Kaggle Titanic 入門 02

スコア向上策

庭師

庭子アバター v01.50
Avatar Museum12に展示中

0.73444 だった。gender_submission.csvよりもスコアが低いなぜ？

- **スコアが 0.73444 で、gender_submission.csv（おそらく 0.76555 程度）より低い理由は、主に以下のような要因が考えられます：**

✅ `gender_submission.csv` は「女性は生存、男性は死亡」のルールベース予測

- 非常にシンプルですが、Titanicデータでは **女性の生存率が高く、男性の生存率が低い** ため、このルールだけでも **そこそこ高い精度** を出せます。
- 実際、Kaggle公式のベースラインとしても提示されています。

🔍 あなたの `RandomForest` モデルのスコアが低い理由

1. 過学習 or 過剰な特徴量

- `Title`, `FareBin`, `AgeBin`, `IsAlone` などの特徴量を多数導入しましたが、適切に効いていない or ノイズになっている可能性があります。

2. 欠損値補完の方法が単純

- 年齢や運賃などを「中央値」や「最頻値」で一律補完していますが、より賢い方法（例: `Sex` や `Pclass` を使ったグループ中央値補完）だと精度が上がります。

3. 特徴量のスケーリングや調整がされていない

- RandomForestはスケーリングに強いですが、`AgeBin` や `FareBin` などカテゴリ化が精度を落としている可能性も。

4. パラメータチューニング未実施

- `n_estimators`, `max_depth`, `min_samples_split` などのチューニングが行われていません。
- デフォルト設定のままでは精度が出ないケースも多いです。

5. 学習データとテストデータで分布がズレている

- 特に `FareBin` や `Title` などがテスト側に未出現カテゴリを含んでいた可能性も。

✅ 改善案（提案）

1. **シンプルな特徴量に絞る**
   - `Pclass`, `Sex`, `Age`, `Fare` などだけで
2. **モデルをチューニング**
   - `RandomForest` のパラメータ調整
   - または `LightGBM` に変更（高性能）
3. **クロスバリデーションで検証**
   - `train.csv` 内で交差検証して精度を確認
   - Public LB を基準にせず過学習を防ぐ

**前回はここまでで
でした！**

この続きです！

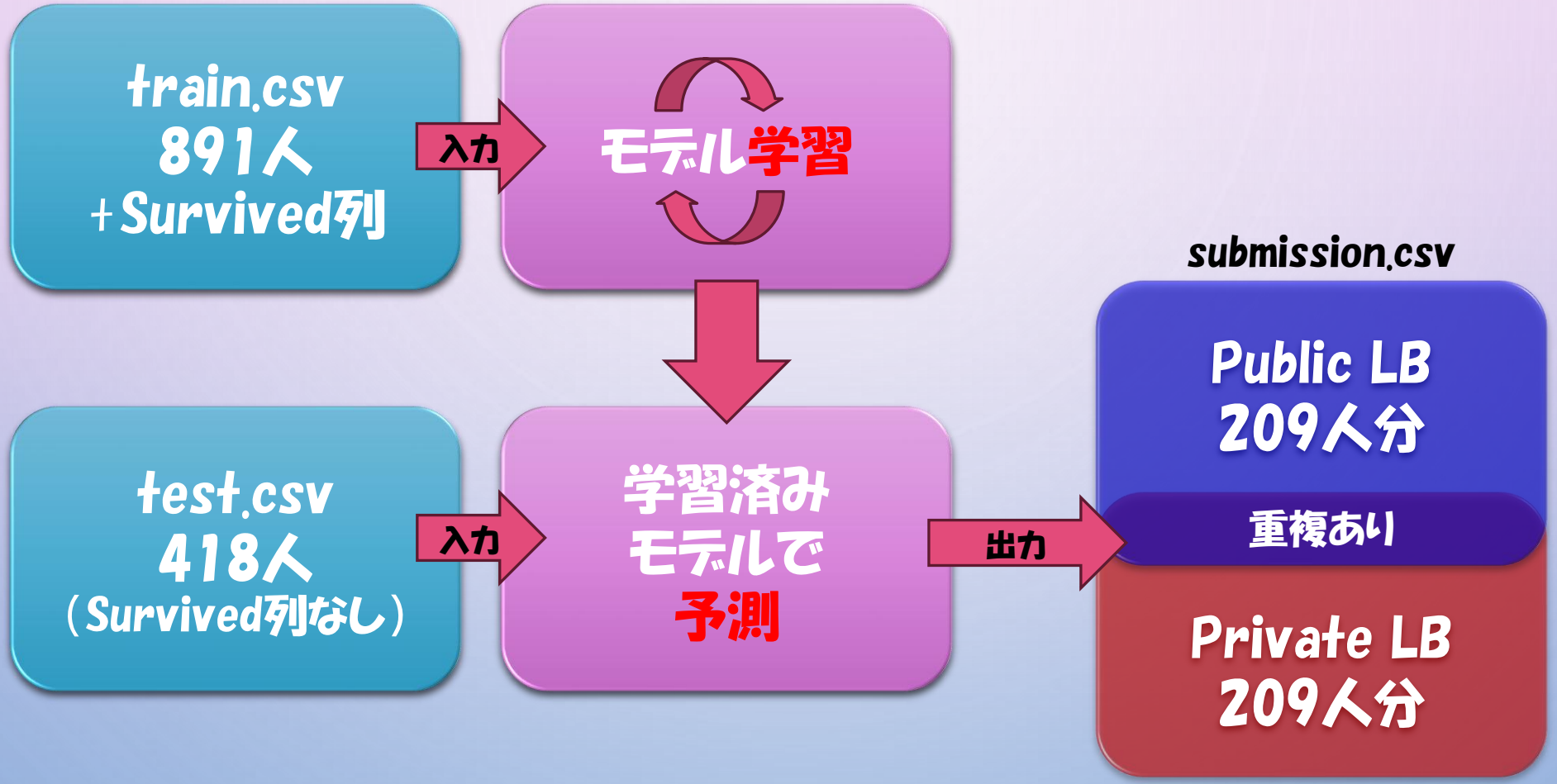
🔚 結論

`gender_submission.csv` は実は「データに対して最適なバイアス的ルール」に基づいていて、意外と強いベースラインです。
一方でモデルを使う場合は**適切な特徴量選択・欠損処理・チューニング**が必要です。

希望があれば、「精度を gender_submission.csv 以上に上げるための改良案」をコード付きで出せます。どうしますか？

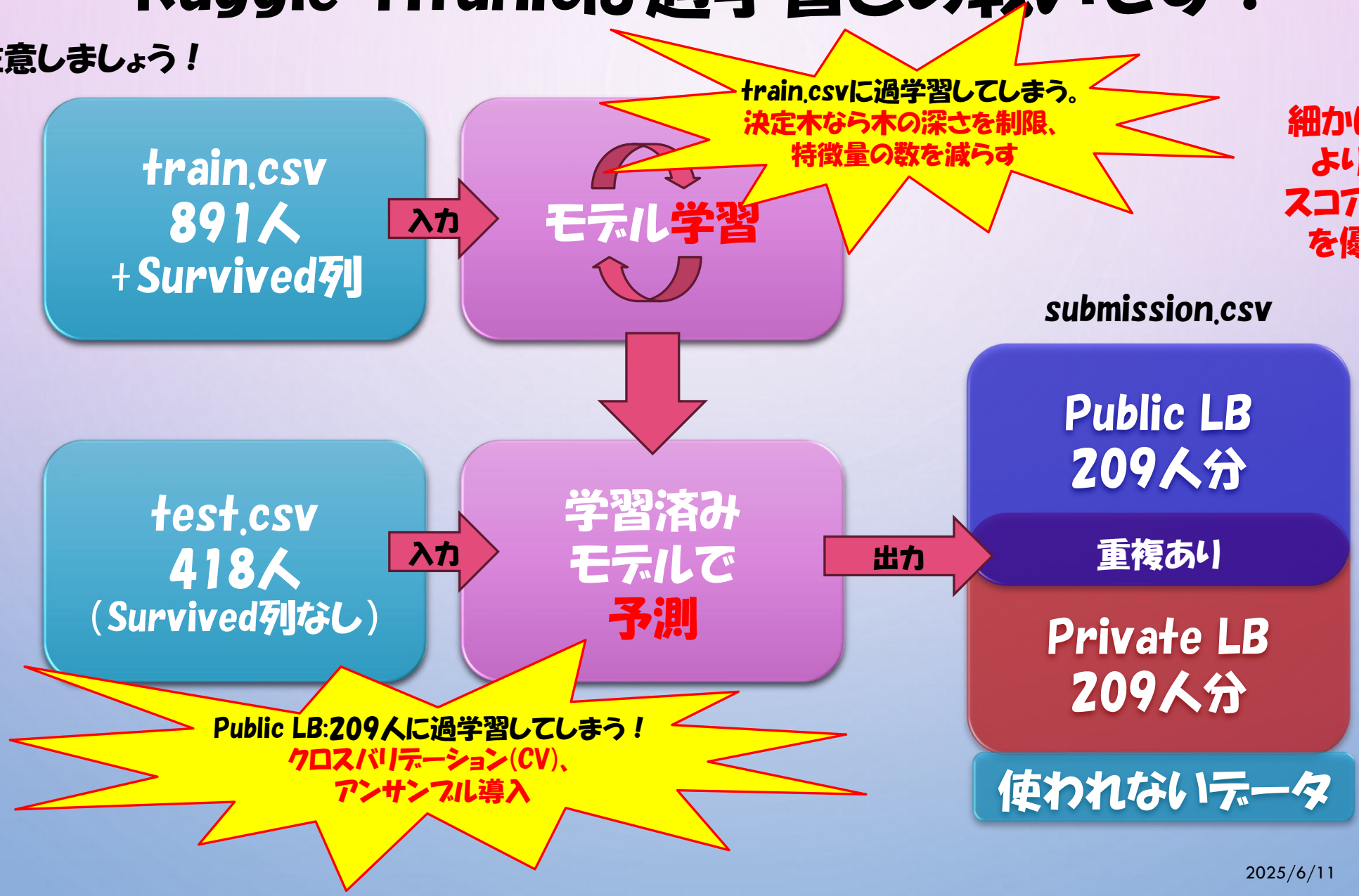
続く！！

train, test, Public LB, Private LB の関係



Kaggle Titanicは過学習との戦いです！

2カ所、注意しましょう！



戦略：まずスコアを安定させる！汎化対策！

- ChatGPTでtitanic予測させると、0.73～0.76くらいで大きくばらつきます。

【今回のシナリオ】

- さらに過学習(over fitting)のリスクあるので、細かい特徴量調整やるまえに安定化・再現性をあげます。
  - random_state 固定 (random_state=42 はよく使います。他の数字でもいい)
- その後汎用的なスコア向上策をいれます。
  - クロスバリデーション(CV)は早めに導入する。汎化対策。
  - 複数モデルの結果を集約するときは、平均より多数決(mode)が良さそう
- 段階的に特徴量エンジニアリングしていく

プログラムコード部は文字が小さいので
雰囲気でみてやってください！
プログラムコードは配布します。

参考にさせてもらいました！

- **Kaggleチュートリアル Titanicで上位2%以内に入るノウハウ**
- https://qiita.com/jun40vn/items/d8a1f71fae680589e05c

@jun40vn (Yuichi Sugita)

Kaggleチュートリアル Titanicで上位2%以内に入るノウハウ

Python 機械学習 Kaggle titanic

投稿日 2019年11月21日

1.はじめに

機械学習を勉強し始めたので、なにはともあれ **Kaggleの初心者向けチュートリアル、タイタニック**にトライしてみました。

最初は、日本語のWeb情報を参考に、自分なりに特徴量を色々いじってみたわけですが、学習データでは良い精度が出ても、テストデータをSubmitすると、思った様に精度は上がらず、なかなか80%の壁を破れず悶々としていました。

そうした中で、英文なので取っ付き難いですが、Kaggle/TaitanicのNotebookにある先人の知恵を拝借して、やっと上位2%に入れたので、特に参考になった点を中心に備忘録として残します。

Kaggle titanic をランダムフォレストで予測して

01_kaggle_rf_titanic_0.75358.py

gender_submission.csv : 0.76555

02_kaggle_lgbm_titanic_0.74641.py

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# CSVファイルの読み込み
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

# 前処理
train_df["Sex"] = train_df["Sex"].map({"male": 0, "female": 1})
test_df["Sex"] = test_df["Sex"].map({"male": 0, "female": 1})

train_df["Age"].fillna(train_df["Age"].median(), inplace=True)
test_df["Age"].fillna(test_df["Age"].median(), inplace=True)
test_df["Fare"].fillna(test_df["Fare"].median(), inplace=True)

train_df["Embarked"].fillna("S", inplace=True)
test_df["Embarked"].fillna("S", inplace=True)
train_df["Embarked"] = train_df["Embarked"].map({"S": 0, "C": 1, "Q": 2})
test_df["Embarked"] = test_df["Embarked"].map({"S": 0, "C": 1, "Q": 2})

# 特徴量の指定
features = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]
X = train_df[features]
y = train_df["Survived"]
X_test = test_df[features]

# モデル学習と予測
model = RandomForestClassifier(random_state=42)
model.fit(X, y)
predictions = model.predict(X_test)

# 提出ファイルの作成
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": predictions
})
submission.to_csv("01_kaggle_rf_titanic.csv", index=False)
```

前処理を入れてくれている。

再現性確保のため random_state 固定は必ず入れるように指示しましょう。

Kaggle code (notebook) 向けLGBMにして

LGBMの方がスコアが落ちる。チューニングすれば、LGBMの方が良いスコアが出る可能性あるが、ChatGPT内で実行出来ることもあり、今回は主にランダムフォレストを採用。

Kaggle code (notebook) にコピペして実行

```python
import pandas as pd
import lightgbm as lgb

# CSVファイルの読み込み
train_df = pd.read_csv("/kaggle/input/titanic/train.csv")
test_df = pd.read_csv("/kaggle/input/titanic/test.csv")

# 前処理
train_df["Sex"] = train_df["Sex"].map({"male": 0, "female": 1})
test_df["Sex"] = test_df["Sex"].map({"male": 0, "female": 1})

train_df["Age"].fillna(train_df["Age"].median(), inplace=True)
test_df["Age"].fillna(test_df["Age"].median(), inplace=True)
test_df["Fare"].fillna(test_df["Fare"].median(), inplace=True)

train_df["Embarked"].fillna("S", inplace=True)
test_df["Embarked"].fillna("S", inplace=True)
train_df["Embarked"] = train_df["Embarked"].map({"S": 0, "C": 1, "Q": 2})
test_df["Embarked"] = test_df["Embarked"].map({"S": 0, "C": 1, "Q": 2})

# 特徴量の指定
features = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]
X = train_df[features]
y = train_df["Survived"]
X_test = test_df[features]

# LightGBM モデルで学習と予測
model = lgb.LGBMClassifier(random_state=42)
model.fit(X, y)
preds = model.predict(X_test)

# 提出ファイルの作成
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": preds
})
submission.to_csv("02_kaggle_lgbm_titanic.csv", index=False)
```

特徴量全部(LE:ラベルエンコーディング版)

03_rf_LabelE_nodrop_0.77272.py

特徴量全部入れてラベルエンコーディングしてからランダムフォレスト予測

このあたりから完全に同じコードは出てこないと思います。
必要に応じて、提供されているコードをChatGPTへアップロードして使ってください。

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# データ読み込みと結合
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
test_df['Survived'] = np.nan # 後で分離できるように np.nan(null) を入れておく
df = pd.concat([train_df, test_df], ignore_index=True, sort=False)

# 欠損補完
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())

# ラベルエンコーディング
label_cols = ['Sex', 'Embarked', 'Name', 'Ticket', 'Cabin']
for col in label_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))

# X_train, y_train, X_test 分離
dfm = df # 列を削除しない
train = dfm[dfm['Survived'].notnull()]
test = dfm[dfm['Survived'].isnull()].drop('Survived', axis=1)
X = train.drop('Survived', axis=1).values
y = train['Survived'].astype(int).values
X_test = test.values

# モデル学習・予測
model = RandomForestClassifier(random_state=42)
model.fit(X, y)
final_preds = model.predict(X_test)

# submissionファイル出力
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": final_preds
})
submission.to_csv("03_rf_LabelE_nodrop.csv", index=False)
```

LE

特徴量数(Survived を除く):11個

勝手に順番変えられる時がある。安定化のために
sort=False
を指定する。
順番変わるとLabel Encoding, Kaggle Score判定など挙動が変わる可能性あり。

gender_submission.csv :
0.76555 は超えた！

Titanic 初期特徴量

おさらい

列名	データ型	説明
`PassengerId`	int	乗客ID。ユニークな識別番号。モデルには不要（識別用）。
`Survived`	int (0/1)	生存フラグ（0 = 死亡, 1 = 生存）。**目的変数**（`train.csv` のみに存在）。
`Pclass`	int (1-3)	チケットクラス（1 = 上級, 2 = 中級, 3 = 下級）。社会階級のproxy。
`Name`	str	名前。敬称（`Mr`, `Mrs`, `Miss` など）を抽出可能。
`Sex`	str	性別（`male` / `female`）。非常に重要な特徴。
`Age`	float	年齢。欠損あり（約20%）。子供・大人の判別などに有用。
`SibSp`	int	タイタニックに同乗していた**兄弟または配偶者**の数。家族構成の一部を表す。
`Parch`	int	タイタニックに同乗していた**親または子供**の数。これも家族構成を示す。
`Ticket`	str	チケット番号。情報抽出可能だが、ノイズも多い。
`Fare`	float	チケット料金。社会的地位や等級と相関がある。欠損値あり（`test.csv` のみ）。
`Cabin`	str	キャビン番号。欠損多数。文字列の先頭（デッキレベル）を抽出して使う。
`Embarked`	str	乗船港（`C` = Cherbourg, `Q` = Queenstown, `S` = Southampton）。欠損あり。

文字列はLE必要

‘Braund, Mr. Owen Harris’ など

‘11774’, ‘STON/O2. 3101282’ など

‘B28’, ‘E46’, ‘F G73’ など

特徴量削減版(LE版)

特徴量数(Survived を除く):
7個

【削除した特徴量と理由】
・PassengerId: ただの固有ID。いらない。
・Name, Ticket, Cabin:
単なるラベルエンコーディングでは意味が無い。

n_estimators=200はこの後のCV安定させる大きめの木を採用。デフォルトは100。

Titanicのような小さいテーブルでは、特徴量削減し、max_depth制限すると、過学習防げて、スコアが上がる。

```
n_estimators=200, max_depth=6,
```

04_rf_LabelE_0.76315.py

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# データ読み込みと結合
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
test_df['Survived'] = np.nan # 後で分離できるように np.nan(null) を入れておく
df = pd.concat([train_df, test_df], ignore_index=True, sort=False)

# 欠損補完
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())

# ラベルエンコーディング
label_cols = ['Sex', 'Embarked']
for col in label_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))

# X_train, y_train, X_test 分離
#['Pclass','Sex','Age','Fare','Embarked','SibSp','Parch']
dfm = df.drop(columns=['PassengerId','Name','Ticket','Cabin'], errors='ignore')
train = dfm[dfm['Survived'].notnull()]
test = dfm[dfm['Survived'].isnull()].drop('Survived', axis=1)
X = train.drop('Survived', axis=1).values
y = train['Survived'].astype(int).values
X_test = test.values

# モデル学習・予測
model = RandomForestClassifier(random_state=42)
model.fit(X, y)
final_preds = model.predict(X_test)

# submissionファイル出力
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": final_preds
})
submission.to_csv("04_rf_LabelE.csv", index=False)
```

05_rf_LabelE_200-6_0.78229.py

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# データ読み込みと結合
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
test_df['Survived'] = np.nan # 後で分離できるように np.nan(null) を入れておく
df = pd.concat([train_df, test_df], ignore_index=True, sort=False)

# 欠損補完
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())

# ラベルエンコーディング
label_cols = ['Sex', 'Embarked']
for col in label_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))

# X_train, y_train, X_test 分離
#['Pclass','Sex','Age','Fare','Embarked','SibSp','Parch']
dfm = df.drop(columns=['PassengerId','Name','Ticket','Cabin'], errors='ignore')
train = dfm[dfm['Survived'].notnull()]
test = dfm[dfm['Survived'].isnull()].drop('Survived', axis=1)
X = train.drop('Survived', axis=1).values
y = train['Survived'].astype(int).values
X_test = test.values

# モデル学習・予測
model = RandomForestClassifier(n_estimators=200, max_depth=6, random_state=42)
model.fit(X, y)
final_preds = model.predict(X_test)

# submissionファイル出力
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": final_preds
})
submission.to_csv("05_rf_LabelE_200-6.csv", index=False)
```

CV StratifiedKFold:5 説明

train.csv (Survivedあり): 891人

test.csv: 418人

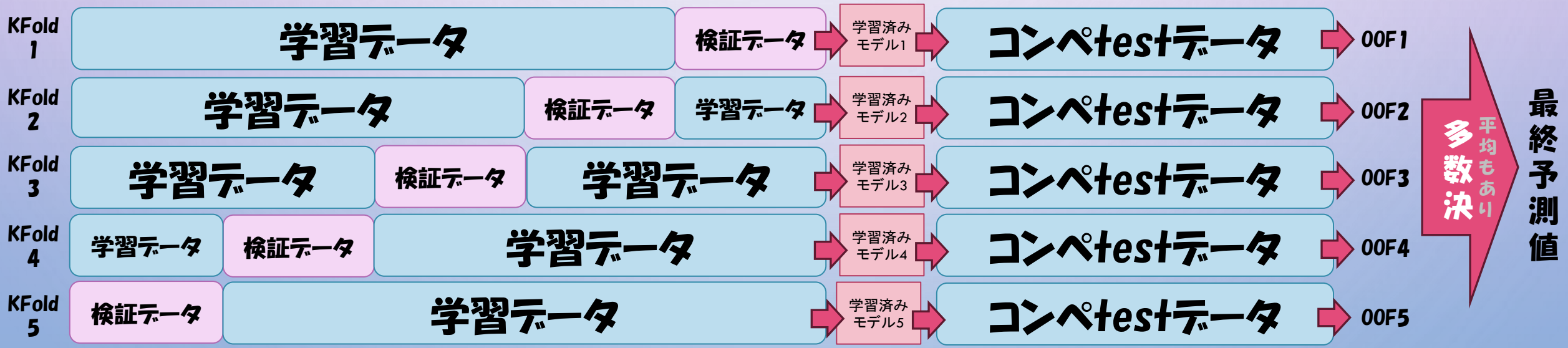
04_rf_LabelE_0.76315.py のケース　学習/検証 データ → 学習済みモデル → コンペtestデータ

全データで学習/検証するため、trainに過学習しすぎる可能性あり。未知のデータに弱くなる→汎化性が低い

学習データ　検証データ → 学習済みモデル → コンペtestデータ

汎化性は増すが、学習に使われないデータが出てくる。

CV(Cross Validation) KFold:5



通常 StratifiedKFold() を使う。各特徴量を均等に配分してくれる。

OOF: Out of Fold

CV:SKFold:5導入

06_rf_LabelE_200-6_skfold5_0.77990.py

CV SKFold:5 を導入して、少しスコアが下がったが・・・、Private LBでの安定性(スコアが下がらないよう)を考慮して、CV SKFold:5、n_estimators:200で進める。

CVスコアは高めに表示されます

```
Fold 1 CV Accuracy: 0.84916
Fold 2 CV Accuracy: 0.80899
Fold 3 CV Accuracy: 0.81461
Fold 4 CV Accuracy: 0.84270
Fold 5 CV Accuracy: 0.85955
Mean CV Accuracy: 0.83500
```

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from scipy.stats import mode

# データ読み込みと結合
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
test_df['Survived'] = np.nan # 後で分離できるように np.nan(null) を入れておく
df = pd.concat([train_df, test_df], ignore_index=True, sort=False)

# Age, Fare補完 : base
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())

# 特徴量とラベルエンコーディング, Drop 'PassengerId','Name','Ticket','Cabin'
features = ['Pclass','Sex','Age','Fare','Embarked', 'SibSp', 'Parch'] '
dfm = df[['Survived'] + features].copy()
for col in ['Sex', 'Embarked']:
    le = LabelEncoder()
    dfm[col] = le.fit_transform(dfm[col].astype(str))

# train/test 分割
train = dfm[dfm['Survived'].notnull()]
test = dfm[dfm['Survived'].isnull()].drop('Survived', axis=1)
X = train.drop('Survived', axis=1).values
y = train['Survived'].astype(int).values
X_test = test.values

# StratifiedKFold + Mode voting(多数決)、CVスコア表示
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
test_preds = []
val_scores = []
for fold, (train_idx, val_idx) in enumerate(skf.split(X, y), 1):
    model = RandomForestClassifier(n_estimators=200, max_depth=6, random_state=42)
    model.fit(X[train_idx], y[train_idx])
    val_pred = model.predict(X[val_idx])
    val_acc = accuracy_score(y[val_idx], val_pred)
    val_scores.append(val_acc)
    print(f"Fold {fold} CV Accuracy: {val_acc:.5f}")
    test_preds.append(model.predict(X_test))
print(f"Mean CV Accuracy: {np.mean(val_scores):.5f}")

# 多数決で最終予測を決定
test_preds = np.array(test_preds)
final_preds = mode(test_preds, axis=0, keepdims=False).mode

# サブミッション作成
submission = pd.DataFrame({
    'PassengerId': test_df['PassengerId'],
    'Survived': final_preds.astype(int)
})
submission.to_csv("06_rf_LabelE_200-6_skfold5.csv", index=False)
```

SKFold:5 CVスコア表示なしだともっとシンプル。

多数決アンサンブル meanもよく見る。

前処理導入

07_rf_LabelE_200-6_skfold5_Fe_0.78947.py

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from scipy.stats import mode

# データ読み込みと結合
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
test_df['Survived'] = np.nan # 後で分離できるように np.nan(null) を入れておく
df = pd.concat([train_df, test_df], ignore_index=True, sort=False)

# Age, Fare補完 : base
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())
```

前処理追加

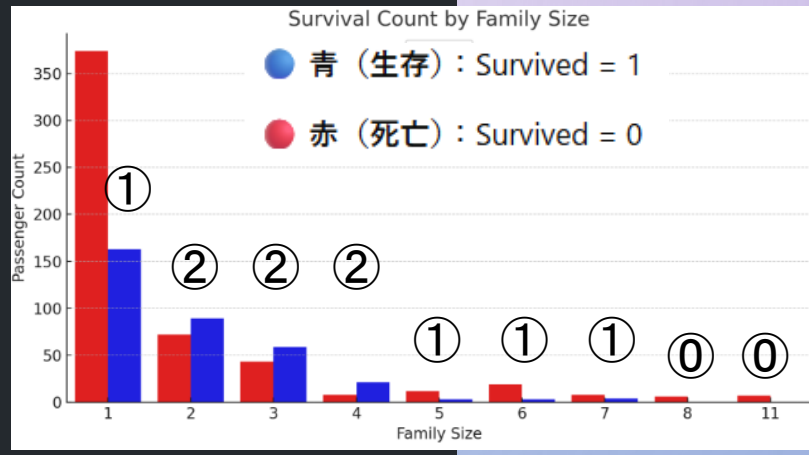
```python
#### 特徴量エンジニアリング : 全てまとめて 0.01 の精度向上
# Title作成 : 0.007 の精度向上, ['Master', 'Miss', 'Mr', 'Mrs', 'Officer', 'Royalty']に集約
df['Title'] = df['Name'].map(lambda x: x.split(', ')[1].split('. ')[0])
df['Title'].replace(['Capt', 'Col', 'Major', 'Dr', 'Rev'], 'Officer', inplace=True)
df['Title'].replace(['Don', 'Sir', 'the Countess', 'Lady', 'Dona'], 'Royalty', inplace=True)
df['Title'].replace(['Mme', 'Ms'], 'Mrs', inplace=True)
df['Title'].replace(['Mlle'], 'Miss', inplace=True)
df['Title'].replace(['Jonkheer'], 'Master', inplace=True)
# Cabin補完 : 0.005 の精度向上
df['Cabin'] = df['Cabin'].fillna('Unknown')
df['Cabin_L'] = df['Cabin'].str.get(0) # 最初の文字をCabinラベルとして使用
# Embarked補完 : 0.005 の精度向上
df['Embarked'] = df['Embarked'].fillna('S') # 'S'は最貧度
# Familyラベル : シンプルSamilySizeより0.015、なしより0.017 の精度向上
df['Family'] = df['SibSp'] + df['Parch'] + 1  # 家族人数を計算（本人を含める）
df['Family_L'] = np.select([
    (df['Family'] >= 2) & (df['Family'] <= 4),                    # 2:中規模
    (df['Family'] == 1) | ((df['Family'] >= 5) & (df['Family'] <= 7)), # 1:単独 or やや多め
    df['Family'] >= 8                                             # 0:超大規模
], [2, 1, 0])
```

```python
# 特徴量とラベルエンコーディング
features = ['Pclass','Sex','Age','Fare','Embarked','Title','Family_L','Cabin_L']
dfm = df[['Survived'] + features].copy()
for col in ['Sex', 'Embarked', 'Title', 'Cabin_L']:
    le = LabelEncoder()
    dfm[col] = le.fit_transform(dfm[col].astype(str))

# train/test 分割
train = dfm[dfm['Survived'].notnull()]
test = dfm[dfm['Survived'].isnull()].drop('Survived', axis=1)
X = train.drop('Survived', axis=1).values
y = train['Survived'].astype(int).values
X_test = test.values

# StratifiedKFold + Mode voting(多数決)、CVスコア表示
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
test_preds = []
val_scores = []
for fold, (train_idx, val_idx) in enumerate(skf.split(X, y), 1):
    model = RandomForestClassifier(n_estimators=200, max_depth=6, random_state=42)
    model.fit(X[train_idx], y[train_idx])
    val_pred = model.predict(X[val_idx])
    val_acc = accuracy_score(y[val_idx], val_pred)
    val_scores.append(val_acc)
    print(f"Fold {fold} CV Accuracy: {val_acc:.5f}")
    test_preds.append(model.predict(X_test))
print(f"Mean CV Accuracy: {np.mean(val_scores):.5f}")

# 多数決で最終予測を決定
test_preds = np.array(test_preds)
final_preds = mode(test_preds, axis=0, keepdims=False).mode

# サブミッション作成
submission = pd.DataFrame({
    'PassengerId': test_df['PassengerId'],
    'Survived': final_preds.astype(int)
})
submission.to_csv("07_rf_LabelE_200-6_skfold5_Fe.csv", index=False)
```

前処理説明

全体で0.01ほどのスコア向上。各特徴量を個別にon/offすると、下記コメント数値(非CV)でより影響が大きいようだ。

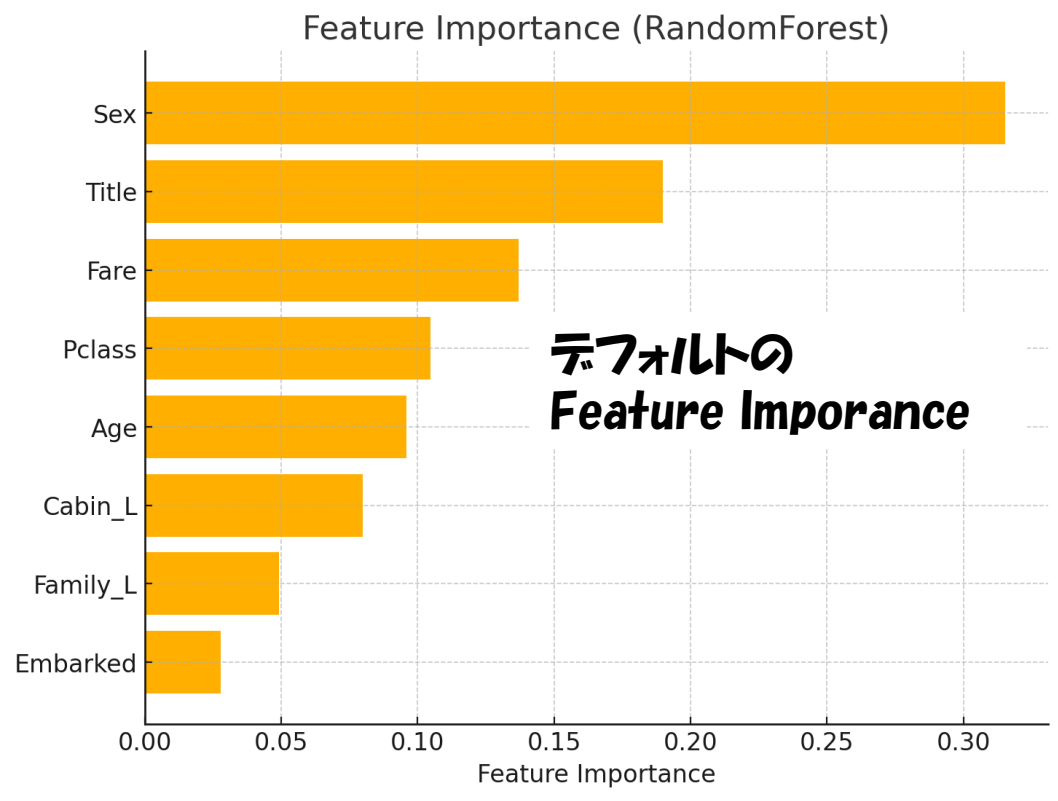
```python
#### 特徴量エンジニアリング : 全てまとめて 0.01 の精度向上
# Title作成 : 0.007 の精度向上, ['Master', 'Miss', 'Mr', 'Mrs', 'Officer', 'Royalty']に集約
df['Title'] = df['Name'].map(lambda x: x.split(', ')[1].split('. ')[0])
df['Title'].replace(['Capt', 'Col', 'Major', 'Dr', 'Rev'], 'Officer', inplace=True)
df['Title'].replace(['Don', 'Sir', 'the Countess', 'Lady', 'Dona'], 'Royalty', inplace=True)
df['Title'].replace(['Mme', 'Ms'], 'Mrs', inplace=True)
df['Title'].replace(['Mlle'], 'Miss', inplace=True)
df['Title'].replace(['Jonkheer'], 'Master', inplace=True)
# Cabin補完 : 0.005 の精度向上
df['Cabin'] = df['Cabin'].fillna('Unknown')
df['Cabin_L'] = df['Cabin'].str.get(0) # 最初の文字をCabinラベルとして使用
# Embarked補完 : 0.005 の精度向上
df['Embarked'] = df['Embarked'].fillna('S') # 'S'は最貧度
# Familyラベル : シンプルSamilySizeより0.015、なしより0.017 の精度向上
df['Family'] = df['SibSp'] + df['Parch'] + 1  # 家族人数を計算（本人を含める）
df['Family_L'] = np.select([
    (df['Family'] >= 2) & (df['Family'] <= 4),                    # 2:中規模
    (df['Family'] == 1) | ((df['Family'] >= 5) & (df['Family'] <= 7)), # 1:単独 or やや多め
    df['Family'] >= 8                                             # 0:超大規模
], [2, 1, 0])
```

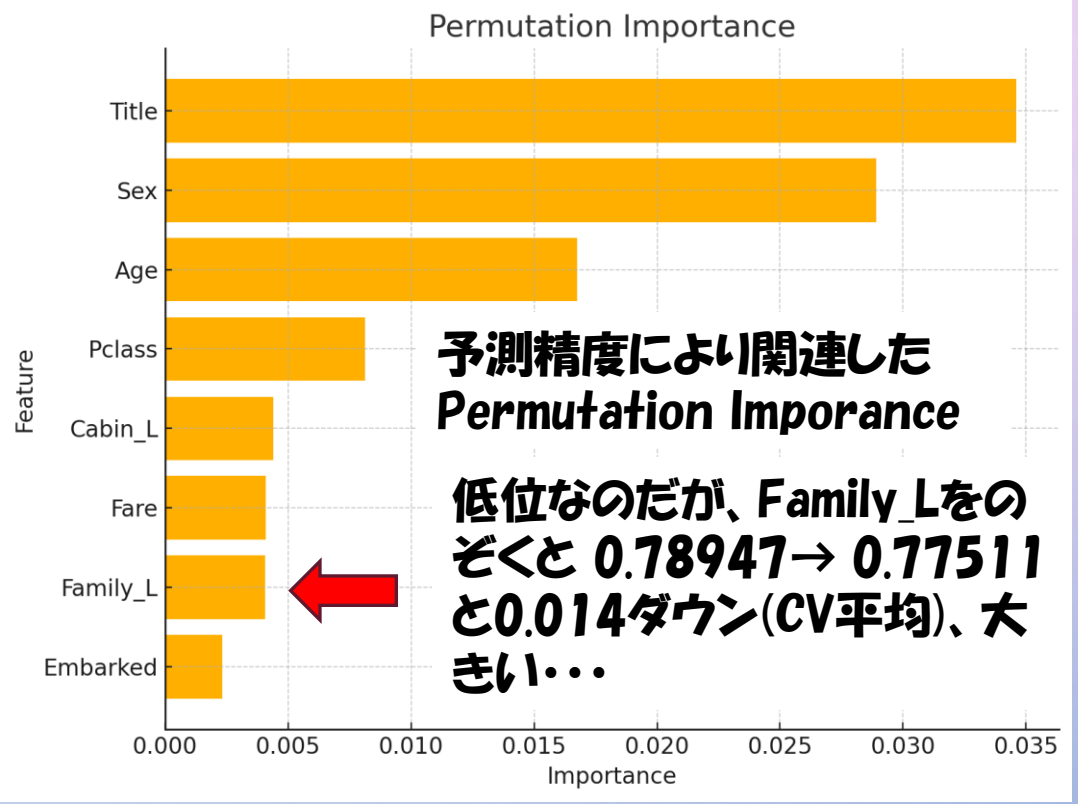


Feature Importance

LGBMでは
左がsplit(デフォルト)、
右がgain。

feature importanceを横棒グラフで表示して





① Mean Decrease in Impurity (MDI)

（決定木の分割で減少したジニ不純度やエントロピーの合計）

- scikit-learn の `feature_importances_` 属性がこれ
- 各特徴量が木の分割にどれだけ貢献したかを測る
- 計算が高速
- バイアスがある（カテゴリ特徴量や多くの分割候補がある特徴量に有利になる傾向）

② Permutation Importance

（特徴量の値をシャッフルして予測精度の低下を測定）

- 特徴量の値をシャッフル → 精度低下を測定 → 重要度を算出
- モデル訓練後に外部から計算可能
- 特徴量のスケールや型に依存しにくい
- ノイズに強く、実用上こちらを採用する論文やKaggleも多い
- 計算コストは高め

後処理(ルールベース)導入

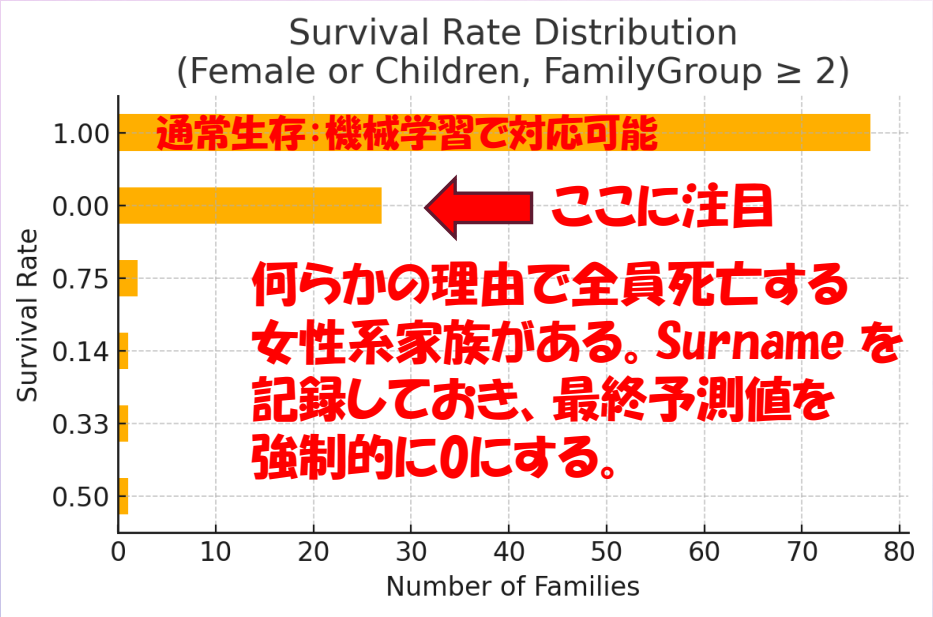
08_rf_LabelE_200-6_skfold5_FeAll_0.80382.py

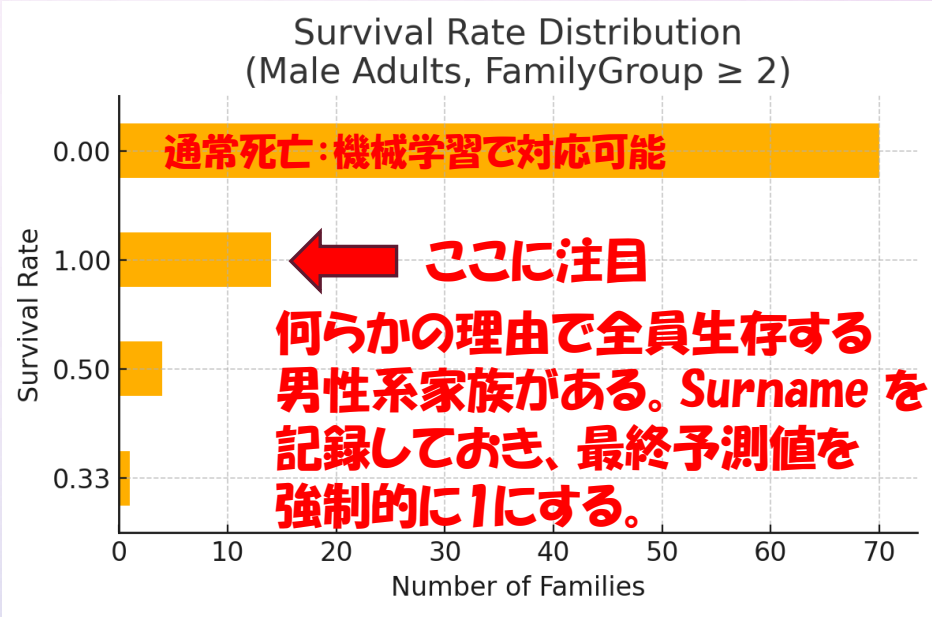
0.80越え！

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from scipy.stats import mode

# データ読み込みと結合
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
test_df['Survived'] = np.nan # 後で分離できるように np.nan(null) を入れておく
df = pd.concat([train_df, test_df], ignore_index=True, sort=False)

# Age, Fare補完 : base
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())

#### 特徴量エンジニアリング : 全てまとめて 0.01 の精度向上
# Title作成 : 0.007 の精度向上, ['Master', 'Miss', 'Mr', 'Mrs', 'Officer', 'Royalty']に集約
df['Title'] = df['Name'].map(lambda x: x.split(', ')[1].split('. ')[0])
df['Title'].replace(['Capt', 'Col', 'Major', 'Dr', 'Rev'], 'Officer', inplace=True)
df['Title'].replace(['Don', 'Sir', 'the Countess', 'Lady', 'Dona'], 'Royalty', inplace=True)
df['Title'].replace(['Mme', 'Ms'], 'Mrs', inplace=True)
df['Title'].replace(['Mlle'], 'Miss', inplace=True)
df['Title'].replace(['Jonkheer'], 'Master', inplace=True)
# Cabin補完 : 0.005 の精度向上
df['Cabin'] = df['Cabin'].fillna('Unknown')
df['Cabin_L'] = df['Cabin'].str.get(0) # 最初の文字をCabinラベルとして使用
# Embarked補完 : 0.005 の精度向上
df['Embarked'] = df['Embarked'].fillna('S') # 'S'は最貧度
# Familyラベル : シンプルSamilySizeより0.015、なしより0.017 の精度向上
df['Family'] = df['SibSp'] + df['Parch'] + 1  # 家族人数を計算（本人を含める）
df['Family_L'] = np.select([
    (df['Family'] >= 2) & (df['Family'] <= 4),                      # 2:中規模
    (df['Family'] == 1) | ((df['Family'] >= 5) & (df['Family'] <= 7)), # 1:単独 or やや多め
    df['Family'] >= 8                                               # 0:超大規模
], [2, 1, 0])

# 特徴量とラベルエンコーディング
features = ['Pclass','Sex','Age','Fare','Embarked','Title','Family_L','Cabin_L']
dfm = df[['Survived'] + features].copy()
for col in ['Sex', 'Embarked', 'Title', 'Cabin_L']:
    le = LabelEncoder()
    dfm[col] = le.fit_transform(dfm[col].astype(str))

# train/test 分割
train = dfm[dfm['Survived'].notnull()]
test = dfm[dfm['Survived'].isnull()].drop('Survived', axis=1)
X = train.drop('Survived', axis=1).values
y = train['Survived'].astype(int).values
X_test = test.values

# StratifiedKFold + Mode voting(多数決)、CVスコア表示
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
test_preds = []
val_scores = []
for fold, (train_idx, val_idx) in enumerate(skf.split(X, y), 1):
    model = RandomForestClassifier(n_estimators=200, max_depth=6, random_state=42)
    model.fit(X[train_idx], y[train_idx])
    val_pred = model.predict(X[val_idx])
    val_acc = accuracy_score(y[val_idx], val_pred)
    val_scores.append(val_acc)
    print(f"Fold {fold} CV Accuracy: {val_acc:.5f}")
    test_preds.append(model.predict(X_test))
print(f"Mean CV Accuracy: {np.mean(val_scores):.5f}")

# 多数決で最終予測を決定
test_preds = np.array(test_preds)
final_preds = mode(test_preds, axis=0, keepdims=False).mode

# 特殊家族補正
# 通常予測とはことなる特殊な家族の補正（最終予測用）: 0.014 の精度向上
surnames = df['Name'].str.split(',').str[0].str.strip()  # 名前から姓（Surname）を取り出す
family_n = surnames.map(surnames.value_counts()) # 同じ姓の人数をカウント（家族単位）
# 「2人以上の家族」かつ「16歳以下または女性」で構成(通常生存)され「全員死亡」している特異な家族リスト
Dead_list = set(surnames[ (family_n>=2)&((df['Age']<=16)|(df['Sex']=='female'))&(df['Survived']==0) ])
# 「2人以上の家族」かつ「男性の大人」で構成(通常死亡)され「全員生存」している特異な家族リスト
Surv_list = set(surnames[ (family_n>=2)&((df['Age']> 16)&(df['Sex']=='male'  ))&(df['Survived']==1) ])
# 予測データに同じ姓があれば、そのパターンに従って生死を補正する
for i, s in enumerate(df.loc[df['Survived'].isnull(), 'Name'].str.split(',').str[0].str.strip()):
    final_preds[i] = 0 if s in Dead_list else 1 if s in Surv_list else final_preds[i]

# サブミッション作成
submission = pd.DataFrame({
    'PassengerId': test_df['PassengerId'],
    'Survived': final_preds.astype(int)
})
submission.to_csv("08_rf_LabelE_200-6_skfold5_FeAll.csv", index=False)
```

後処理追加

後処理(ルールベース)説明





将来は、こういう特徴量を自動発見して学習する機械学習アルゴリズムが生まれるでしょう。

```
# 特殊家族補正
# 通常予測とはことなる特殊な家族の補正（最終予測用）: 0.014 の精度向上
surnames = df['Name'].str.split(',').str[0].str.strip()  # 名前から姓（Surname）を取り出す
family_n = surnames.map(surnames.value_counts()) # 同じ姓の人数をカウント（家族単位）
# 「2人以上の家族」かつ「16歳以下または女性」で構成(通常生存)され「全員死亡」している特異な家族リスト
Dead_list = set(surnames[ (family_n>=2)&((df['Age']<=16)|(df['Sex']=='female'))&(df['Survived']==0) ])
# 「2人以上の家族」かつ「男性の大人」で構成(通常死亡)され「全員生存」している特異な家族リスト
Surv_list = set(surnames[ (family_n>=2)&((df['Age']> 16)&(df['Sex']=='male'   ))&(df['Survived']==1) ])
# 予測データに同じ姓があれば、そのパターンに従って生死を補正する
for i, s in enumerate(df.loc[df['Survived'].isnull(), 'Name'].str.split(',').str[0].str.strip()):
    final_preds[i] = 0 if s in Dead_list else 1 if s in Surv_list else final_preds[i]
```

さらなるスコア向上案

- 特徴量エンジニアリングのさらなる探求
- LGBMでtuning
- アンサンブル/スタッキング
  - たとえば下記アルゴリズムはscikit learnに入っているので、ChatGPT内でアンサンブル実行可能。
    - LogisticRegression
    - Random Forests
    - Gradient Boosting Trees
  - Kaggle code実行などでより強いアルゴリズムでアンサンブル
    - LGBM
    - CatBoost
    - XGBoost, etc

やってみたが・・・
アンサンブルでスコアは上がらなかった：0.79186（汎化性は高いかも）
各アルゴリズム毎に、特徴量前処理、ハイパーパラメータの
チューニングが必要かも知れない。

まとめ

- まだスコア向上策はありそう
- 最後のルールベース追加は、様々な議論あるだろう
  - Kaggleとしては合法ルール
  - 人間が特徴量エンジニアリングで気づくべき派
    - ひらめき・特徴量エンジニアリングで気づいた者が勝利
    - ルールベース追加や、この予測が入るように前処理するなど
  - 機械学習アルゴリズムが自動導入するべき派
    - 人間に頼らず、強い機械学習アルゴリズムが自動で導入するべきという意見もある。
    - よりより機械学習アルゴリズムの探索・開発へ
    - なのでTitanicのような古いコンペでも新しいアルゴリズムが試されている
- ChatGPTは万能ではない。しかしAIペアプログラミング相手として優秀！
  - 細かいSyntax Error・凡ミスは、発生しても自動修正してくれる。
  - 人間は本質的なデータサイエンス思考を止めなくて良い。細かいプログラミングに煩わされない。
  - 今のChatGPTがAIエージェント的にどこまで使えるか、楽しみながら確認してみてください！

Kaggle面白いですよ！
ChatGPT(生成AI)
駆使して、楽しみましょう！

本資料、各プログラムは 菜園の柱 ブログに置きます

- https://garden-pillars.blog/