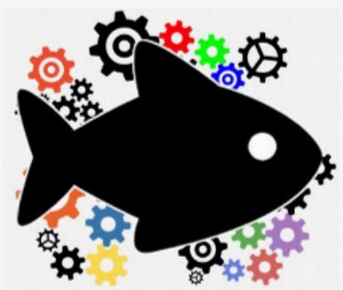
PyCaretでkaggleにチャレンジ中

Pythonデータ分析勉強会＃19　2020/5/16(土)

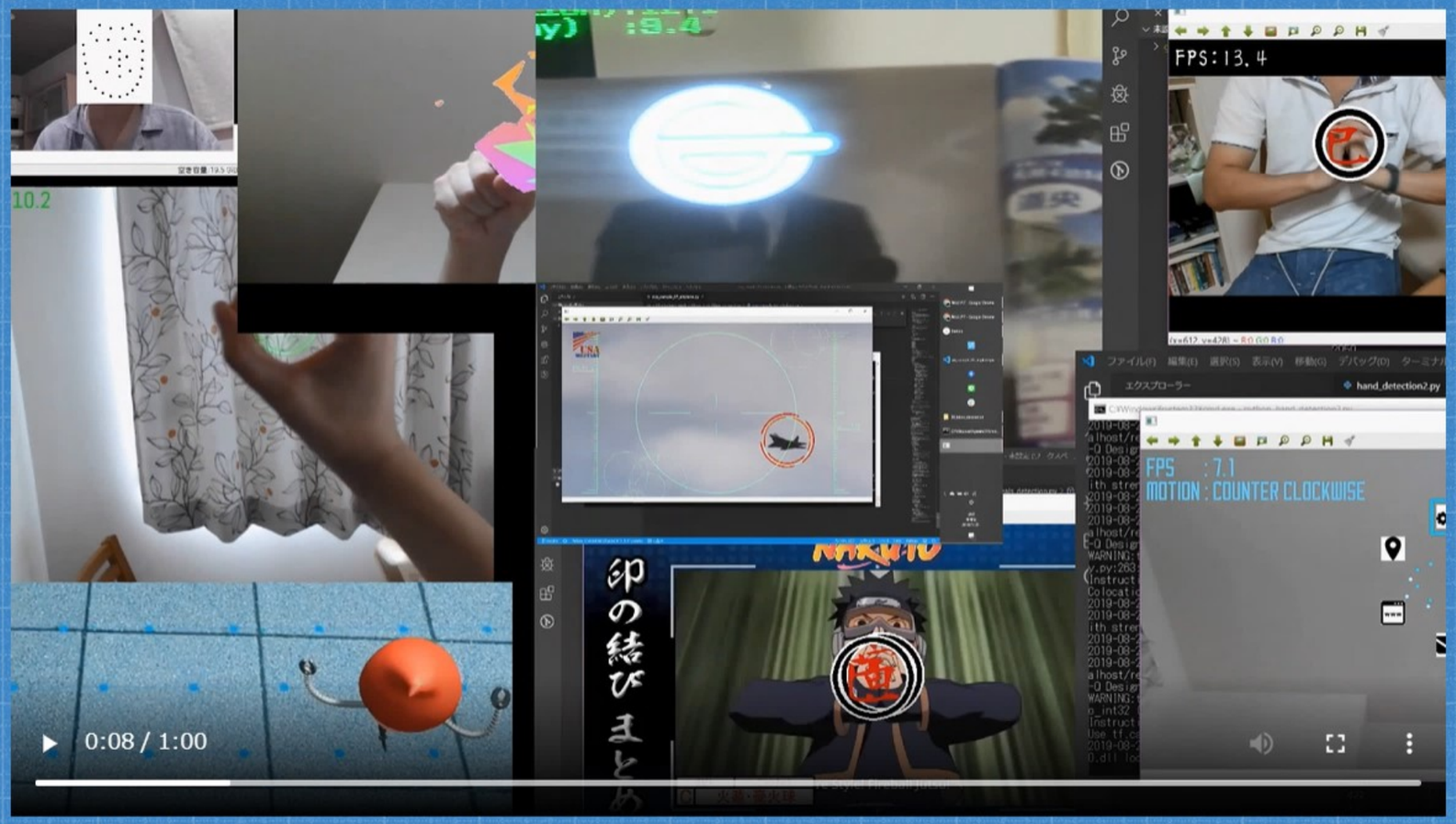
お前、誰よ？



名前　　：高橋かずひと
Twitter　：@KzhtTkhs
お仕事　　：組込、画像処理、IoT、Webアプリなど

その他、個人の活動で、インディーゲーム制作のお手伝をしています。

最近はディープラーニング(主に物体検出)やＡＲでよく遊んでいます。



業務命令

上長「高橋くん機械学習(※何を指しているか謎)出来たよね？」
高橋「機械学習(画像系)出来なくもなくもないです」

上長「全社取り組みでAI注力することになったので、
kaggleでメダル取ってください」

上長「あと後輩くん(知識無し)もサクッと育ててほしい」

高橋「そんなバカな🤮」　※ノンノンフィクション

PyCaretって？

「PyCaret is an open source, low-code machine learning library in Python that allows you to go from preparing your data to deploying your model within seconds in your choice of notebook environment.」

DeepL翻訳：PyCaretはオープンソースのPythonによるローコード機械学習ライブラリで、データの準備からモデルのデプロイまで、お好みのノートブック環境で数秒で行うことができます。

参照：https://pycaret.org/

特徴①：Increased Productivity

「PyCaret being a low-code library makes you more productive. With less time spent coding, you and your team can now focus on business problems.」

DeepL翻訳：PyCaret はローコードライブラリであるため、より生産性が高くなります。コーディングに費やす時間が減り、あなたとあなたのチームはビジネス上の問題に集中できるようになりました。

特徴②：Easy to Use

「PyCaret is simple and easy to use machine learning library that will help you to perform end-to-end ML experiments with less lines of code.」

DeepL翻訳：PyCaretはシンプルで使いやすい機械学習ライブラリで、少ないコード行数でエンドツーエンドのML実験を行うことができます。

特徴③：Business Ready

「PyCaret is a business ready solution. It allows you to do prototyping quickly and efficiently from your choice of notebook environment.」

DeepL翻訳：PyCaretはビジネスに即したソリューションです。選んだノートブック環境から素早く効率的にプロトタイピングを行うことができます。

出来ること

- クラス分類
- 回帰分析
- クラスタリング
- 異常検知(教師無し学習)
- 自然言語処理
- 相関ルールマイニング

実装例：https://github.com/Kazuhito00/PyCaret-Learn

とりあえずTitanic

データ読み込み

In [2]:
```
import numpy as np
import pandas as pd
import pandas_profiling
```

In [3]:
```
train_data = pd.read_csv('./input/titanic/train.csv')
test_data  = pd.read_csv('./input/titanic/test.csv')
submission_data = pd.read_csv('./input/titanic/gender_submission.csv')
```

In [4]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [5]:
```
train_data.profile_report()
```

Pandas Profiling Report

Overview | Variables | Correlations | Missing values | Sample

Overview

Dataset info

Number of variables	12
Number of observations	891
Missing cells	866 (8.1%)
Duplicate rows	0 (0.0%)
Total size in memory	83.7 KiB
Average record size in memory	96.1 B

Variables types

Numeric	5
Categorical	5
Boolean	1
Date	0
URL	0
Text (Unique)	1
Rejected	0
Unsupported	0

Warnings

`Age` has 177 (19.9%) missing values	Missing
`Cabin` has a high cardinality: 148 distinct values	Warning
`Cabin` has 687 (77.1%) missing values	Missing
`Fare` has 15 (1.7%) zeros	Zeros
`Parch` has 678 (76.1%) zeros	Zeros
`SibSp` has 608 (68.2%) zeros	Zeros
`Ticket` has a high cardinality: 681 distinct values	Warning

今回はEDA、特徴量エンジニアリング省略。。。

データセットアップ

In [6]:
```
# クラス分類用インポート
from pycaret.classification import *
```

In [7]:
```
exp = setup(data=train_data,    # 訓練用データ
            target='Survived', # 目標変数
            ignore_features = ['PassengerId', 'Name'],    # 訓練に使用しない説明変数
            session_id=42,      # 乱数シード
            silent=True)
```

Setup Succesfully Completed!

	Description	Value
0	session_id	42
1	Target Type	Binary
2	Label Encoded	None
3	Original Data	(891, 12)
4	Missing Values	True
5	Numeric Features	3
6	Categorical Features	8
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(891, 12)
11	Transformed Train Set	(623, 757)
12	Transformed Test Set	(268, 757)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	False

モデル比較

In [8]:
```
compare_models(fold=5)
```

Out[8]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	Ridge Classifier	0.839400	0.000000	0.752700	0.820200	0.782700	0.655800
1	Logistic Regression	0.833000	0.872500	0.752700	0.804400	0.775400	0.642800
2	CatBoost Classifier	0.831400	0.873900	0.702700	0.836200	0.762200	0.633000
3	Light Gradient Boosting Machine	0.821800	0.862400	0.723800	0.798700	0.757700	0.617500
4	Extreme Gradient Boosting	0.820200	0.866700	0.719600	0.795100	0.754800	0.613400
5	Gradient Boosting Classifier	0.817000	0.873400	0.694300	0.803400	0.743800	0.602700
6	Extra Trees Classifier	0.815400	0.865500	0.702900	0.796000	0.745500	0.601500
7	Ada Boost Classifier	0.812200	0.848100	0.727900	0.772800	0.748400	0.598800
8	Decision Tree Classifier	0.805700	0.787200	0.707400	0.770100	0.736600	0.583200
9	Random Forest Classifier	0.801000	0.846800	0.644100	0.804300	0.712000	0.563000
10	Linear Discriminant Analysis	0.743200	0.798700	0.572400	0.709800	0.626600	0.435900
11	K Neighbors Classifier	0.695000	0.733600	0.514500	0.626500	0.561400	0.332100
12	SVM - Linear Kernel	0.635800	0.000000	0.648300	0.649200	0.534500	0.270600

モデル生成

In [9]:
```
lgbm_model = create_model('lightgbm')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8889	0.9049	0.7917	0.9048	0.8444	0.7586
1	0.7778	0.8472	0.5833	0.7778	0.6667	0.5051
2	0.7937	0.8339	0.7083	0.7391	0.7234	0.5590
3	0.7903	0.9208	0.7826	0.6923	0.7347	0.5624
4	0.8387	0.8805	0.6667	0.8889	0.7619	0.6437
5	0.8871	0.8958	0.8333	0.8696	0.8511	0.7602
6	0.8548	0.9024	0.7917	0.8261	0.8085	0.6917
7	0.7581	0.7544	0.6667	0.6957	0.6809	0.4862
8	0.8871	0.8914	0.8333	0.8696	0.8511	0.7602
9	0.7581	0.8169	0.7083	0.6800	0.6939	0.4940
Mean	0.8235	0.8648	0.7366	0.7944	0.7616	0.6221
SD	0.0512	0.0488	0.0786	0.0837	0.0688	0.1087

モデルチューニング

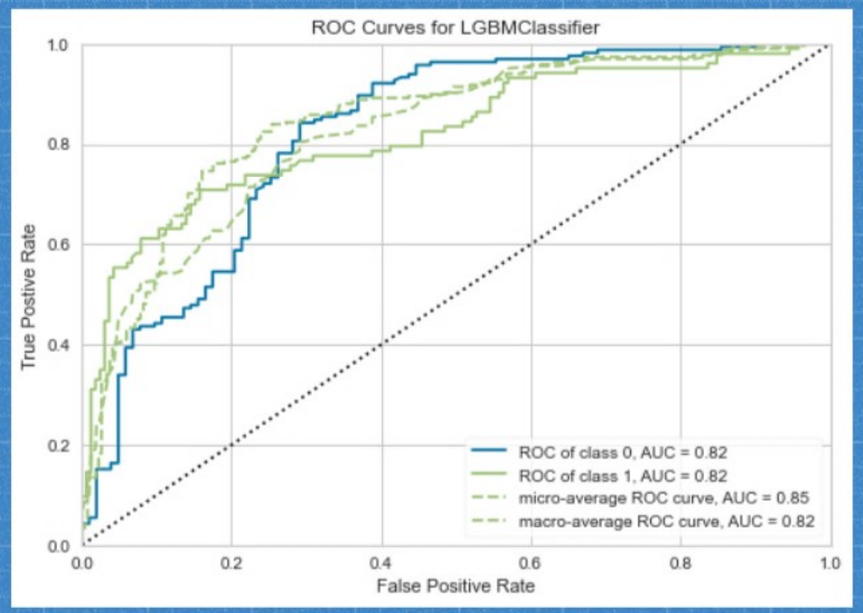
In [10]:
```
tune_lgbm_model = tune_model('lr', optimize='Recall')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8095	0.8803	0.8750	0.7000	0.7778	0.6147
1	0.8413	0.8868	0.8333	0.7692	0.8000	0.6688
2	0.7778	0.8723	0.7500	0.6923	0.7200	0.5363
3	0.8387	0.9164	0.8696	0.7407	0.8000	0.6663
4	0.8065	0.8289	0.7083	0.7727	0.7391	0.5857
5	0.8548	0.9353	0.8333	0.8000	0.8163	0.6964
6	0.7903	0.8991	0.7500	0.7200	0.7347	0.5615
7	0.8065	0.8015	0.6667	0.8000	0.7273	0.5792
8	0.8548	0.8421	0.8333	0.8000	0.8163	0.6964
9	0.7258	0.8015	0.7500	0.6207	0.6792	0.4435
Mean	0.8106	0.8664	0.7870	0.7416	0.7611	0.6049
SD	0.0379	0.0440	0.0677	0.0559	0.0449	0.0764

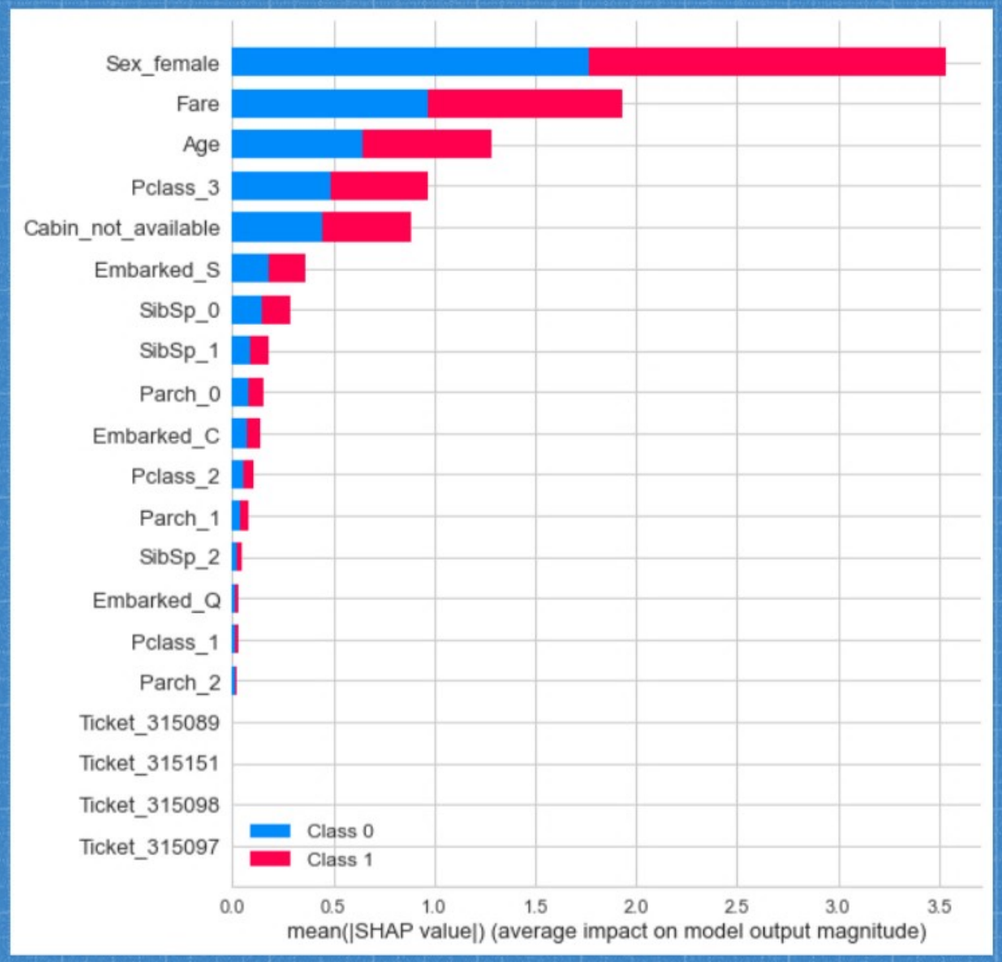
プロット

In [11]:
```
evaluate_model(lgbm_model)
```

Plot Type: Hyperparameters | AUC | Confusion Matrix | Threshold | Precision Recall | Error | Class Report | Feature Selection | Learning Curve | Manifold Learning | Calibration Curve | Validation Curve | Dimensions | Feature Importan··· | Decision Boundary



In [12]:
```
interpret_model(lgbm_model)
```



アンサンブル学習（ブレンディング）

In [13]:
```
blend_model = blend_models(fold=5)   # モデル未指定時はサポートするモデル全てでブレンディング
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8240	0.0	0.7500	0.7826	0.7660	0.6250
1	0.8320	0.0	0.7292	0.8140	0.7692	0.6378
2	0.8720	0.0	0.7292	0.9211	0.8140	0.7184
3	0.8548	0.0	0.7234	0.8718	0.7907	0.6811
4	0.8226	0.0	0.7292	0.7955	0.7609	0.6203
Mean	0.8411	0.0	0.7322	0.8370	0.7801	0.6565
SD	0.0193	0.0	0.0092	0.0519	0.0197	0.0376

アンサンブル学習（スタッキング）

In [14]:
```
rf_model = create_model('rf', verbose=False)
dt_model = create_model('dt', verbose=False)
stack_model = stack_models([rf_model, dt_model, lgbm_model], fold=5)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8400	0.9002	0.8125	0.7800	0.7959	0.6644
1	0.7920	0.8941	0.7083	0.7391	0.7234	0.5569
2	0.8720	0.8715	0.7292	0.9211	0.8140	0.7184
3	0.8226	0.8501	0.6809	0.8205	0.7442	0.6102
4	0.8065	0.8276	0.7500	0.7500	0.7500	0.5921
Mean	0.8266	0.8687	0.7362	0.8021	0.7655	0.6284
SD	0.0278	0.0271	0.0445	0.0658	0.0339	0.0569

アンサンブル学習（バギング）

In [15]:
```
ensembled_model = ensemble_model(dt_model)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8413	0.8520	0.7500	0.8182	0.7826	0.6580
1	0.7937	0.8686	0.5833	0.8235	0.6829	0.5365
2	0.7937	0.8868	0.6250	0.7895	0.6977	0.5442
3	0.8548	0.9108	0.8261	0.7917	0.8085	0.6917
4	0.8548	0.8635	0.6667	0.9412	0.7805	0.6767
5	0.8871	0.9178	0.8333	0.8696	0.8511	0.7602
6	0.8387	0.8898	0.7500	0.8182	0.7826	0.6548
7	0.7903	0.8092	0.6250	0.7895	0.6977	0.5405
8	0.8871	0.8646	0.8333	0.8696	0.8511	0.7602
9	0.7419	0.7895	0.6250	0.6818	0.6522	0.4477
Mean	0.8283	0.8653	0.7118	0.8193	0.7587	0.6271
SD	0.0446	0.0387	0.0931	0.0643	0.0676	0.0993

アンサンブル学習（ブースティング）

In [16]:
```
ensembled_model = ensemble_model(dt_model, method='Boosting')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8571	0.8446	0.7917	0.8261	0.8085	0.6947
1	0.7619	0.7196	0.5417	0.7647	0.6341	0.4652
2	0.7619	0.7356	0.6250	0.7143	0.6667	0.4828
3	0.8065	0.7926	0.7391	0.7391	0.7391	0.5853
4	0.8065	0.7730	0.6250	0.8333	0.7143	0.5724
5	0.8710	0.8640	0.8333	0.8333	0.8333	0.7281
6	0.8226	0.8092	0.7500	0.7826	0.7660	0.6232
7	0.7903	0.7752	0.7083	0.7391	0.7234	0.5547
8	0.8710	0.8640	0.8333	0.8333	0.8333	0.7281
9	0.7903	0.7599	0.6250	0.7895	0.6977	0.5405
Mean	0.8139	0.7938	0.7072	0.7855	0.7416	0.5975
SD	0.0388	0.0485	0.0946	0.0428	0.0649	0.0899

推論

In [17]:
```
predictions = predict_model(lgbm_model, data=test_data)
predictions.head()
```

Out[17]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

チャレンジ中のコンペ

- University of Liverpool - Ion Switching
  細胞の電気シグナルを元にイオンチャンネルの開放数を予想するコンペ
- M5 Forecasting - Accuracy
  ウォルマートの商品毎の売り上げ個数を予想するコンペ

一晩、学習寝かせて、さっき見たら
スコア激下がりしてた、、、ミスったorz

所感

- (一部の)前処理、学習、推論が簡潔なコードで書ける
  とりあえずLightGBM学習回して
  interpret_model()で重要特徴量見るのとか瞬殺
- (当然ですが)EDAや特徴量エンジニアリングが大事
  でも、ある程度までは自動でそこそこ精度が上がる

- 初心者向けでも上級者向けでも無く、
  割り切った中級者向けな印象(後輩くんにはどうだろう🤔)

所感

- AutoMLに該当するかプチ議論があるっぽい
  Optunaが許容されるのであれば、
  こちらもセーフでは？と言う意見も。

ご清聴ありがとうございました🙃