Tugas 1 Terdapat dataset mushroom. Berdasarkan dataset yang tersebut, bandingkan peforma antara algoritma Decision Tree dan RandomForest. Gunakan tunning hyperparameter untuk mendapatkan parameter dan akurasi yang terbaik.

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # import DT
from sklearn.ensemble import RandomForestClassifier # import 
RandomForest
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

Langkah 1: Memuat Dataset

```
# Load data
df = pd.read_csv('data/mushrooms.csv')

df.head()
```

```
  class cap-shape cap-surface cap-color bruises odor gill-
attachment  \
0     p         x           s         n       t    p               f

1     e         x           s         y       t    a               f

2     e         b           s         w       t    l               f

3     p         x           y         w       t    p               f

4     e         x           s         g       f    n               f


  gill-spacing gill-size gill-color  ... stalk-surface-below-ring  \
0            c         n          k  ...                        s
1            c         b          k  ...                        s
2            c         b          n  ...                        s
3            c         n          n  ...                        s
4            w         b          k  ...                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-
color  \
0                      w                      w         p          w

1                      w                      w         p          w

2                      w                      w         p          w

3                      w                      w         p          w

4                      w                      w         p          w
```

```
  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

```python
# Cek kolom null
df.isnull().sum()
```

```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

```python
# Seleksi fitur

# Slice dataframe mulai dari kolom 'cap-shape' sampai 'habitat'
X = df.iloc[:,0:-1]
y = df['bruises']
y = y.map({'M':1, 'B':0}) # Encode label

# Cek jumlah fitur dan instance
X.shape
```

```
(8124, 22)
```

Langkah Split Data

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

Langkah Decision tree Model

```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import load_iris

# Load dataset
data = load_iris()
X = data.data
y = data.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Decision Tree model
dt = DecisionTreeClassifier(random_state=42)

# Hyperparameter tuning untuk Decision Tree
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# GridSearchCV untuk menemukan parameter terbaik
grid_search_dt = GridSearchCV(estimator=dt, param_grid=param_grid_dt,
cv=5, n_jobs=-1, verbose=1)
grid_search_dt.fit(X_train, y_train)

# Hasil terbaik dari Decision Tree
best_dt = grid_search_dt.best_estimator_
y_pred_dt = best_dt.predict(X_test)

# Akurasi dan laporan klasifikasi
accuracy_dt = accuracy_score(y_test, y_pred_dt) * 100  # Akurasi dalam
persen
print(f"Best Decision Tree Parameters: {grid_search_dt.best_params_}")
print(f"Decision Tree Accuracy: {accuracy_dt:.2f}%")
print(classification_report(y_test, y_pred_dt))
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Decision Tree Parameters: {'criterion': 'entropy', 'max_depth':
10, 'min_samples_leaf': 4, 'min_samples_split': 2}
Decision Tree Accuracy: 100.00%
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Langkah Random Forest Model

```
# Random Forest
rf = RandomForestClassifier(random_state=42)

# Hyperparameter tuning untuk Random Forest
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'criterion': ['gini', 'entropy'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf,
cv=2, n_jobs=-1, verbose=1)
grid_search_rf.fit(X_train, y_train)

# Hasil terbaik dari Random Forest
best_rf = grid_search_rf.best_estimator_
y_pred_rf = best_rf.predict(X_test)

# Akurasi dan laporan klasifikasi
accuracy_rf = accuracy_score(y_test, y_pred_rf) * 100  # Akurasi dalam
persen
print(f"Best Random Forest Parameters: {grid_search_rf.best_params_}")
print(f"Random Forest Accuracy: {accuracy_rf:.2f}%")
print(classification_report(y_test, y_pred_rf))

Fitting 2 folds for each of 216 candidates, totalling 432 fits
Best Random Forest Parameters: {'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Random Forest Accuracy: 100.00%
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

TUGAS 2

Terdapat dataset mushroom. Berdasarkan dataset tersebut, bandingkan peforma antara algoritma Decision Tree dan AdaBoost. Gunakan tunning hyperparameter untuk mendapatkan parameter dan akurasi yang terbaik.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
```

Langkah Memuat Dataset

```
# Load data
df = pd.read_csv('data/mushrooms.csv')

df.head()

  class cap-shape cap-surface cap-color bruises odor gill-
attachment  \
0     p         x           s         n       t    p               f

1     e         x           s         y       t    a               f

2     e         b           s         w       t    l               f

3     p         x           y         w       t    p               f

4     e         x           s         g       f    n               f


  gill-spacing gill-size gill-color  ... stalk-surface-below-ring  \
0            c         n          k  ...                        s
1            c         b          k  ...                        s
2            c         b          n  ...                        s
3            c         n          n  ...                        s
4            w         b          k  ...                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-
```

```
color  \
0                       w                       w         p          w

1                       w                       w         p          w

2                       w                       w         p          w

3                       w                       w         p          w

4                       w                       w         p          w


  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

Langkah Encoding Data

```
import pandas as pd
from sklearn.datasets import load_iris  # Misal data berasal dari
dataset sklearn
from sklearn.preprocessing import LabelEncoder

# Load dataset (misal iris dataset sebagai contoh)
data_bunch = load_iris()  # Menghasilkan Bunch object

# Convert Bunch menjadi DataFrame
data = pd.DataFrame(data_bunch.data, columns=data_bunch.feature_names)

# Misalkan menambahkan kolom kategori 'class'
data['class'] = pd.Categorical.from_codes(data_bunch.target,
data_bunch.target_names)

# Menggunakan Label Encoding untuk semua kolom
label_encoder = LabelEncoder()
for column in data.columns:
    data[column] = label_encoder.fit_transform(data[column])

# Split data menjadi fitur dan label
X = data.drop('class', axis=1)
y = data['class']
```

Langkah Split Data

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

Langkah Decision Tree Model

```
# Decision Tree
dt = DecisionTreeClassifier(random_state=42)

# Hyperparameter tuning untuk Decision Tree
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_dt = GridSearchCV(estimator=dt, param_grid=param_grid_dt,
cv=5, n_jobs=-1, verbose=1)
grid_search_dt.fit(X_train, y_train)

# Hasil terbaik dari Decision Tree
best_dt = grid_search_dt.best_estimator_
y_pred_dt = best_dt.predict(X_test)

# Akurasi dan laporan klasifikasi
accuracy_dt = accuracy_score(y_test, y_pred_dt) * 100  # Akurasi dalam
persen
print(f"Best Decision Tree Parameters: {grid_search_dt.best_params_}")
print(f"Decision Tree Accuracy: {accuracy_dt:.2f}%")
print(classification_report(y_test, y_pred_dt))

Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Decision Tree Parameters: {'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 1, 'min_samples_split': 10}
Decision Tree Accuracy: 100.00%
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Langkah AdaBoost Model

```
# AdaBoost
ada = AdaBoostClassifier()

# Hyperparameter tuning untuk AdaBoost
param_grid_ada = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1, 10]
}

grid_search_ada = GridSearchCV(estimator=ada,
param_grid=param_grid_ada, cv=5, n_jobs=-1, verbose=1)
grid_search_ada.fit(X_train, y_train)

# Hasil terbaik dari AdaBoost
best_ada = grid_search_ada.best_estimator_
y_pred_ada = best_ada.predict(X_test)

# Akurasi dan laporan klasifikasi
accuracy_ada = accuracy_score(y_test, y_pred_ada) * 100  # Akurasi
dalam persen
print(f"Best AdaBoost Parameters: {grid_search_ada.best_params_}")
print(f"AdaBoost Accuracy: {accuracy_ada:.2f}%")
print(classification_report(y_test, y_pred_ada))
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best AdaBoost Parameters: {'learning_rate': 1, 'n_estimators': 100}
AdaBoost Accuracy: 100.00%
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


C:\Users\M. Rofiq Aulia\AppData\Roaming\Python\Python312\site-
packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The
SAMME.R algorithm (the default) is deprecated and will be removed in
1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

Evaluasi dan Perbandingan

```
# Output Akhir
print(f"Decision Tree Accuracy: {accuracy_dt:.2f}%")
print(f"AdaBoost Accuracy: {accuracy_ada:.2f}%")
```

```
Decision Tree Accuracy: 100.00%
AdaBoost Accuracy: 100.00%
```

TUGAS 3

engan menggunakan dataset diabetes, buatlah ensemble voting dengan algoritma

1. Logistic Regression
2. SVM kernel polynomial
3. Decission Tree

Anda boleh melakukan eksplorasi dengan melakukan tunning hyperparameter

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load data
df = pd.read_csv('data/diabetes.csv')

df.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Cek Kolom Null

```python
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
feature_columns = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
for column in feature_columns:
    print("============================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] ==
0])}")
```

```
============================================
Pregnancies ==> Missing zeros : 111
============================================
Glucose ==> Missing zeros : 5
============================================
BloodPressure ==> Missing zeros : 35
============================================
SkinThickness ==> Missing zeros : 227
============================================
Insulin ==> Missing zeros : 374
============================================
BMI ==> Missing zeros : 11
============================================
DiabetesPedigreeFunction ==> Missing zeros : 0
============================================
Age ==> Missing zeros : 0
```

Input Nilai 0 dengan Mean

```
# Import SimpleImputer dari sklearn
from sklearn.impute import SimpleImputer

# Inisialisasi SimpleImputer
fill_values = SimpleImputer(missing_values=0, strategy="mean",
copy=False)

# Menerapkan imputasi pada kolom fitur
df[feature_columns] = fill_values.fit_transform(df[feature_columns])
```

Split Data

```
X = df[feature_columns]
y = df.Outcome

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

Standarisasi Fitur

```
# Import StandardScaler dari sklearn
from sklearn.preprocessing import StandardScaler

# Inisialisasi StandardScaler
sc = StandardScaler()

# Standarisasi pada fitur di X_train dan X_test
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

Model

```
# Import model yang diperlukan dari sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# Inisialisasi model
log_reg = LogisticRegression(random_state=42)
svc = SVC(kernel='poly', probability=True, random_state=42)  # SVM
dengan kernel polynomial
dt = DecisionTreeClassifier(random_state=42)
```

Logistik regeression w/ Hyperparameter Tunning

```
# Definisikan hyperparameter Logistic Regression
param_grid_logreg = {
    'C': [0.1, 1, 10, 100],  # Regularization strength
    'solver': ['liblinear', 'lbfgs'],  # Solver untuk optimasi
    'max_iter': [100, 200, 500]  # Jumlah iterasi maksimum
}

# GridSearchCV untuk Logistic Regression
grid_search_logreg = GridSearchCV(estimator=log_reg,
param_grid=param_grid_logreg, cv=5, verbose=1, n_jobs=-1)

# Fit model Logistic Regression
grid_search_logreg.fit(X_train_std, y_train)

# Prediksi pada data test
y_pred_logreg = grid_search_logreg.best_estimator_.predict(X_test_std)
```

```
# Evaluasi Logistic Regression
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print(f"Accuracy (Logistic Regression): {accuracy_logreg*100:.2f}%")
print(f"Classification Report (Logistic Regression):\
n{classification_report(y_test, y_pred_logreg)}")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Accuracy (Logistic Regression): 73.59%
Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       151
           1       0.63      0.59      0.61        80

    accuracy                           0.74       231
   macro avg       0.71      0.70      0.70       231
weighted avg       0.73      0.74      0.73       231
```

SVM Polynomial w/Hyperparameter Tunning

```
param_grid_svc = {
    'C': [0.1, 1, 10],  # Regularisasi
    'degree': [2, 3, 4],  # Derajat polynomial
    'gamma': ['scale', 'auto'],  # Kernel coefficient
}

# GridSearchCV untuk SVM
grid_search_svc = GridSearchCV(estimator=svc,
param_grid=param_grid_svc, cv=5, verbose=1, n_jobs=-1)

# Fit model SVM
grid_search_svc.fit(X_train_std, y_train)

# Prediksi pada data test
y_pred_svc = grid_search_svc.best_estimator_.predict(X_test_std)

# Evaluasi SVM
accuracy_svc = accuracy_score(y_test, y_pred_svc)
print(f"Accuracy (SVM): {accuracy_svc*100:.2f}%")
print(f"Classification Report (SVM):\n{classification_report(y_test,
y_pred_svc)}")
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Accuracy (SVM): 69.70%
Classification Report (SVM):
              precision    recall  f1-score   support

           0       0.72      0.88      0.79       151
```

```
           1       0.61      0.35      0.44        80

    accuracy                           0.70       231
   macro avg       0.66      0.62      0.62       231
weighted avg       0.68      0.70      0.67       231
```

Desicion Tree w/Hyperparameter Tunning

```
param_grid_dt = {
    'max_depth': [3, 5, 7, 10],  # Maksimal kedalaman pohon
    'min_samples_split': [2, 5, 10],  # Minimum jumlah sampel untuk
split
    'min_samples_leaf': [1, 2, 4]  # Minimum jumlah sampel di setiap
daun
}

# GridSearchCV untuk Decision Tree
grid_search_dt = GridSearchCV(estimator=dt, param_grid=param_grid_dt,
cv=5, verbose=1, n_jobs=-1)

# Fit model Decision Tree
grid_search_dt.fit(X_train_std, y_train)

# Prediksi pada data test
y_pred_dt = grid_search_dt.best_estimator_.predict(X_test_std)

# Evaluasi Decision Tree
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Accuracy (Decision Tree): {accuracy_dt*100:.2f}%")
print(f"Classification Report (Decision Tree):\
n{classification_report(y_test, y_pred_dt)}")

Fitting 5 folds for each of 36 candidates, totalling 180 fits
Accuracy (Decision Tree): 74.46%
Classification Report (Decision Tree):
              precision    recall  f1-score   support

           0       0.79      0.82      0.81       151
           1       0.64      0.60      0.62        80

    accuracy                           0.74       231
   macro avg       0.72      0.71      0.71       231
weighted avg       0.74      0.74      0.74       231


c:\Python312\Lib\site-packages\numpy\ma\core.py:2881: RuntimeWarning:
invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

Ensemble Voting

```
# Import modul yang diperlukan
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score, classification_report

# Definisikan model dengan hyperparameter terbaik dari GridSearch
log_reg_best = grid_search_logreg.best_estimator_  # Logistic
Regression terbaik
svc_best = grid_search_svc.best_estimator_         # SVM terbaik
dt_best = grid_search_dt.best_estimator_           # Decision Tree
terbaik

# Ensemble Voting dengan soft voting
voting_clf = VotingClassifier(estimators=[('lr', log_reg_best),
('svc', svc_best), ('dt', dt_best)], voting='soft')

# Fit model pada data train
voting_clf.fit(X_train_std, y_train)

# Prediksi pada data test
y_pred_voting = voting_clf.predict(X_test_std)

# Evaluasi Ensemble Voting
accuracy_voting = accuracy_score(y_test, y_pred_voting)
print(f"Accuracy (Ensemble Voting): {accuracy_voting*100:.2f}%")
print(f"Classification Report (Ensemble Voting):\
n{classification_report(y_test, y_pred_voting)}")
```

```
Accuracy (Ensemble Voting): 76.19%
Classification Report (Ensemble Voting):
              precision    recall  f1-score   support

           0       0.79      0.86      0.83       151
           1       0.69      0.57      0.63        80

    accuracy                           0.76       231
   macro avg       0.74      0.72      0.73       231
weighted avg       0.76      0.76      0.76       231
```

PRAKTIKUM 1

Bagging dengan RandomForest Pada kasus ini kita akan menggunakan salah satu metode bagging yaitu RandomForest untuk mengklasifikasikan jenis tumor. Dalam latihan ini Anda akan melakukan training dengan data Wisconsin Breast Cancer Dataset dari UCI machine learning repository. Latihan ini akan melakukan prediksi memprediksi apakah tumor ganas atau jinak.

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # import DT
from sklearn.ensemble import RandomForestClassifier # import
RandomForest
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

Persiapan Data

```
# Load data
df = pd.read_csv('data/wbc.csv')

df.head()

         id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
0    842302         M        17.99         10.38          122.80
1001.0
1    842517         M        20.57         17.77          132.90
1326.0
2  84300903         M        19.69         21.25          130.00
1203.0
3  84348301         M        11.42         20.38           77.58
386.1
4  84358402         M        20.29         14.34          135.10
1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   ...  texture_worst  perimeter_worst  area_worst
```

```
smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622

1  ...          23.41           158.80      1956.0            0.1238

2  ...          25.53           152.50      1709.0            0.1444

3  ...          26.50            98.87       567.7            0.2098

4  ...          16.67           152.20      1575.0            0.1374


   compactness_worst  concavity_worst  concave points_worst
symmetry_worst  \
0             0.6656           0.7119                0.2654
0.4601
1             0.1866           0.2416                0.1860
0.2750
2             0.4245           0.4504                0.2430
0.3613
3             0.8663           0.6869                0.2575
0.6638
4             0.2050           0.4000                0.1625
0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

```
# Cek kolom null
df.isnull().sum()
```

```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
```

```
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

```
# Seleksi fitur

# Slice dataframe mulai dari kolom 'radius_mean' sampai
'fractal_dimension_worst'
X = df.iloc[:,3:-1]
y = df['diagnosis']
y = y.map({'M':1, 'B':0}) # Encode label

# Cek jumlah fitur dan instance
X.shape
```

```
(569, 29)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)

# Secara default, DecisionTreeClassifier dari scikit-learn akan
menggunakan nilai "Gini" untuk kriteria
# Terdapat beberapa "hyperparamater" yang dapat digunakan. Silahka
baca dokumentasi
# Pada kasus ini kita akan menggunakan parameter default
dt = DecisionTreeClassifier()

# Sesuaikan dt ke set training
dt.fit(X_train, y_train)

# Memprediksi label set test
```

```
y_pred_dt = dt.predict(X_test)

#  menghitung set accuracy
acc_dt = accuracy_score(y_test, y_pred_dt)
print("Test set accuracy: {:.2f}".format(acc_dt))
print(f"Test set accuracy: {acc_dt}")

Test set accuracy: 0.96
Test set accuracy: 0.956140350877193

# Pada kasus kali ini kita akan menggunakan estimator pada
RandomForest
# Untuk detail parameter (hyperparameter) silahkan cek dokumentasi

rf = RandomForestClassifier(n_estimators=10, random_state=1)

# Sesuaikan dt ke set training
rf.fit(X_train, y_train)

# Memprediksi label set test
y_pred_rf = rf.predict(X_test)

#  menghitung set accuracy
acc_rf = accuracy_score(y_test, y_pred_rf)
print("Test set accuracy: {:.2f}".format(acc_rf))
print(f"Test set accuracy: {acc_rf}")

Test set accuracy: 0.96
Test set accuracy: 0.956140350877193
```

PRAKTIKUM 2

Boosting dengan AdaBoost Pada kasus ini kita akan menggunakan salah satu metode boosting yaitu AdaBoost untuk mengklasifikasikan jenis bunga Iris. Dalam latihan ini kita akan menggunakan dataset Iris yang sangat lazim digunakan. Latihan ini akan melakukan prediksi memprediksi 3 jenis bunga Iris yaitu, Iris Setosa, Iris Versicolor, dan Iris Virginica berdasarkan panjang dan lebar sepal dan petal.

Import Library

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # import DT
from sklearn.ensemble import AdaBoostClassifier # import AdaBoost
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder # Kebutuhan encoding
label

# Load data
df = pd.read_csv('data/iris.csv')
```

```
df.head()

   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
0   1            5.1           3.5            1.4           0.2  Iris-
setosa
1   2            4.9           3.0            1.4           0.2  Iris-
setosa
2   3            4.7           3.2            1.3           0.2  Iris-
setosa
3   4            4.6           3.1            1.5           0.2  Iris-
setosa
4   5            5.0           3.6            1.4           0.2  Iris-
setosa

# Cek kolom null
df.isnull().sum()

Id                 0
SepalLengthCm      0
SepalWidthCm       0
PetalLengthCm      0
PetalWidthCm       0
Species            0
dtype: int64

# Seleksi fitur
X = df.iloc[:,2:-1]
y = df['Species']

# encode label
ec = LabelEncoder()
y = ec.fit_transform(y)

# Cek jumlah fitur dan instance
print(X.shape)

# Cek label
print(y)

(150, 3)
[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 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 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2]
```

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)

# Secara default, DecisionTreeClassifier dari scikit-learn akan
menggunakan nilai "Gini" untuk kriteria
# Terdapat beberapa "hyperparamater" yang dapat digunakan. Silahka
baca dokumentasi
# Pada kasus ini kita akan menggunakan parameter default
dt = DecisionTreeClassifier()

# Sesuaikan dt ke set training
dt.fit(X_train, y_train)

# Memprediksi label set test
y_pred_dt = dt.predict(X_test)

#  menghitung set accuracy
acc_dt = accuracy_score(y_test, y_pred_dt)
print("Test set accuracy: {:.2f}".format(acc_dt))
print(f"Test set accuracy: {acc_dt}")
```

```
Test set accuracy: 0.97
Test set accuracy: 0.9666666666666667
```

```python
# Pada kasus kali ini kita akan menggunakan estimator pada AdaBoost
# Untuk detail parameter (hyperparameter) silahkan cek dokumentasi

ada = AdaBoostClassifier(n_estimators=2)

# Sesuaikan dt ke set training
ada.fit(X_train, y_train)

# Memprediksi label set test
y_pred_ada = ada.predict(X_test)

#  menghitung set accuracy
acc_ada = accuracy_score(y_test, y_pred_ada)
print("Test set accuracy: {:.2f}".format(acc_ada))
print(f"Test set accuracy: {acc_ada}")
```

```
Test set accuracy: 0.97
Test set accuracy: 0.9666666666666667
```

```
C:\Users\M. Rofiq Aulia\AppData\Roaming\Python\Python312\site-
packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The
SAMME.R algorithm (the default) is deprecated and will be removed in
1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

PRAKTIKUM 3

Stacking Lengkapi bagian berikut dengan data sesuai tugas, dan tentukan perbedaan nilai akurasi antara Random Forest, Adaboost, dan Stacking

```
from sklearn.ensemble import RandomForestClassifier,
StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier



layer_one_estimators = [
                        ('rf_1',
RandomForestClassifier(n_estimators=10, random_state=42)),
                        ('knn_1', KNeighborsClassifier(n_neighbors=5))

                       ]
layer_two_estimators = [
                        ('dt_2', DecisionTreeClassifier()),
                        ('rf_2',
RandomForestClassifier(n_estimators=50, random_state=42)),
                       ]
layer_two = StackingClassifier(estimators=layer_two_estimators,
final_estimator=LogisticRegression())


clf = StackingClassifier(estimators=layer_one_estimators,
final_estimator=layer_two)

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
random_state=42)
clf.fit(X_train, y_train).score(X_test, y_test)

0.868421052631579
```

PRAKTIKUM 4

Stacking dengan Voting Pada kasus ini kita akan menggunakan salah satu metode stacking yaitu voting untuk mengklasifikasikan pasien penderita diabetes dengan beberapa ciri. Pasien akan di klasifikasikan menjadi pasien menderita diabetes (1) dan tidak menderita diabetes (0). Pertama-tama, kita akan menggunakan beberapa algoritma klasifikasi secara terpisah, yaitu Naive Bayes, SVM Linier, dan SVM RBF. Setelah itu, kita akan menggabungkan performa dari 3 algoritma tersebut dengan menggunakan metode ensemble voting.

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB # import Naive Bayes model
```

```python
Gaussian (asumsi data terdistribusi normal)
from sklearn.svm import SVC # import SVM classifier
from sklearn.ensemble import VotingClassifier # import model Voting
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

Persiapan Data

```python
# Load Data

dbt = pd.read_csv('data/diabetes.csv')

dbt.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```python
# Cek nama kolom
dbt.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```python
# Cek kolom null
dbt.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
```

```
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64

# Pada kasus ini, agak tidak masuk akal jika beberapa parameter
bernilai 0
# sebagai contoh adalah nilai 'Glucose', 'BloodPlessure' ataupun
'Insulin'.
# Sekecil apapun nilainya, setiap manusia yang hidup pasti miliki
nilai-nilai tersebut

# Kita akan manipulasi nilai yang 0 dengan melakukan 'imputasi' atau
mengganti nilainya dengan nilai sintetis
# Pada kasus ini, kita akan menggunakan nilai mean

# Cek kolom neng nilai 0
feature_columns = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
for column in feature_columns:
    print("============================================")
    print(f"{column} ==> Missing zeros : {len(dbt.loc[dbt[column] ==
0])}")

============================================
Pregnancies ==> Missing zeros : 111
============================================
Glucose ==> Missing zeros : 5
============================================
BloodPressure ==> Missing zeros : 35
============================================
SkinThickness ==> Missing zeros : 227
============================================
Insulin ==> Missing zeros : 374
============================================
BMI ==> Missing zeros : 11
============================================
DiabetesPedigreeFunction ==> Missing zeros : 0
============================================
Age ==> Missing zeros : 0

# Impute nilai 0 dengan mean
from sklearn.impute import SimpleImputer

fill_values = SimpleImputer(missing_values=0, strategy="mean",
copy=False)

dbt[feature_columns] = fill_values.fit_transform(dbt[feature_columns])
```

Split data training dan testing

```
X = dbt[feature_columns]
y = dbt.Outcome

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

Training dengan GaussianNB Standarisasi Fitur

```
# Karena asumsi Gaussian NB adalah data terdistribusi secara normal,
# maka kita perlu melakukan standarisasi

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

# Standarisasi pada fitur di X_train dan X_test
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

Training dan Evaluasi

```
# Buat obyek GaussianNB
gnb_std = GaussianNB()

# Fit dengan data yang telah di standarisasi
gnb_std.fit(X_train_std, y_train)

# Prediksi dengan data test
y_pred_gnb = gnb_std.predict(X_test_std)

# Evaluasi akurasi testing data
acc_gnb = accuracy_score(y_test, y_pred_gnb)

# Print hasil evaluasi
print("Test set accuracy: {:.2f}".format(acc_gnb))
print(f"Test set accuracy: {acc_gnb}")

Test set accuracy: 0.74
Test set accuracy: 0.7359307359307359
```

Training dengan SVM Linier

```
# Model SVM linier tanpa tunnning hyperparameter
svm_lin = SVC(kernel='linear')

# Fit ke model
svm_lin.fit(X_train_std, y_train)

# Prediksi
```

```
y_pred_svm_lin = svm_lin.predict(X_test_std)

# Evaluasi akurasi testing data
acc_svm_lin = accuracy_score(y_test, y_pred_svm_lin)

# Print hasil evaluasi
print("Test set accuracy: {:.2f}".format(acc_svm_lin))
print(f"Test set accuracy: {acc_svm_lin}")

Test set accuracy: 0.74
Test set accuracy: 0.7402597402597403
```

Training dengan SVM RBF

```
# Model SVM RBF tanpa tunnning hyperparameter
svm_rbf = SVC(kernel='rbf')

# Fit ke model
svm_rbf.fit(X_train_std, y_train)

# Prediksi
y_pred_svm_rbf = svm_rbf.predict(X_test_std)

# Evaluasi akurasi testing data
acc_svm_rbf = accuracy_score(y_test, y_pred_svm_rbf)

# Print hasil evaluasi
print("Test set accuracy: {:.2f}".format(acc_svm_rbf))
print(f"Test set accuracy: {acc_svm_rbf}")

Test set accuracy: 0.72
Test set accuracy: 0.7229437229437229
```

Training dan Voting

```
# Definisikan algoritma yang akan digunakan untuk voting

clf1 = GaussianNB()
clf2 = SVC(kernel='linear')
clf3 = SVC(kernel='rbf', probability=True)

# model hard voting
voting = VotingClassifier(estimators=[('GaussianNB', clf1), ('SVM-
LIN', clf2), ('SVM-RBF', clf3)], voting='hard')

# Fit model
voting.fit(X_train_std, y_train)

# Prediksi
y_pred_vt1 = voting.predict(X_test_std)
```

```python
# Evaluasi akurasi testing data
acc_vt1 = accuracy_score(y_test, y_pred_vt1)

# Print hasil evaluasi
print('Voting Hard')
print("Test set accuracy: {:.2f}".format(acc_vt1))
print(f"Test set accuracy: {acc_vt1}")
```

```
Voting Hard
Test set accuracy: 0.74
Test set accuracy: 0.7402597402597403
```