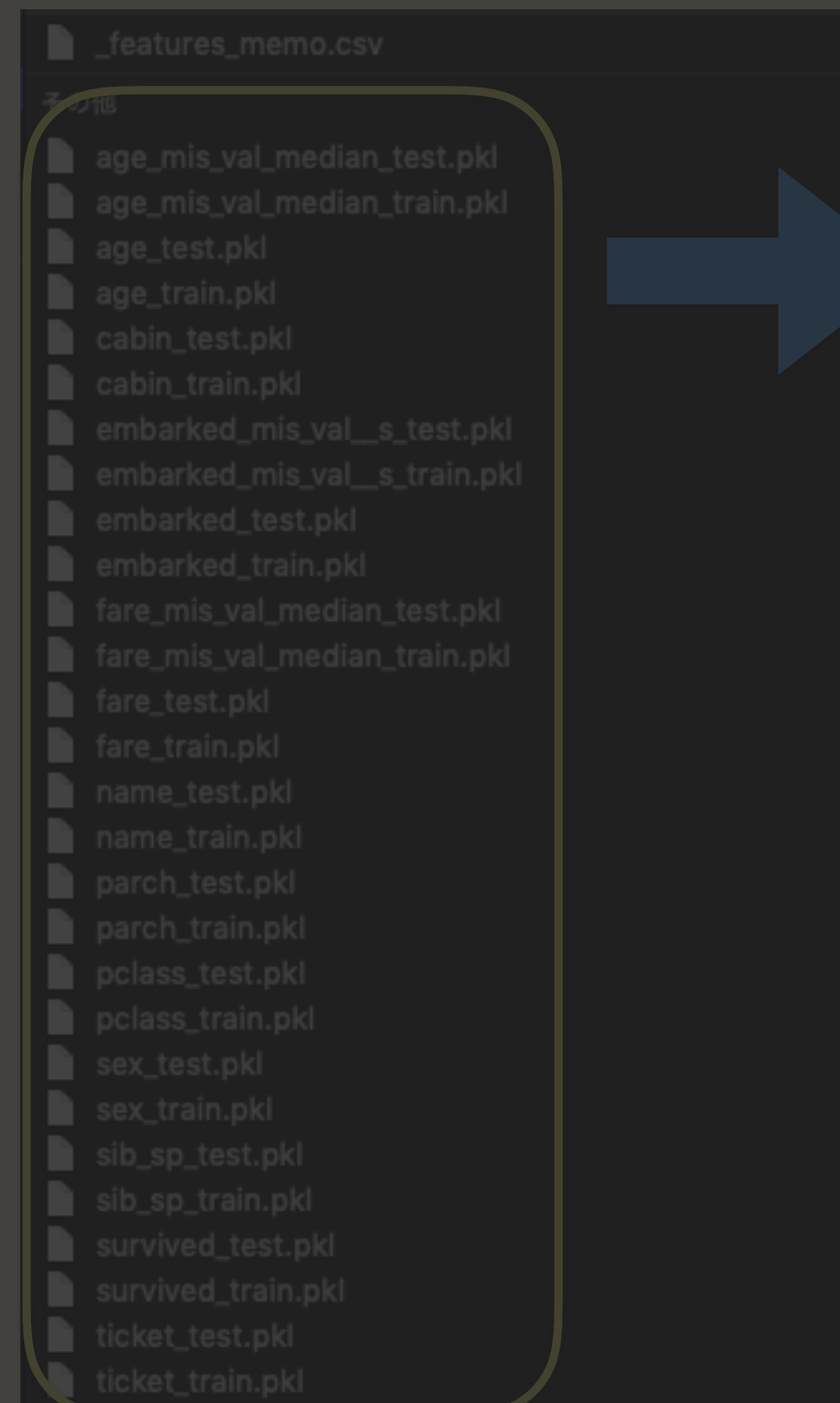
特徴量の指定

```
features = [  
    "age_mis_val_median",  
    "family__size",  
    "cabin",  
    "fare_mis_val_median"  
]
```

```
df = [pd.read_pickle(FEATURE_DIR_NAME + f'{f}_train.pkl') for f in features]  
df = pd.concat(df, axis=1)
```

2. 特徴量管理について

データを読み込む際は、特徴量を指定してロードするだけ



何が嬉しかったか

```
# 特徴量の指定
features = [
    "age_mis_val_median",
    "family_size",
    "cabin",
    "fare_mis_val_median"
]

df = [pd.read_pickle(FEATURE_DIR_NAME + f'{f}_train.pkl') for f in features]
df = pd.concat(df, axis=1)
```

2. 特徴量管理について

- ・ 1つのスクリプトファイルに特徴量生成をまとめることで、同じ計算を複数回実行することを避け、時間を有効活用できる。
→特徴量の再現性も担保。
- ・ 特徴量のメモを同時に生成することで「この特徴量なんだっけ？」と頭を使わずに済んだ。
- ・ 特徴量を列ごとに管理することで取り回しが楽になる。
→pickleファイルだと保存も読み込みも速い！
→特徴量が膨大になる場合は、ある程度の単位でまとめて管理する方が良いかも。

3. 学習・推論パイプラインについて

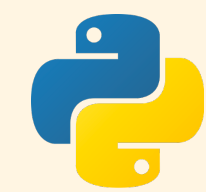
- ▶ コマンド一発で学習→Submitファイル作成までを実行
- ▶ 学習に使用した特徴量やモデルパラメータはlogと一緒に保存
- ▶ shapを用いて特徴量の貢献度を可視化し、次回学習時の勘所を見つける

※下記書籍に掲載されているパイプラインを参考にしました。

・Kaggleで勝つデータ分析の技術

<https://qihyo.jp/book/2019/978-4-297-10843-4>

3. 学習・推論パイプラインについて



run.py を実行することで、学習・推論・Submitファイルを作成

```
# 特徴量の指定
features = [
    "age_mis_val_median",
    "family_size",
    "cabin",
    "fare_mis_val_median"
]

run_name = 'lgb_1102'

# 使用する特徴量リストの保存
with open(LOG_DIR_NAME + run_name + "_features.txt", 'wt') as f:
    for ele in features:
        f.write(ele+'\n')

params_lgb = {
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'early_stopping_rounds': 20,
    'verbose': 10,
    'random_state': 99,
    'num_round': 100
}

# 使用するパラメータの保存
with open(LOG_DIR_NAME + run_name + "_param.txt", 'wt') as f:
    for key,value in sorted(params_lgb.items()):
        f.write(f'{key}:{value}\n')

runner = Runner(run_name, ModelLGB, features, params_lgb, n_fold, name_prefix)
runner.run_train_cv() # 学習
runner.run_predict_cv() # 推論
Submission.create_submission(run_name) # submit作成
```

3. 学習・推論パイプラインについて

 run.py を実行することで、学習・推論・Submitファイルを作成

このrun_nameをprefixとして、ファイルやモデルを保存してくれる。

```
# 特徴量の指定
features = [
    "age_mis_val_median",
    "family_size",
    "cabin",
    "fare_mis_val_median"
]
```

```
run_name = 'lgb_1102'
```

例

- ・ 使用した特徴量リスト
- ・ 使用したハイパーパラメータ
- ・ fold毎のモデル
- ・ 推論結果
- ・ submitファイル
- ・ shapの計算結果イメージファイル など

```
# 使用する特徴量リストの保存
with open(LOG_DIR_NAME + run_name + "_features.txt", 'wt') as f:
    for ele in features:
        f.write(ele+'\n')
```

```
params_lgb = {
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'early_stopping_rounds': 20,
    'verbose': 10,
    'random_state': 99,
    'num_round': 100
}
```

```
# 使用するパラメータの保存
```

```
with open(LOG_DIR_NAME + run_name + "_param.txt", 'wt') as f:
    for key,value in sorted(params_lgb.items()):
        f.write(f'{key}:{value}\n')
```

```
runner = Runner(run_name, ModelLGB, features, params_lgb, n_fold, name_prefix)
```

```
runner.run_train_cv() # 学習
```

```
runner.run_predict_cv() # 推論
```

```
Submission.create_submission(run_name) # submit作成
```

3. 学習・推論パイプラインについて

 run.py を実行することで、学習・推論・Submitファイルを作成

生成されるファイル例

```
# 特徴量の指定
features = [
    "age_mis_val_median",
    "family__size",
    "cabin",
    "fare_mis_val_median"
]

run_name = 'lgb_1102'

# 使用する特徴量リストの保存
with open(LOG_DIR_NAME + run_name + "_features.txt", 'wt') as f:
    for ele in features:
        f.write(ele+'\n')

params_lgb = {
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'early_stopping_rounds': 20,
    'verbose': 10,
    'random_state': 99,
    'num_round': 100
}

# 使用するパラメータの保存
with open(LOG_DIR_NAME + run_name + "_param.txt", 'wt') as f:
    for key,value in sorted(params_lgb.items()):
        f.write(f'{key}:{value}\n')

runner = Runner(run_name, ModelLGB, features, params_lgb, n_fold, name_prefix)
runner.run_train_cv() # 学習
runner.run_predict_cv() # 推論
Submission.create_submission(run_name) # submit作成
```


3. 学習・推論パイプラインについて

生成されるファイルの例（フォルダは適宜分けています）

- ・ lgb_1102_01-fold0.model（fold-0で作成されたモデル）
- ・ lgb_1102_01-fold1.model（fold-1で作成されたモデル）
- ・ lgb_1102_01-fold2.model（fold-2で作成されたモデル）
- ・ lgb_1102_01-pred.pkl（testデータでの推論結果）
- ・ lgb_1102_01_submission.csv（推論結果をkaggleに提出できるcsvに変換したもの）
- ・ lgb_1102_01_features.txt（今回の学習に使用した特徴量リスト）
- ・ lgb_1102_01_param.txt（今回の学習に使用したハイパーパラメータ）
- ・ lgb_1102_01_shap.png（shapで計算した可視化イメージ）
- ・ general.log（計算ログファイル）
- ・ result.log（モデルのスコアだけが記載されたログファイル）

3. 学習・推論パイプラインについて

生成されるファイルの例（フォルダは適宜分けています）

- ・ lgb_1102_01-fold0.m
 - ・ lgb_1102_01-fold1.m
 - ・ lgb_1102_01-fold2.m
 - ・ lgb_1102_01-pred.pk
 - ・ lgb_1102_01_submis
- ```
1 age_mis_val_median
2 family__size
3 fare_mis_val_median
4 sib_sp
5 parch
```
- （このファイルは、lgb\_1102\_01\_submis.csvから変換したものです）

- ・ lgb\_1102\_01\_features.txt（今回の学習に使用した特徴量リスト）
- ・ lgb\_1102\_01\_param.txt（今回の学習に使用したハイパーパラメータ）
- ・ lgb\_1102\_01\_shap.png（shapで計算した可視化イメージ）
- ・ general.log（計算ログファイル）
- ・ result.log（モデルのスコアだけが記載されたログファイル）

### 3. 学習・推論パイプラインについて

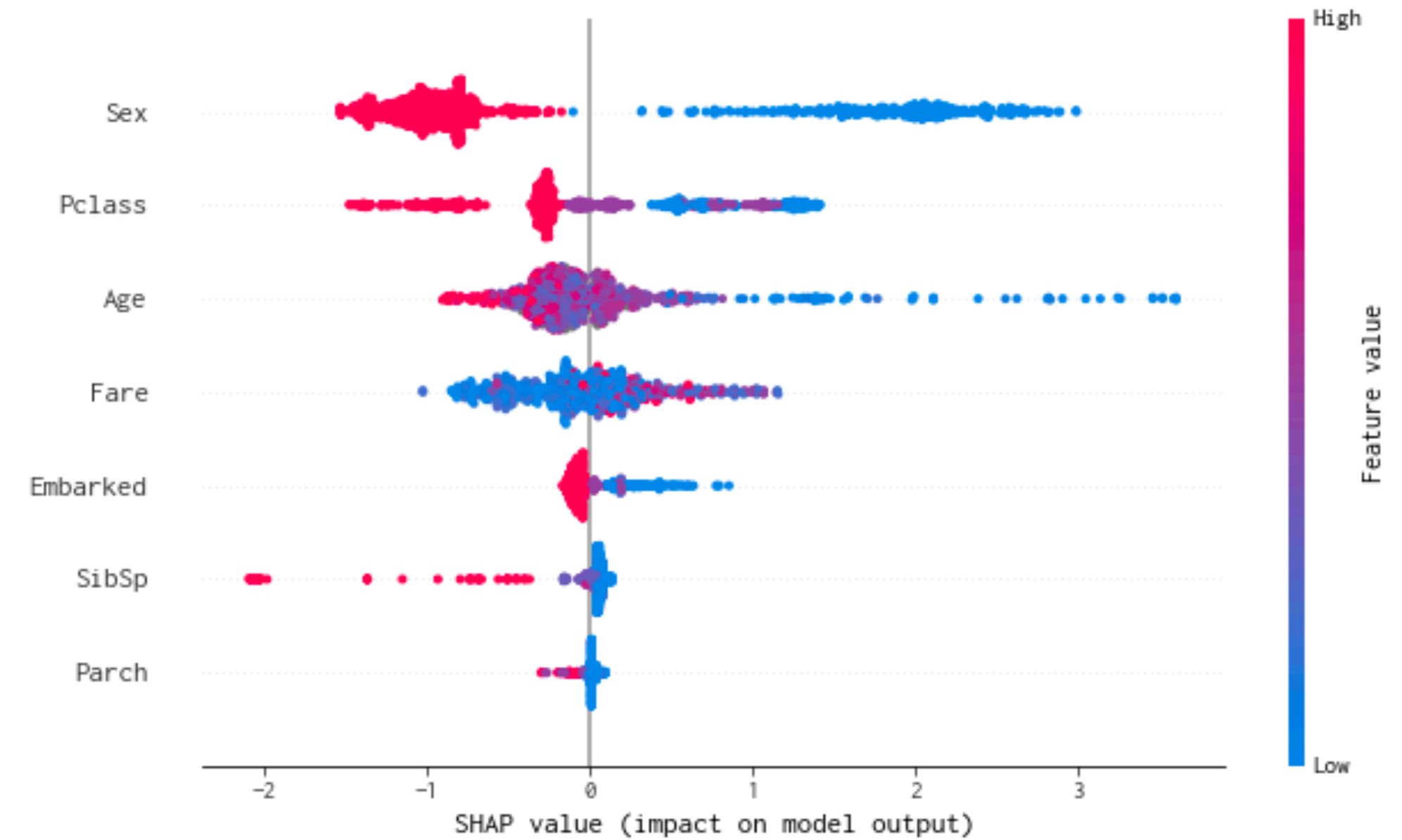
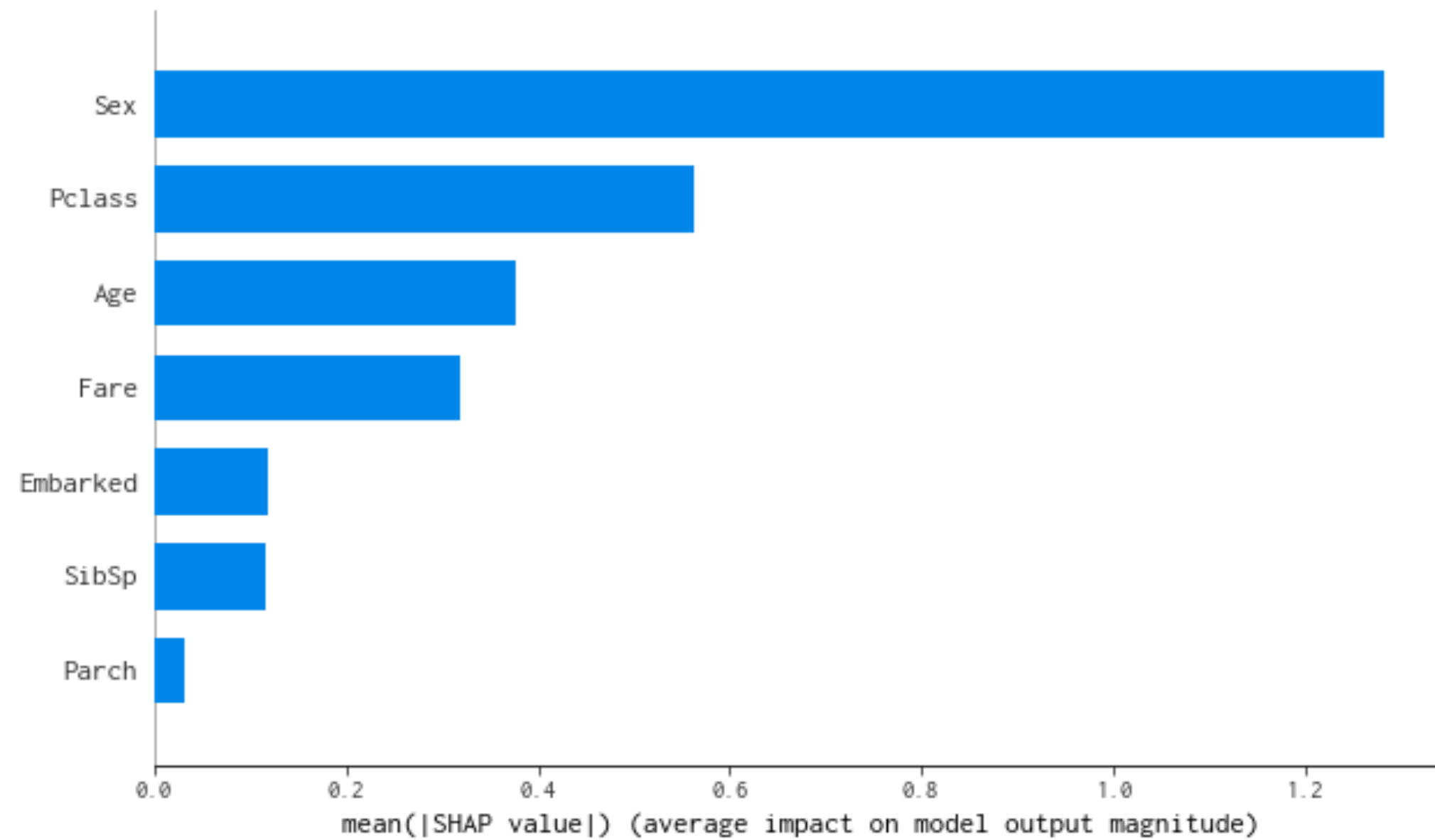
生成されるファイルの例

- ・ lgb\_1102\_01-fold0.model
- ・ lgb\_1102\_01-fold1.model
- ・ lgb\_1102\_01-fold2.model
- ・ lgb\_1102\_01-pred.pkl (pickle形式で保存した予測結果)
- ・ lgb\_1102\_01\_submission.csv (提出できるcsvに変換したもの)
- ・ lgb\_1102\_01\_features.txt (特徴量リスト)

```
1 boosting_type:gbdt
2 early_stopping_rounds:200
3 feature_fraction:0.9
4 lambda_l1:1
5 lambda_l2:1
6 learning_rate:0.1
7 num_depth:7
8 num_leaves:30
9 num_round:1000
10 objective:regression
11 subsample:0.25
12 verbose:100
```

- ・ lgb\_1102\_01\_param.txt (今回の学習に使用したハイパーパラメータ)
- ・ lgb\_1102\_01\_shap.png (shapで計算した可視化イメージ)
- ・ general.log (計算ログファイル)
- ・ result.log (モデルのスコアだけが記載されたログファイル)

# 3. 学習・推論パイプラインについて



・ lgb\_1102\_01\_param.txt (ラウンドの学習に使用したハイパーパラメータ)

・ lgb\_1102\_01\_shap.png (shapで計算した可視化イメージ)

・ general.log (計算ログファイル)

・ result.log (モデルのスコアだけが記載されたログファイル)



### 3. 学習・推論パイプラインについて

```
general.log の例
[2019-11-02 15:40:25] - lgb_1102_01 - start training cv
[2019-11-02 15:40:25] - lgb_1102_01 fold 0 - start training
[2019-11-02 15:44:50] - lgb_1102_01 fold 0 - end training - score 1.0354044974485535
[2019-11-02 15:44:50] - lgb_1102_01 fold 1 - start training
[2019-11-02 15:49:08] - lgb_1102_01 fold 1 - end training - score 1.0349502578930894
[2019-11-02 15:49:08] - lgb_1102_01 fold 2 - start training
[2019-11-02 15:53:26] - lgb_1102_01 fold 2 - end training - score 1.030552438453732
[2019-11-02 15:53:27] - lgb_1102_01 - end training cv - score 1.033635731265125
[2019-11-02 15:55:27] - lgb_1102_01 - start prediction cv
[2019-11-02 16:01:41] - lgb_1102_01 - end prediction cv
[2019-11-02 16:02:43] - lgb_1102_01 - start create submission
[2019-11-02 16:04:47] - lgb_1102_01 - end create submission
```

```
result.log の例
name:lgb_1102_01 score:1.033635731265125 score0:1.0354044974485535 score1:1.0349502578930894 score2:1.030552438453732
```

- ・ general.log (計算ログファイル)
- ・ result.log (モデルのスコアだけが記載されたログファイル)

# 3. 学習・推論パイプラインについて

```
general.log の例
[2019-11-02 15:40:25] - lgb_1102_01 - start training cv
[2019-11-02 15:40:25] - lgb_1102_01 fold 0 - start training
[2019-11-02 15:44:50] - lgb_1102_01 fold 0 - end training - score 1.0354044974485535
[2019-11-02 15:44:50] - lgb_1102_01 fold 1 - start training
[2019-11-02 15:49:08] - lgb_1102_01 fold 1 - end training - score 1.0349502578930894
[2019-11-02 15:49:08] - lgb_1102_01 fold 2 - start training
[2019-11-02 15:53:26] - lgb_1102_01 fold 2 - end training - score 1.030552438453732
[2019-11-02 15:53:27] - lgb_1102_01 - end training cv - score 1.033635731265125
[2019-11-02 15:55:27] - lgb_1102_01 - start prediction cv
[2019-11-02 16:01:41] - lgb_1102_01 - end prediction cv
[2019-11-02 16:02:43] - lgb_1102_01 - start create submission
[2019-11-02 16:04:47] - lgb_1102_01 - end create submission
```

何が嬉しかったか

(cvに変換したもの)

```
result.log の例
name:lgb_1102_01 score:1.033635731265125 score0:1.0354044974485535 score1:1.0349502578930894 score2:1.030552438453732
lgb_1102_01_snap.png (Snapで計算した可視化イメージ)
```

- ・ general.log (計算ログファイル)
- ・ result.log (モデルのスコアだけが記載されたログファイル)



### 3. 学習・推論パイプラインについて

- ・ 「この特徴量」と「このパラメータ」を使って学習させたモデルに関して、「各タスクに要した時間」と「各fold+最終的なスコア」を意識しなくても管理できるように。
- ・ shapの計算結果やfeature importanceを出力しておくことで、次の学習時の勘所が掴めるように。

# まとめ

# まとめ

- ・ **特徴量管理いいぞ！**

- 1つのスクリプトファイルに特徴量生成をまとめることで、同じ計算を複数回実行することを回避できる！
- 特徴量のメモを同時に生成することで「この特徴量なんだっけ？」と頭を使う回数が減る！
- 特徴量を列ごとに管理することで取り回しが楽になった！（が、特徴量が膨大な場合はある程度のまとまりで管理した方が良さかも）

- ・ **パイプラインいいぞ！**

- パイプラインを構築することで、高速なPDCAを実現！
- 学習に使用した特徴量とパラメータを管理することで、再現性も担保され心理的安全性も

++😊

ご清聴ありがとうございました🙇