Pertemuan 6

April 4, 2024

1 Gathering Data and Preprocessing

1.1 Import Library and Gathering Data

```
%pip install ipython-autotime

import pandas as pd
import numpy as np

%load_ext autotime

df = pd.read_json('Data pertemuan 5.json')
df.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	None	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	None	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	None	S

```
time: 47 ms (started: 2024-04-04 12:27:24 +07:00)
```

1.2 Preprocessing

```
X_numeric = df[['Age', 'Fare']]

X_categoric = df[['Sex', 'Cabin', 'Embarked', 'Pclass', 'SibSp', '
    Parch']]

y = df['Survived']
```

```
X_numeric.isna().sum()
```

```
Out[5]: Age     177
        Fare      0
        dtype: int64

        time: 0 ns (started: 2024-04-04 12:27:25 +07:00)
```

```
X_categoric.isna().sum()
```

```
Out[6]: Sex           0
        Cabin       687
        Embarked      2
        Pclass        0
        SibSp         0
        Parch         0
        dtype: int64

        time: 0 ns (started: 2024-04-04 12:27:25 +07:00)
```

```
mean = X_numeric.Age.mean()
X_numeric.Age = X_numeric.Age.fillna(mean)

X_categoric['Cabin'] = X_categoric['Cabin'].fillna('Other')

mode = X_categoric['Embarked'].mode()[0]
X_categoric['Embarked'].fillna(mode, inplace = True)

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse_output = False, handle_unknown = '
    ignore')

dummy = ohe.fit_transform(X_categoric)

X_encoded = pd.DataFrame(dummy, columns = ohe.
    get_feature_names_out())
X_encoded
```

Out[10]:

	Sex_female	Sex_male	Cabin_A10	Cabin_A14	Cabin_A16	Cabin_A19	Cabin_A20	Cabin_A23	Cabin_A24	Cabin_A26	...	SibSp_4	SibSp_5	SibSp_8	Pa
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
886	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
887	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
888	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
889	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
890	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

891 rows × 170 columns

```
time: 47 ms (started: 2024-04-04 12:27:26 +07:00)
```

```python
X = pd.concat([X_numeric, X_encoded], axis = 1)
X
```

Out[11]:

	Age	Fare	Sex_female	Sex_male	Cabin_A10	Cabin_A14	Cabin_A16	Cabin_A19	Cabin_A20	Cabin_A23	...	SibSp_4	SibSp_5	SibSp_8	Parch_
0	22.000000	7.2500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
1	38.000000	71.2833	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
2	26.000000	7.9250	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
3	35.000000	53.1000	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
4	35.000000	8.0500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
886	27.000000	13.0000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
887	19.000000	30.0000	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
888	29.699118	23.4500	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
889	26.000000	30.0000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1
890	32.000000	7.7500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1

891 rows × 172 columns

```
time: 46 ms (started: 2024-04-04 12:27:26 +07:00)
```

```python
X.isna().any().all()
```

Out[12]: False

```
time: 0 ns (started: 2024-04-04 12:27:26 +07:00)
```

2 Modelling

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.naive_bayes import BernoulliNB, GaussianNB,
    MultinomialNB
from sklearn.linear_model import LogisticRegression,
    LogisticRegressionCV
from sklearn.ensemble import HistGradientBoostingClassifier,
    GradientBoostingClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, f1_score,
    precision_score, recall_score, classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.2, random_state = 26)
```

Buatlah Fungsi untuk Menjalankan Fitting dan Evaluasi Secara Otomatis

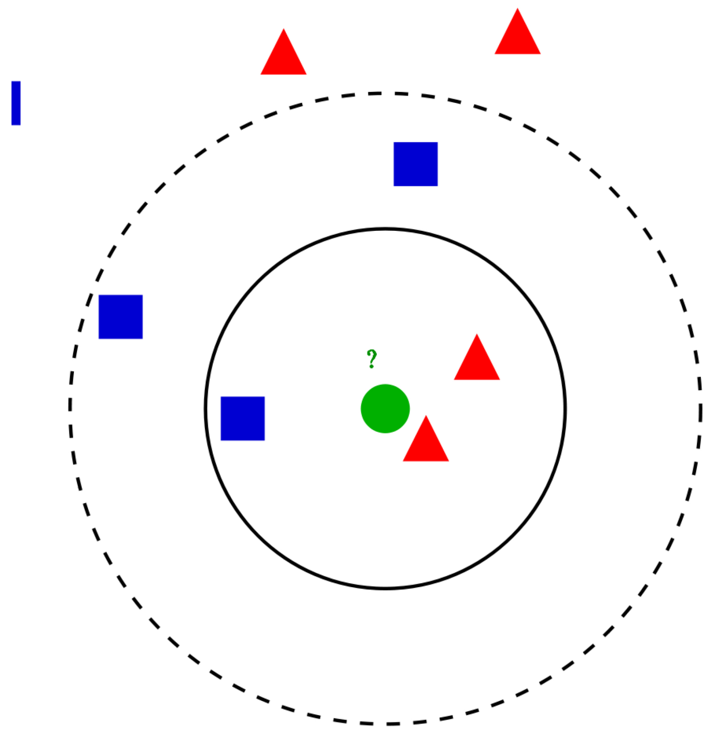
```
def train_and_evaluate(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = pd.DataFrame(model.predict(X_test), columns = ['
        y_pred'])

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    result = {'Accuracy': accuracy, 'Precision': precision, '
        Recall': recall, 'F1-score': f1}

    print(classification_report(y_test, y_pred))
    return y_pred, result
```

2.1 K-Nearest Neighbors Classifier



KNN adalah algoritma machine learning yang bekerja berdasarkan prinsip bahwa objek yang mirip cenderung berada dalam jarak yang dekat satu sama lain. Dengan kata lain, data yang memiliki karakteristik serupa akan cenderung saling bertetangga dalam ruang fitur (feature space).

Algoritma KNN mengasumsikan bahwa objek yang mirip akan berada dalam jarak yang dekat satu sama lain. KNN menggunakan seluruh data yang tersedia dalam pengambilan keputusan. Ketika ada data baru yang perlu diklasifikasikan, algoritma mengukur tingkat kemiripan atau fungsi jarak antara data baru tersebut dengan data yang sudah ada. Data baru kemudian ditempatkan dalam kelas yang paling banyak dimiliki oleh data tetangga terdekatnya.

```
clf = KNeighborsClassifier(n_neighbors = 3)
clf.fit(X_train, y_train)
y_pred = clf.predict(np.array(X_test))
print(classification_report(y_test, y_pred))
```

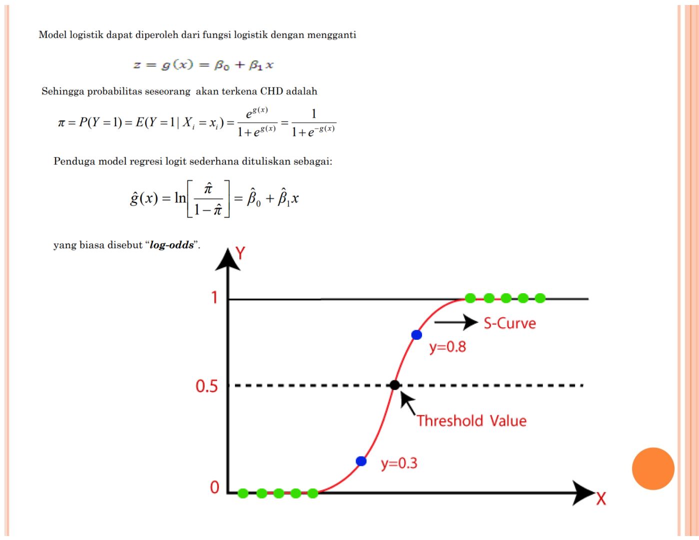
```
              precision    recall  f1-score   support

           0       0.76      0.78      0.77       116
           1       0.58      0.56      0.57        63

    accuracy                           0.70       179
   macro avg       0.67      0.67      0.67       179
weighted avg       0.70      0.70      0.70       179

time: 250 ms (started: 2024-04-04 12:27:30 +07:00)
```

2.2 Regresi Logistik



Regresi Logistik merupakan analisis regresi yang digunakan ketika variabel dependennya berupa biner, atau juga bisa disebut sebagai klasifikasi. Regresi logistik menghitung nilai probabilitas terjadinya kejadian pada variabel respons, kemudian mengkategorikannya berdasarkan *threshold* atau batas nilai, umumnya adalah 0.5. Module sklearn menyediakan dua jenis regresi logistik yakni LogisticRegression dan LogisticRegressionCV. Meski terdapat dua jenis fungsi, namun sejatinya kedua fungsi tersebut adalah sama.

```
y_pred, result = train_and_evaluate(LogisticRegression(max_iter =
    1000), X_train, X_test, y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       116
           1       0.76      0.67      0.71        63

    accuracy                           0.81       179
   macro avg       0.80      0.78      0.79       179
weighted avg       0.81      0.81      0.81       179

Out[20]: {'Accuracy': 0.8100558659217877,
          'Precision': 0.7636363636363637,
          'Recall': 0.6666666666666666,
          'F1-score': 0.711864406779661}

time: 250 ms (started: 2024-04-04 12:27:30 +07:00)
```

```
y_pred, result = train_and_evaluate(LogisticRegressionCV(max_iter
    = 10000), X_train, X_test, y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.86       116
           1       0.77      0.68      0.72        63

    accuracy                           0.82       179
   macro avg       0.80      0.79      0.79       179
weighted avg       0.81      0.82      0.81       179

Out[21]: {'Accuracy': 0.8156424581005587,
          'Precision': 0.7678571428571429,
          'Recall': 0.6825396825396826,
          'F1-score': 0.7226890756302521}

time: 4.86 s (started: 2024-04-04 12:27:31 +07:00)
```

2.3 Naive Bayes

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

A, B = events

$P(A|B)$ = probability of A given B is true

$P(B|A)$ = probability of B given A is true

$P(A), P(B)$ = the independent probabilities of A and B

Naive Bayes adalah metode atau algoritma klasifikasi yang didasarkan pada perhitungan probabilitas bersyarat yaitu Teorema Bayes. Metode ini secara *'naive'* menganggap bahwa setiap variabel tidak berhubungan satu sama lain/independen. Naive Bayes menghitung probabilitas dari setiap kondisi, kemudian memilih hasilnya berdasarkan nilai peluang yang paling besar.

Terdapat beberapa jenis klasifikasi Naive Bayes, yakni

- Bernoulli Naive Bayes
  utamanya digunakan ketika data yang dimiliki adalah data diskrit dan variabel berbentuk biner.
- Gaussian Naive Bayes
  digunakan ketika dimiliki data kontinu dan menggunakan distribusi normal.
- Multinomial Naive Bayes
  digunakan ketika didapati data nominal/diskrit.

```
y_pred, result = train_and_evaluate(BernoulliNB(), X_train, X_test
    , y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.81      0.81      0.81       116
           1       0.65      0.65      0.65        63

    accuracy                           0.75       179
   macro avg       0.73      0.73      0.73       179
weighted avg       0.75      0.75      0.75       179

Out[22]: {'Accuracy': 0.7541899441340782,
          'Precision': 0.6507936507936508,
          'Recall': 0.6507936507936508,
          'F1-score': 0.6507936507936508}

time: 47 ms (started: 2024-04-04 12:27:35 +07:00)
```

```
y_pred, result = train_and_evaluate(GaussianNB(), X_train, X_test,
    y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.67      0.95      0.78       116
           1       0.57      0.13      0.21        63

    accuracy                           0.66       179
   macro avg       0.62      0.54      0.50       179
weighted avg       0.63      0.66      0.58       179

Out[23]: {'Accuracy': 0.659217877094972,
          'Precision': 0.5714285714285714,
          'Recall': 0.12698412698412698,
          'F1-score': 0.2077922077922078}

time: 62 ms (started: 2024-04-04 12:27:36 +07:00)
```

```
y_pred, result = train_and_evaluate(MultinomialNB(), X_train,
    X_test, y_train, y_test)
```

```
result
```

```
              precision    recall  f1-score   support

           0       0.72      0.83      0.77       116
           1       0.57      0.41      0.48        63

    accuracy                           0.68       179
   macro avg       0.64      0.62      0.62       179
weighted avg       0.67      0.68      0.67       179
```

```
Out[24]: {'Accuracy': 0.6815642458100558,
          'Precision': 0.5652173913043478,
          'Recall': 0.4126984126984127,
          'F1-score': 0.47706422018348627}

         time: 47 ms (started: 2024-04-04 12:27:36 +07:00)
```

2.4 Gradient Boosting

```
y_pred, result = train_and_evaluate(GradientBoostingClassifier(),
    X_train, X_test, y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       116
           1       0.77      0.63      0.70        63

    accuracy                           0.80       179
   macro avg       0.79      0.77      0.78       179
weighted avg       0.80      0.80      0.80       179
```

```
Out[25]: {'Accuracy': 0.8044692737430168,
          'Precision': 0.7692307692307693,
          'Recall': 0.6349206349206349,
          'F1-score': 0.6956521739130435}

         time: 266 ms (started: 2024-04-04 12:27:36 +07:00)
```

2.5 Hist Gradient Boosting

```
y_pred, result = train_and_evaluate(HistGradientBoostingClassifier
    (), X_train, X_test, y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.83      0.86      0.85       116
           1       0.73      0.68      0.70        63

    accuracy                           0.80       179
   macro avg       0.78      0.77      0.78       179
weighted avg       0.80      0.80      0.80       179
```

```
Out[26]: {'Accuracy': 0.7988826815642458,
          'Precision': 0.7288135593220338,
          'Recall': 0.6825396825396826,
          'F1-score': 0.7049180327868853}

         time: 782 ms (started: 2024-04-04 12:27:36 +07:00)
```

2.6 CatBoost

```
y_pred, result = train_and_evaluate(CatBoost(), X_train, X_test,
    y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       116
           1       0.77      0.63      0.70        63

    accuracy                           0.80       179
   macro avg       0.79      0.77      0.78       179
weighted avg       0.80      0.80      0.80       179
```

```
Out[27]: {'Accuracy': 0.8044692737430168,
          'Precision': 0.7692307692307693,
          'Recall': 0.6349206349206349,
          'F1-score': 0.6956521739130435}

         time: 3.2 s (started: 2024-04-04 12:27:37 +07:00)
```

2.7 XGBoost

```
y_pred, result = train_and_evaluate(XGBClassifier(), X_train,
    X_test, y_train, y_test)
result
```

```
              precision    recall  f1-score   support

           0       0.83      0.84      0.83       116
           1       0.69      0.68      0.69        63

    accuracy                           0.78       179
   macro avg       0.76      0.76      0.76       179
weighted avg       0.78      0.78      0.78       179
```

```
Out[28]: {'Accuracy': 0.7821229050279329,
 'Precision': 0.6935483870967742,
 'Recall': 0.6825396825396826,
 'F1-score': 0.688}

time: 1.3 s (started: 2024-04-04 12:27:40 +07:00)
```

3 Modelling All Algorithm And Evaluate

3.1 Modelling All Algorithm

Membuat fungsi untuk train semua algoritma dan mengukur nilai kebaikannya

```
def all_model(list_model, X, y, test_size = 0.2, random_state =
    None):

     X_train, X_test, y_train, y_test = train_test_split(X, y,
         test_size = test_size, random_state = random_state)

     result = []
     for model in list_model:
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)

         accuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred)
         recall = recall_score(y_test, y_pred)
         f1 = f1_score(y_test, y_pred)

         hasil = {
             'Model' : type(model).__name__,
             'Accuracy' : accuracy,
             'F1 Score' : f1,
             'Precision' : precision,
             'Recall' : recall
         }
```

```
        result.append(hasil)

    result_all = pd.DataFrame(result)

    return result_all
```

Tentukan semua algoritma yang digunakan

```
list_of_model = [
    LogisticRegression(max_iter = 1000),
    LogisticRegressionCV(max_iter = 10000),
    GradientBoostingClassifier(random_state = 26),
    HistGradientBoostingClassifier(random_state = 26),
    CatBoostClassifier(random_state = 26, logging_level = 'Silent'
       ),
    XGBClassifier(random_state = 26),
    BernoulliNB(),
    GaussianNB(),
    MultinomialNB(),
    KNeighborsClassifier()
 ]
```

Panggil kembali fungsi dan urutkan berdasarkan nilai pengukuran

```
all_listed_model = all_model(list_of_model, X, y, test_size = 0.25)
all_listed_model
```

Out[32]:

	Model	Accuracy	F1 Score	Precision	Recall
0	LogisticRegression	0.834081	0.778443	0.812500	0.747126
1	LogisticRegressionCV	0.838565	0.783133	0.822785	0.747126
2	GradientBoostingClassifier	0.820628	0.736842	0.861538	0.643678
3	HistGradientBoostingClassifier	0.798206	0.727273	0.769231	0.689655
4	CatBoostClassifier	0.807175	0.726115	0.814286	0.655172
5	XGBClassifier	0.775785	0.695122	0.740260	0.655172
6	BernoulliNB	0.775785	0.719101	0.703297	0.735632
7	GaussianNB	0.636771	0.181818	0.750000	0.103448
8	MultinomialNB	0.654709	0.476190	0.583333	0.402299
9	KNeighborsClassifier	0.695067	0.575000	0.630137	0.528736

time: 8.88 s (started: 2024-04-04 12:27:41 +07:00)

```
all_listed_model.sort_values('Accuracy', ascending = False)
```

Out[33]:

	Model	Accuracy	F1 Score	Precision	Recall
1	LogisticRegressionCV	0.838565	0.783133	0.822785	0.747126
0	LogisticRegression	0.834081	0.778443	0.812500	0.747126
2	GradientBoostingClassifier	0.820628	0.736842	0.861538	0.643678
4	CatBoostClassifier	0.807175	0.726115	0.814286	0.655172
3	HistGradientBoostingClassifier	0.798206	0.727273	0.769231	0.689655
5	XGBClassifier	0.775785	0.695122	0.740260	0.655172
6	BernoulliNB	0.775785	0.719101	0.703297	0.735632
9	KNeighborsClassifier	0.695067	0.575000	0.630137	0.528736
8	MultinomialNB	0.654709	0.476190	0.583333	0.402299
7	GaussianNB	0.636771	0.181818	0.750000	0.103448

time: 16 ms (started: 2024-04-04 12:27:50 +07:00)

3.2 Modelling All Algorithm with Cross Validation

Membuat fungsi untuk train semua algoritma dan melakukan cross validation

```
from sklearn.model_selection import KFold, cross_validate

def all_model_cv(list_model, metric_list, X, y, random_state =
    None, n_split = 5):
    kfold = KFold(n_splits = n_split, shuffle = True, random_state
        = random_state)

    result = []
    for model in list_model:

        score = []
        for metric in metric_list:
            metric_score = cross_validate(model, X, y, cv = kfold,
                scoring = metric)
            score.append(metric_score['test_score'].mean())

        result.append(score)

    result_all = pd.DataFrame(result, columns = metric_list)

    return result_all
```

Tentukan semua metric yang digunakan

```
list_of_metric = [
     'accuracy',
     'f1',
     'recall',
     'precision',
     'roc_auc',
     'neg_log_loss',
     'f1_weighted'
 ]
```

Panggil kembali fungsinya

```
all_listed_model_cv = all_model_cv(list_of_model, list_of_metric,
   X, y, n_split = 5, random_state = 26)
all_listed_model_cv
```

Out[60]:

	accuracy	f1	recall	precision	roc_auc	neg_log_loss	f1_weighted
0	0.813703	0.746076	0.718723	0.778574	0.853721	-0.448499	0.812099
1	0.814814	0.748621	0.722508	0.779578	0.850654	-0.452318	0.813423
2	0.819321	0.737133	0.666464	0.831668	0.858491	-0.425909	0.814368
3	0.813709	0.741015	0.699632	0.795151	0.857809	-0.501510	0.810990
4	0.822692	0.740064	0.662963	0.842620	0.862624	-0.420016	0.817323
5	0.803616	0.731342	0.704249	0.767378	0.852238	-0.554905	0.801747
6	0.784546	0.719451	0.725445	0.715752	0.841989	-0.602223	0.784840
7	0.658810	0.263274	0.159749	0.761492	0.797573	-12.179865	0.581377
8	0.696987	0.550014	0.485216	0.636849	0.746447	-3.043764	0.686861
9	0.714921	0.603375	0.569210	0.644046	0.739066	-2.764644	0.711325

time: 6min 24s (started: 2024-04-04 13:18:08 +07:00)

Tambahkan kolom berisikan nama algoritma dan modifikasi nama kolom supaya lebih nyaman dilihat

```
column_name = [metric.capitalize() for metric in list_of_metric]
model_name = [type(model).__name__ for model in list_of_model]

all_listed_model_cv.columns = column_name
all_listed_model_cv.insert(0, 'Model', value = model_name)
all_listed_model_cv
```

Out[62]:

	Model	Accuracy	F1	Recall	Precision	Roc_auc	Neg_log_loss	F1_weighted
0	LogisticRegression	0.813703	0.746076	0.718723	0.778574	0.853721	-0.448499	0.812099
1	LogisticRegressionCV	0.814814	0.748621	0.722508	0.779578	0.850654	-0.452318	0.813423
2	GradientBoostingClassifier	0.819321	0.737133	0.666464	0.831668	0.858491	-0.425909	0.814368
3	HistGradientBoostingClassifier	0.813709	0.741015	0.699632	0.795151	0.857809	-0.501510	0.810990
4	CatBoostClassifier	0.822692	0.740064	0.662963	0.842620	0.862624	-0.420016	0.817323
5	XGBClassifier	0.803616	0.731342	0.704249	0.767378	0.852238	-0.554905	0.801747
6	BernoulliNB	0.784546	0.719451	0.725445	0.715752	0.841989	-0.602223	0.784840
7	GaussianNB	0.658810	0.263274	0.159749	0.761492	0.797573	-12.179865	0.581377
8	MultinomialNB	0.696987	0.550014	0.485216	0.636849	0.746447	-3.043764	0.686861
9	KNeighborsClassifier	0.714921	0.603375	0.569210	0.644046	0.739066	-2.764644	0.711325

```
time: 15 ms (started: 2024-04-04 13:24:32 +07:00)
```