HANDBOOK
MACHINE LEARNING

Studi Kasus Data Stunting di Indonesia

Gustriza Erda, Anne Mudya Yolanda, Arisman Adnan

HANDBOOK MACHINE LEARNING

Gustriza Erda
Anne Mudya Yolanda
Arisman Adnan

GETPRESS INDONESIA

HANDBOOK MACHINE LEARNING

Penulis : Gustriza Erda
Anne Mudya Yolanda
Arisman Adnan

ISBN : 978-623-125-542-6

Editor : Mila Sari, M.Si.
Desain Sampul dan Tata Letak : Tri Putri Wahyuni, S.Pd.

PENERBIT GET PRESS INDONESIA
Anggota IKAPI No. 033/SBA/2022
Jl. Palarik RT 01 RW 06 Kelurahan Air Pacah
Kecamatan Koto Tangah Padang Sumatera Barat

website: www.getpress.co.id
email: adm.getpress@gmail.com

Cetakan pertama, Desember 2024

KATA PENGANTAR

Puji syukur ke hadirat Tuhan Yang Maha Kuasa, yang telah memberikan rahmat-Nya sehingga buku *Handbook Machine Learning* ini dapat terselesaikan dengan baik. *Machine learning* merupakan cabang kecerdasan buatan yang telah menjadi salah satu teknologi revolusioner dalam berbagai bidang, dari prediksi cuaca hingga analisis perilaku konsumen. Buku ini, *Handbook Machine Learning*, hadir sebagai panduan komprehensif yang membahas konsep-konsep dasar hingga implementasi model machine learning yang lebih kompleks. Dimulai dengan pengantar tentang variabel target dan fitur, buku ini memberikan pemahaman yang mendalam mengenai peran data sebagai inti dari pembelajaran mesin. Dengan mengulas metode *supervised* dan *unsupervised learning*, pembaca diajak untuk mengenal berbagai pendekatan dalam membangun model prediksi yang andal.

Diharapkan, buku ini tidak hanya membantu pembaca memahami dasar-dasar *machine learning* tetapi juga memberikan inspirasi untuk menerapkan teknologi ini secara inovatif dalam berbagai konteks.

Pekanbaru, Desember 2024

Penulis

DAFTAR ISI

Pengantar *Machine Learning*

Machine learning adalah cabang dari kecerdasan buatan yang memungkinkan komputer untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit.

Pada *Machine learning* terdapat dua variable yang sering digunakan, yakni variable target dan fitur.

- Variabel target, juga dikenal sebagai label atau variabel dependen, adalah variabel yang ingin diprediksi atau diklasifikasikan dalam model machine learning. Variabel ini merupakan hasil yang diinginkan dari proses prediksi dan menjadi fokus utama dalam pembelajaran terawasi (supervised learning). Variabel target bergantung pada fitur-fitur yang ada untuk menghasilkan nilai atau kategori tertentu.
- Fitur, juga dikenal sebagai variabel independen, prediktor, atau variabel input, adalah variabel-variabel yang digunakan untuk membuat prediksi atau klasifikasi

terhadap variabel target. Fitur ini menyediakan informasi yang relevan yang digunakan oleh model untuk mempelajari pola dan hubungan dalam data. Fitur bisa berupa angka, kategori, teks, atau kombinasi dari berbagai jenis data.

Machine learning dibagi menjadi dua jenis utama yakni *Supervised* dan *Unsupervised Learning*. Dalam pembelajaran *supervised learning,* model dilatih menggunakan data yang telah diberi label. Dalam *supervised learning*, terdapat variabel target (label) yang ingin diprediksi berdasarkan variabel input (fitur). Sementara dalam *Unsupervised Learning*, model mencari pola dalam data tanpa label. Dataset pelatihan, pengujian, dan validasi adalah elemen penting dalam machine learning untuk menguji dan mengevaluasi kinerja model.

Type of ML

	Supervised	Unsupervised
Dataset	Labeled Dataset	Unlabeled Dataset
Task	Predicting or Classifying based on previous labelled data	Find hidden pattern and grouping from unlabelled data
Method	Classification. Regression	Clustering, Dimensional Reduction, Anomaly Detection
Algorithm	Linear Regression, Logistic Regression, KNN, Decision Tree, Random Forest, Boosting	K-Means, DBScan, t-SNE, PCA, Factor Analysis, One-class SVM, Isolation Forest
Use cases	Credit Scoring, Churn Analysis, Propensity Analysis, House Price	Customer segmentation, Market Basket Analysis, Fraud detection

Kasus Unsupervised Learning pada Data Stunting
Pengelompokan (Clustering) Daerah Berdasarkan Tingkat Stunting

Tujuan
Mengelompokkan daerah-daerah di Indonesia berdasarkan karakteristik yang berkaitan dengan stunting untuk mengidentifikasi daerah yang membutuhkan intervensi lebih intensif.

Fitur:
- Persentase anak stunting.
- Usia Anak: Usia anak dalam bulan atau tahun.
- Berat Badan: Berat badan anak dalam kilogram.
- Tinggi Badan: Tinggi badan anak dalam sentimeter.
- Status Gizi Ibu: Status gizi ibu saat hamil (misalnya, cukup gizi, kurang gizi).
- Pendidikan Orang Tua: Tingkat pendidikan ibu dan ayah (misalnya, SD, SMP, SMA, Perguruan Tinggi).
- Pendapatan Keluarga: Pendapatan bulanan keluarga dalam rupiah.
- Akses ke Layanan Kesehatan: Jarak atau waktu tempuh ke fasilitas kesehatan terdekat.
- Kebersihan Air: Kualitas air yang digunakan oleh keluarga (misalnya, bersih, terkontaminasi).

Model machine learning akan menggunakan fitur-fitur untuk mengelompokkan karakteristik yang sesuai menjadi beberapa kluster berdasarkan kesamaan karakteristik.

Kasus Supervised Learning: Prediksi Status Stunting Anak

Variabel Target: Status Stunting Anak
Merupakan variabel target yang ingin diprediksi. Status ini bisa berupa kategori biner (misalnya, "Stunting" dan "Tidak Stunting") atau berupa angka yang menunjukkan tingkat stunting.

Fitur:

- Usia Anak: Usia anak dalam bulan atau tahun.
- Berat Badan: Berat badan anak dalam kilogram.
- Tinggi Badan: Tinggi badan anak dalam sentimeter.
- Status Gizi Ibu: Status gizi ibu saat hamil (misalnya, cukup gizi, kurang gizi).
- Pendidikan Orang Tua: Tingkat pendidikan ibu dan ayah (misalnya, SD, SMP, SMA, Perguruan Tinggi).
- Pendapatan Keluarga: Pendapatan bulanan keluarga dalam rupiah.
- Akses ke Layanan Kesehatan: Jarak atau waktu tempuh ke fasilitas kesehatan terdekat.
- Kebersihan Air: Kualitas air yang digunakan oleh keluarga (misalnya, bersih, terkontaminasi).

Model machine learning akan menggunakan fitur-fitur ini untuk mempelajari pola dan hubungan yang ada dalam data, dan kemudian memprediksi variabel target (status stunting) untuk data baru yang belum pernah dilihat sebelumnya.

Decision Tree

Decision Tree adalah salah satu algoritma machine learning yang digunakan untuk tugas klasifikasi dan regresi. Algoritma ini membagi data ke dalam subset yang lebih kecil berdasarkan fitur tertentu, dan proses ini berlanjut secara rekursif, sehingga membentuk struktur pohon dengan node, cabang, dan daun.

Decision Tree merupakan salah satu metode pembelajaran mesin yang mengimplementasikan struktur hierarkis dalam pengambilan keputusan. Metodologi ini mengadopsi pendekatan rekursif dalam melakukan klasifikasi dan regresi data, di mana setiap node merepresentasikan titik pemisahan berdasarkan fitur-fitur yang telah teridentifikasi. Dalam konteks pembelajaran mesin, *Decision Tree* memiliki kemampuan untuk mengekstraksi pola-pola yang kompleks dari dataset melalui serangkaian pemisahan biner yang sistematis, menghasilkan model prediktif yang dapat diinterpretasikan secara visual.

Implementasi *Decision Tree* melibatkan beberapa algoritma fundamental seperti ID3 (Iterative Dichotomiser 3), CART (Classification and Regression Trees), dan C4.5 yang merupakan evolusi dari algoritma ID3. Setiap algoritma memiliki karakteristik distingtif dalam mengevaluasi nilai informasi pada setiap node dan menentukan kriteria pemisahan optimal. Proses pembentukan pohon keputusan dimulai dari node akar yang merepresentasikan keseluruhan dataset, kemudian secara iteratif melakukan partisi data berdasarkan kriteria information

gain atau gini impurity, hingga mencapai node terminal atau daun yang mengandung prediksi atau klasifikasi final.

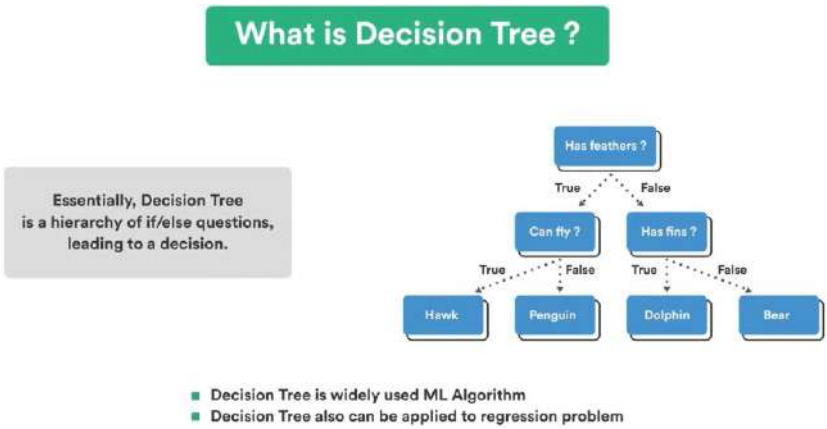
Dalam perspektif metodologis, *Decision Tree* menawarkan keunggulan signifikan dalam hal interpretabilitas model dan kemampuan untuk menangani berbagai tipe data, baik numerik maupun kategorikal. Struktur pohon yang dihasilkan memungkinkan para peneliti dan praktisi untuk memahami logika pengambilan keputusan secara transparan, sekaligus mengidentifikasi fitur-fitur yang memiliki kontribusi substansial dalam proses klasifikasi. Namun, perlu diperhatikan bahwa kompleksitas model yang berlebihan dapat mengakibatkan overfitting, sehingga diperlukan teknik pruning atau pemangkasan untuk mengoptimalkan generalisasi model.

Aplikasi *Decision Tree* telah terimplementasi secara luas dalam berbagai domain keilmuan dan industri, mencakup diagnosis medis, analisis risiko kredit, sistem rekomendasi, dan prediksi fenomena alam. Efektivitas metode ini dalam menghasilkan model prediktif yang akurat dan interpretabel telah mendorong pengembangan varian-varian algoritma yang lebih sophisticated, seperti *Random Forest* dan Gradient Boosted Trees. Melalui integrasi dengan teknik ensemble learning, kapabilitas *Decision Tree* dapat ditingkatkan secara substansial, memberikan solusi yang lebih robust untuk permasalahan klasifikasi dan regresi yang kompleks dalam era analisis data modern.

Istilah yang sering ditemui dalam *Decision Tree* , yakni:

- Node: Titik di mana keputusan dibuat berdasarkan fitur tertentu.
- Cabang: Hasil dari keputusan pada node, yang mengarahkan ke node berikutnya atau ke daun.

- Daun: Titik akhir yang memberikan prediksi atau klasifikasi akhir



Proses Pembentukan *Decision Tree*

- Pemilihan Fitur: Algoritma memilih fitur yang paling informatif untuk membagi data pada setiap node. Kriteria pemilihan biasanya menggunakan ukuran seperti Gini impurity atau entropy.
- Pemisahan Data: Data dibagi berdasarkan nilai fitur yang dipilih.
- Rekursi: Proses ini diulangi secara rekursif untuk setiap subset data yang dihasilkan, sampai salah satu kondisi penghentian tercapai (misalnya, semua data dalam subset termasuk dalam satu kategori atau kedalaman maksimum pohon tercapai).

Langkah-langkah penerapan algoritma *Decision Tree*

1. Impor Pustaka yang Diperlukan
2. Mengumpulkan dan Memuat Data
3. Memisahkan Fitur dan Variabel Target
4. Membagi Data Menjadi Data Latihan dan Data Uji

5. Membangun dan Melatih Model *Decision Tree*
6. Membuat Prediksi
7. Mengevaluasi Model
8. Visualisasi Pohon Keputusan

Studi Kasus: Memprediksi Status Stunting Anak di Indonesia Menggunakan *Decision Tree*

Stunting adalah kondisi gagal tumbuh pada anak balita akibat kekurangan gizi kronis sehingga anak terlalu pendek untuk usianya. Masalah stunting di Indonesia menjadi perhatian serius karena berdampak pada kualitas hidup dan masa depan anak. Oleh karena itu, diperlukan upaya untuk memprediksi dan mengidentifikasi faktor-faktor yang berkontribusi terhadap stunting agar intervensi dapat dilakukan secara tepat.

Tujuan

Membangun model machine learning menggunakan algoritma *Decision Tree* untuk memprediksi status stunting anak berdasarkan berbagai faktor sosial, ekonomi, dan lingkungan. Studi kasus ini bertujuan untuk mengidentifikasi fitur-fitur penting yang berkontribusi terhadap stunting dan membantu dalam pengambilan keputusan kebijakan kesehatan.

Data yang Digunakan

Target: Status Stunting (Target): Severely Stunting (0), Stunting (1), Normal (2)

Dataset yang digunakan mencakup data anak-anak di berbagai provinsi di Indonesia dengan fitur-fitur berikut:

- Provinsi
- Tipe Wilayah (perkotaan/pedesaan)
- Kepemilikan Jaminan Kesehatan
- Pengetahuan Ayah tentang Stunting
- Pengetahuan Ibu tentang Stunting
- Jumlah Anggota Rumah Tangga (ART)
- Jenis Sarana Air Minum Utama
- Pengolahan Air Minum Sebelum Dikonsumsi
- Jenis Sarana Air Selain Minum Utama
- Kepemilikan Fasilitas Tempat Buang Air Besar
- Ketersediaan Tempat Cuci Tangan
- Jenis Bahan Bangunan Utama Atap Rumah
- Jenis Bahan Bangunan Utama Langit-langit Rumah
- Jenis Bahan Bangunan Utama Dinding
- Status Penguasaan Bangunan Tempat Tinggal
- Mendapatkan Bantuan Program Keluarga Harapan (PKH)
- Mendapatkan Bantuan Bantuan Pangan Non Tunai (BPNT)
- Mendapatkan Bantuan Bantuan Langsung Tunai (BLT)
- Mendapatkan Bantuan Program Subsidi Listrik

Pengolahan data menggunakan googlecolab
https://colab.research.google.com/ dengan syntax:

```
# Import pustaka yang diperlukan
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn import tree

# Mengimpor data dari file CSV
data = pd.read_csv('data_stunting_provinsi.csv')

# Menampilkan beberapa baris pertama data
print(data.head())

# Mendefinisikan fitur (X) dan variabel target (y)
X = data[['Provinsi', 'Tipe Wilayah', 'Kepemilikan Jaminan
Kesehatan', 'Pengetahuan Ayah tentang Stunting',
'Pengetahuan Ibu tentang Stunting', 'Jumlah ART', 'Jenis Sarana
Air Minum Utama', 'Pengolahan Air Minum Sebelum
Dikonsumsi', 'Jenis Sarana Air Selain Minum Utama',
'Kepemilikan Fasilitas Tempat Buang Air Besar', 'Ketersediaan
Tempat Cuci Tangan', 'Jenis Bahan Bangunan Utama Atap
Rumah', 'Jenis Bahan Bangunan Utama Langit-langit Rumah',
'Jenis Bahan Bangunan Utama Dinding', 'Status Penguasaan
Bangunan Tempat Tinggal', 'Mendapatkan Bantuan PKH',
'Mendapatkan Bantuan BPNT', 'Mendapatkan Bantuan BLT',
'Mendapatkan Bantuan Program Subsidi Listrik']]
y = data['Status Stunting']
```

```
# Mengubah variabel kategori menjadi numerik
X = pd.get_dummies(X)

# Membagi data menjadi data latihan dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Membuat model Decision Tree
clf = DecisionTreeClassifier(criterion='gini', max_depth=5)

# Melatih model
clf.fit(X_train, y_train)

# Membuat prediksi pada data uji
y_pred = clf.predict(X_test)

# Menghitung akurasi
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Akurasi: {accuracy:.2f}')

# Menampilkan confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Menampilkan classification report
class_report = metrics.classification_report(y_test, y_pred,
target_names=['Severely Stunting', 'Stunting', 'Normal'])
print('Classification Report:')
print(class_report)

# Visualisasi pohon keputusan
```

```
plt.figure(figsize=(20,10))
tree.plot_tree(clf, feature_names=X.columns,
class_names=['Severely Stunting', 'Stunting', 'Normal'],
filled=True)
plt.show()
```

Salah satu cara yang dapat digunakan untuk meningkatkaan akurasi dari pemodelan, dapat digunakan GridSearchCV. Grid Search adalah teknik pencarian hyperparameter yang digunakan untuk menemukan kombinasi parameter terbaik untuk model machine learning. Teknik ini mencoba setiap kombinasi parameter yang ditentukan dalam sebuah grid (kotak) untuk mengidentifikasi set parameter yang memberikan kinerja terbaik berdasarkan metrik evaluasi tertentu, seperti akurasi.

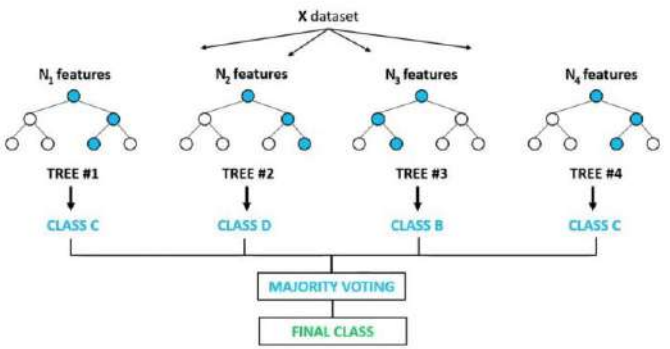
Cara kerja Grid Search yakni sebagai berikut:

- Definisikan Model: Tentukan model machine learning yang akan digunakan.
- Tentukan Parameter Grid: Buat daftar parameter yang ingin dicoba beserta rentang nilai yang memungkinkan.
- Pelatihan dan Evaluasi Model: Grid Search akan melatih model dengan setiap kombinasi parameter dalam grid dan mengevaluasi kinerjanya menggunakan teknik cross-validation.
- Pilih Kombinasi Terbaik: Kombinasi parameter yang menghasilkan kinerja terbaik (misalnya, akurasi tertinggi) dipilih sebagai hyperparameter optimal untuk model.

```
# Menentukan parameter grid untuk dicari
param_grid = {
   'criterion': ['gini', 'entropy'],
   'max_depth': [3, 5, 7, 10],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4]}

# Membuat model Decision Tree
clf = DecisionTreeClassifier()

# Melakukan pencarian grid
grid_search = GridSearchCV(estimator=clf,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Menampilkan parameter terbaik
print("Best parameters found: ", grid_search.best_params_)

# Menggunakan model terbaik untuk prediksi
best_clf = grid_search.best_estimator_
y_pred = best_clf.predict(X_test)
```

Random Forest

Random Forest adalah algoritma ensemble learning yang menggabungkan beberapa pohon keputusan (*Decision Tree*) untuk meningkatkan kinerja model dan mengurangi risiko overfitting. *Random Forest* merepresentasikan suatu pendekatan metodologis dalam domain pembelajaran mesin yang mengintegrasikan multiplisitas pohon keputusan untuk menghasilkan prediksi dengan tingkat akurasi superior. Algoritma ini mengimplementasikan paradigma ensemble learning, di mana agregasi prediksi dari sejumlah pohon keputusan independen diutilisasi untuk menghasilkan output final, baik dalam konteks klasifikasi maupun regresi. Karakteristik fundamentalnya terletak pada kapabilitas dalam melakukan random sampling dengan pengembalian (bootstrap sampling) dari dataset original, yang secara signifikan meningkatkan robustness dan generalizability model.



Dalam konstruksi modelnya, *Random Forest* mengaplikasikan metrik evaluasi berupa Gini Index, yang diformulasikan sebagai:

$$\text{Gini Index} = 1 - \sum_{i=1}^{c} (p_i)^2$$

dimana pi merepresentasikan probabilitas relatif dari kelas yang diobservasi dalam dataset. Metrik ini berfungsi sebagai parameter kuantitatif dalam determinasi optimal splitting nodes pada setiap pohon keputusan. Signifikansi Gini Index terletak pada kemampuannya untuk mengukur tingkat impurity atau heterogenitas dalam suatu node, yang berimplikasi langsung terhadap efektivitas segregasi data pada setiap split.

Sebagai alternatif dari Gini Index, *Random Forest* juga mengimplementasikan konsep Entropy dalam proses pembentukan pohon keputusan. Entropy, yang diekspresikan melalui formula:

$$\text{Entropy} = \sum_{i=1}^{c} - p_i \times \log_2 (p_i)$$

yang menyediakan pengukuran terhadap degree of randomness atau ketidakpastian dalam distribusi kelas pada suatu node. Meskipun memerlukan kompleksitas komputasional yang lebih tinggi karena penggunaan fungsi logaritmik, Entropy memberikan perspektif complementary dalam evaluasi kualitas split pada setiap node.

Optimalisasi performa *Random Forest* melibatkan kalibrasi beberapa parameter krusial, termasuk kuantitas pohon dalam ensemble dan kriteria terminasi untuk setiap pohon individual. Peningkatan jumlah pohon dalam forest secara general berkorelasi positif dengan akurasi model, namun dengan trade-off berupa kompleksitas komputasional yang lebih tinggi.

Proses sampling hingga mencapai node terminal untuk kalkulasi probabilitas, yang dinormalisasi terhadap total sampel, merupakan aspek integral dalam konstruksi model yang robust dan reliable untuk aplikasi klasifikasi maupun regresi.

Langkah-langkah Pembentukan *Random Forest*

1. Pengambilan Sampel Acak (Bootstrap Sampling)

Random Forest menggunakan teknik bootstrap sampling untuk membuat beberapa subset data dari dataset asli. Setiap subset data (disebut bootstrap sample) diambil dengan penggantian dari dataset asli, sehingga satu data point bisa muncul lebih dari sekali dalam satu bootstrap sample.

2. Pembangunan Pohon Keputusan (*Decision Tree*)

Untuk setiap bootstrap sample, sebuah pohon keputusan dibangun. Proses pembangunan pohon keputusan ini sama dengan pohon keputusan biasa, namun dengan dua perbedaan utama:

- Setiap pohon keputusan hanya menggunakan subset data yang berbeda (bootstrap sample).
- Setiap pohon keputusan menggunakan subset fitur yang berbeda untuk menentukan split pada setiap node.

3. Pembagian Fitur Acak pada Setiap Node

Pada setiap node dalam pohon keputusan, hanya sejumlah kecil fitur yang dipilih secara acak yang dipertimbangkan untuk pembagian. Ini membantu dalam mengurangi korelasi antar pohon dan meningkatkan keragaman model. Dengan memilih subset fitur acak pada setiap node, *Random Forest* mengurangi risiko overfitting.

4. Prediksi Ensemble

Setelah semua pohon keputusan dibangun, *Random Forest* menggunakan prediksi ensemble untuk membuat prediksi akhir. Untuk tugas klasifikasi, prediksi akhir diambil berdasarkan mayoritas suara dari semua pohon keputusan. Untuk tugas regresi, prediksi akhir adalah rata-rata dari prediksi semua pohon keputusan.

Studi Kasus: Memprediksi Status Stunting Anak di Indonesia Menggunakan *Random Forest*

Dengan menggunakan studi kasus sebelumnya, buatlah klasifikasi stunting menggunakan algoritma *Random Forest*.

Pengolahan data menggunakan googlecolab https://colab.research.google.com/ dengan syntax

```
# Import pustaka yang diperlukan
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn import tree

# Mengimpor data dari file CSV
data = pd.read_csv('data_stunting_provinsi.csv')
```

```
# Menampilkan beberapa baris pertama data
print(data.head())

# Mendefinisikan fitur (X) dan variabel target (y)
X = data[['Provinsi', 'Tipe Wilayah', 'Kepemilikan Jaminan
Kesehatan', 'Pengetahuan Ayah tentang Stunting',
'Pengetahuan Ibu tentang Stunting', 'Jumlah ART', 'Jenis Sarana
Air Minum Utama', 'Pengolahan Air Minum Sebelum
Dikonsumsi', 'Jenis Sarana Air Selain Minum Utama',
'Kepemilikan Fasilitas Tempat Buang Air Besar', 'Ketersediaan
Tempat Cuci Tangan', 'Jenis Bahan Bangunan Utama Atap
Rumah', 'Jenis Bahan Bangunan Utama Langit-langit Rumah',
'Jenis Bahan Bangunan Utama Dinding', 'Status Penguasaan
Bangunan Tempat Tinggal', 'Mendapatkan Bantuan PKH',
'Mendapatkan Bantuan BPNT', 'Mendapatkan Bantuan BLT',
'Mendapatkan Bantuan Program Subsidi Listrik']]
y = data['Status Stunting']

# Mengubah variabel kategori menjadi numerik
X = pd.get_dummies(X)

# Membagi data menjadi data latihan dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Membuat model Random Forest dengan parameter default
rf = RandomForestClassifier(n_estimators=100,
max_depth=None, random_state=42)

# Melatih model
rf.fit(X_train, y_train)
```

```
# Membuat prediksi pada data uji
y_pred = rf.predict(X_test)

# Menghitung akurasi
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Akurasi: {accuracy:.2f}')

# Menampilkan confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Menampilkan classification report
class_report = metrics.classification_report(y_test, y_pred,
target_names=['Severely Stunting', 'Stunting', 'Normal'])
print('Classification Report:')
print(class_report)

# Visualisasi salah satu pohon keputusan dalam Random Forest
plt.figure(figsize=(20,10))
tree.plot_tree(rf.estimators_[0], feature_names=X.columns,
class_names=['Severely Stunting', 'Stunting', 'Normal'],
filled=True)
plt.show()
```

Optimasi dengan GridsearchCV

```
# Menentukan parameter grid untuk dicari
param_grid = {
   'n_estimators': [100, 200, 300],
   'max_depth': [None, 10, 20, 30],
   'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4]
}

# Membuat model Random Forest
rf = RandomForestClassifier()

# Melakukan pencarian grid
grid_search = GridSearchCV(estimator=rf,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Menampilkan parameter terbaik
print("Best parameters found: ", grid_search.best_params_)

# Menggunakan model terbaik untuk prediksi
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
```

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) adalah algoritma non-parametrik yang digunakan untuk klasifikasi dan regresi. Algoritma ini bekerja dengan mencari sejumlah K tetangga terdekat (nearest neighbors) dari data yang ingin diprediksi dan mengklasifikasikannya berdasarkan mayoritas kelas dari tetangga-tetangga tersebut. Metodologi ini berbasis pada asumsi fundamental bahwa observasi yang memiliki karakteristik similar cenderung terklasifikasi dalam kategori yang sama. Dalam konteks matematis, K-NN mengoperasikan kalkulasi jarak antara data uji dengan setiap instance dalam dataset training, dimana parameter k merepresentasikan jumlah tetangga terdekat yang dipertimbangkan dalam proses klasifikasi atau regresi.

Formula yang diimplementasikan dalam K-NN melibatkan berbagai metrik jarak, dengan Euclidean distance sebagai metrik yang sering digunakan, dengan formula:

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

dimana d merepresentasikan jarak, xi merupakan data latih, dan yi adalah data uji. Alternatif metrik distance yang dapat diaplikasikan mencakup Manhattan distance, Minkowski distance, Chebyshev distance, dan correlation-based distance,

masing-masing dengan karakteristik dan use case spesifik dalam konteks analisis data multidimensional.

Determinasi nilai k merupakan aspek crucial dalam optimalisasi performa K-NN, dimana k dapat direpresentasikan sebagai konstanta atau variabel yang ditentukan melalui proses cross-validation. Seleksi nilai k yang terlalu kecil dapat menghasilkan model yang sensitive terhadap noise, sementara nilai k yang terlalu besar dapat mengaburkan boundary antar kelas. Dalam implementasinya, weighted voting dapat diaplikasikan untuk memberikan pengaruh yang lebih besar pada tetangga yang lebih dekat, meningkatkan akurasi klasifikasi terutama dalam kasus dengan distribusi kelas yang tidak seimbang.

Langkah kerja KNN:

1. Menentukan Nilai K:

   Nilai K adalah jumlah tetangga terdekat yang akan dipertimbangkan dalam proses klasifikasi atau regresi. Nilai K biasanya dipilih secara empiris berdasarkan performa model pada data validasi atau melalui teknik seperti cross-validation.
2. Menghitung Jarak:

   Untuk setiap titik data yang ingin diklasifikasikan, KNN menghitung jarak antara titik data tersebut dengan semua titik data dalam dataset pelatihan. Jarak yang umum digunakan adalah jarak Euclidean, tetapi bisa juga menggunakan jarak Manhattan atau jarak Minkowski.

3. Menentukan K Tetangga Terdekat:

   Setelah menghitung jarak antara titik data baru dengan semua titik data dalam dataset pelatihan, KNN memilih K titik data dengan jarak terdekat.

4. Melakukan Prediksi**:**

   Klasifikasi:

   - Untuk tugas klasifikasi, KNN melakukan prediksi berdasarkan mayoritas kelas dari K tetangga terdekat. Titik data baru akan diklasifikasikan ke dalam kelas yang paling banyak muncul di antara K tetangga terdekat.
   - Misalnya, jika K=3 dan kelas dari tetangga terdekat adalah [1, 0, 1], maka titik data baru akan diklasifikasikan sebagai kelas 1 karena kelas 1 muncul dua kali.

   Regresi:

   - Untuk tugas regresi, KNN melakukan prediksi dengan mengambil rata-rata nilai dari K tetangga terdekat.
   - Misalnya, jika K=3 dan nilai dari tetangga terdekat adalah [4.5, 3.7, 5.1], maka prediksi untuk titik data baru adalah rata-rata dari nilai-nilai tersebut: (4.5+3.7+5.1)/3=4.43(4.5 + 3.7 + 5.1) / 3 = 4.43(4.5+3.7+5.1)/3=4.43.

   Kelebihan KNN

   - Mudah Dipahami: KNN adalah algoritma yang sederhana dan mudah dipahami.

- Tidak Membutuhkan Pelatihan: Tidak ada proses pelatihan eksplisit, sehingga model dapat dengan cepat diimplementasikan.
- Fleksibel: KNN dapat digunakan untuk klasifikasi dan regresi.

Kekurangan:

- Komputasi Berat: Menghitung jarak antara titik data baru dengan semua titik data dalam dataset pelatihan bisa menjadi sangat lambat jika dataset sangat besar.
- Sensitif terhadap Skala Data: KNN memerlukan data yang distandarisasi atau dinormalisasi agar metrik jarak dapat berfungsi dengan baik.
- Overfitting: Jika nilai K terlalu kecil, KNN rentan terhadap overfitting. Sebaliknya, jika nilai K terlalu besar, KNN mungkin kehilangan detail penting dalam data (underfitting).

Studi Kasus: Memprediksi Status Stunting Anak di Indonesia Menggunakan KNN

Dengan menggunakan studi kasus sebelumnya, buatlah klasifikasi stunting menggunakan algoritma KNN

Pengolahan data menggunakan googlecolab https://colab.research.google.com/ dengan syntax

```
# Import pustaka yang diperlukan
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

# Mengimpor data dari file CSV
data = pd.read_csv('data_stunting_provinsi.csv')

# Menampilkan beberapa baris pertama data
print(data.head())

# Mendefinisikan fitur (X) dan variabel target (y)
X = data[['Provinsi', 'Tipe Wilayah', 'Kepemilikan Jaminan
Kesehatan', 'Pengetahuan Ayah tentang Stunting',
'Pengetahuan Ibu tentang Stunting', 'Jumlah ART', 'Jenis Sarana
Air Minum Utama', 'Pengolahan Air Minum Sebelum
Dikonsumsi', 'Jenis Sarana Air Selain Minum Utama',
'Kepemilikan Fasilitas Tempat Buang Air Besar', 'Ketersediaan
Tempat Cuci Tangan', 'Jenis Bahan Bangunan Utama Atap
Rumah', 'Jenis Bahan Bangunan Utama Langit-langit Rumah',
'Jenis Bahan Bangunan Utama Dinding', 'Status Penguasaan
Bangunan Tempat Tinggal', 'Mendapatkan Bantuan PKH',
'Mendapatkan Bantuan BPNT', 'Mendapatkan Bantuan BLT',
'Mendapatkan Bantuan Program Subsidi Listrik']]
y = data['Status Stunting']

# Mengubah variabel kategori menjadi numerik
```

```
X = pd.get_dummies(X)

# Membagi data menjadi data latihan dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standarisasi fitur (fitur KNN bekerja lebih baik dengan data
yang distandarisasi)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Membuat model KNN dengan K=5
knn = KNeighborsClassifier(n_neighbors=5)

# Melatih model
knn.fit(X_train, y_train)

# Membuat prediksi pada data uji
y_pred = knn.predict(X_test)

# Menghitung akurasi
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Akurasi: {accuracy:.2f}')

# Menampilkan confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Menampilkan classification report
class_report = metrics.classification_report(y_test, y_pred,
```

```
target_names=['Severely Stunting', 'Stunting', 'Normal'])
print('Classification Report:')
print(class_report)
```

Optimasi dengan GridsearchCV

```
# Menentukan parameter grid untuk dicari
param_grid = {
   'n_neighbors': [3, 5, 7, 9, 11],
   'weights': ['uniform', 'distance'],
   'metric': ['euclidean', 'manhattan', 'minkowski']
}

# Membuat model KNN
knn = KNeighborsClassifier()

# Melakukan pencarian grid
grid_search = GridSearchCV(estimator=knn,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Menampilkan parameter terbaik
print("Best parameters found: ", grid_search.best_params_)

# Menggunakan model terbaik untuk prediksi
best_knn = grid_search.best_estimator_
y_pred = best_knn.predict(X_test)
```

Gradientboosting

Gradientboosting adalah teknik *machine learning* yang digunakan untuk tugas regresi dan klasifikasi. Teknik ini bekerja dengan membangun model secara bertahap dan menggabungkan mereka untuk menghasilkan model prediktif yang kuat. *Gradientboosting* menggunakan pendekatan boosting, di mana model yang lebih sederhana (*weak learners*) digabungkan untuk membentuk model yang lebih kompleks (*strong learner*).

Gradientboosting merepresentasikan suatu paradigma ensemble learning dalam domain pembelajaran mesin yang mengimplementasikan strategi iteratif untuk meminimalisasi error prediksi secara gradual. Metodologi ini mengadopsi pendekatan sequential dalam konstruksi model, dimana setiap model subsequen difokuskan pada optimasi residual error yang dihasilkan oleh ensemble model sebelumnya. Fundamental teoretis dari *Gradientboosting* terletak pada aplikasi gradient descent dalam ruang fungsi, yang memungkinkan optimasi sistematis terhadap arbitrary differentiable loss function.

Dalam konteks matematis, *Gradientboosting* mengoperasikan dua framework distingtif untuk task klasifikasi dan regresi, masing-masing dengan fungsi objektif yang terspesialisasi. Untuk kasus regresi, Mean Square Error (MSE) diutilisasi sebagai cost function yang mengukur deviasi antara prediksi dan nilai aktual, Sementara itu, dalam konteks klasifikasi, logarithmic loss function diimplementasikan untuk mengoptimasi probabilitas prediksi kelas, memberikan penalti

yang lebih besar untuk misklasifikasi dengan tingkat keyakinan yang tinggi.

Implementasi *Gradientboosting* melibatkan kalibrasi beberapa parameter krusial yang mempengaruhi performa model secara signifikan. Learning rate (shrinkage) mengontrol kontribusi setiap weak learner terhadap ensemble final, sementara number of boosting stages menentukan kompleksitas model. Karakteristik distingtif *Gradientboosting* terletak pada kemampuannya untuk mengakomodasi berbagai base learners, meskipun *Decision Tree* s dengan kedalaman terbatas (shallow trees) merupakan pilihan yang paling commonly utilized karena balance optimal antara bias dan variance.

Dalam perspektif aplikatif, *Gradientboosting* telah mendemonstrasikan efektivitas yang remarkable dalam berbagai domain pembelajaran mesin, mulai dari prediksi finansial hingga natural language processing. Metodologi ini mengekshibisi resiliensi terhadap outliers dan missing values, serta kapabilitas dalam menangani interaksi non-linear antar fitur. Melalui implementasi teknik regularisasi dan early stopping, *Gradientboosting* dapat menghasilkan model yang robust dan generalisable, menjadikannya salah satu pilihan predominan dalam komunitas machine learning untuk task prediktif yang kompleks.

Gradientboosting membangun model secara berurutan, di mana setiap model baru mencoba untuk mengoreksi kesalahan yang dibuat oleh model sebelumnya. Langkah-langkah utama dalam proses *Gradientboosting*, yakni:

1. Inisialisasi Model:
   Proses dimulai dengan model sederhana, biasanya sebuah pohon keputusan yang sangat kecil (stump).
2. Menghitung Residual (Kesalahan):
   Setelah model pertama dilatih, prediksi dibuat dan residual (kesalahan) dihitung sebagai perbedaan antara nilai aktual dan nilai prediksi.
3. Membangun Model Baru untuk Residual:
   Model baru dilatih untuk memprediksi residual dari model sebelumnya. Model ini bertujuan untuk mengurangi kesalahan yang ada.
4. Menggabungkan Model:
   Model yang baru dilatih ditambahkan ke model sebelumnya dengan cara menggabungkan prediksi mereka. Proses ini berlanjut hingga jumlah model yang ditentukan tercapai atau kesalahan tidak lagi berkurang secara signifikan.
5. Mengoptimalkan Fungsi Kehilangan:
   Gradientboosting mengoptimalkan fungsi kehilangan (loss function) dengan menggunakan metode gradien penurunan (gradient descent). Pada setiap iterasi, model baru dibangun untuk mengarahkan penurunan gradien dari fungsi kehilangan.

Kelebihan *Gradientboosting*

- Akurasi Tinggi: *Gradientboosting* dapat menghasilkan model dengan akurasi yang sangat tinggi karena menggabungkan kekuatan dari banyak weak learners.
- Fleksibel: Dapat digunakan untuk berbagai jenis masalah, baik regresi maupun klasifikasi.

- Mampu Menangani Outlier: *Gradientboosting* dapat menangani outlier dengan baik karena model difokuskan pada kesalahan yang besar.

Kekurangan *Gradientboosting*

- Kompleksitas Komputasi: Proses pelatihan *Gradientboosting* bisa sangat memakan waktu dan sumber daya karena melibatkan banyak iterasi.
- Rentan terhadap Overfitting: Jika tidak diatur dengan baik, model *Gradientboosting* bisa mengalami overfitting, terutama jika jumlah model yang dibangun terlalu banyak.
- Sensitif terhadap Hiperparameter: Kinerja *Gradientboosting* sangat bergantung pada pemilihan hiperparameter yang tepat.

Studi Kasus: Memprediksi Status Stunting Anak di Indonesia Menggunakan *Gradientboosting*

Dengan menggunakan studi kasus sebelumnya, buatlah klasifikasi stunting menggunakan algoritma *Gradientboosting*

Pengolahan data menggunakan googlecolab
https://colab.research.google.com/ dengan syntax

```
# Import pustaka yang diperlukan
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientboostingClassifier
from sklearn import metrics

# Mengimpor data dari file CSV
data = pd.read_csv('data_stunting_provinsi.csv')

# Menampilkan beberapa baris pertama data
print(data.head())

# Mendefinisikan fitur (X) dan variabel target (y)
X = data[['Provinsi', 'Tipe Wilayah', 'Kepemilikan Jaminan
Kesehatan', 'Pengetahuan Ayah tentang Stunting',
'Pengetahuan Ibu tentang Stunting', 'Jumlah ART', 'Jenis Sarana
Air Minum Utama', 'Pengolahan Air Minum Sebelum
Dikonsumsi', 'Jenis Sarana Air Selain Minum Utama',
'Kepemilikan Fasilitas Tempat Buang Air Besar', 'Ketersediaan
Tempat Cuci Tangan', 'Jenis Bahan Bangunan Utama Atap
Rumah', 'Jenis Bahan Bangunan Utama Langit-langit Rumah',
'Jenis Bahan Bangunan Utama Dinding', 'Status Penguasaan
Bangunan Tempat Tinggal', 'Mendapatkan Bantuan PKH',
'Mendapatkan Bantuan BPNT', 'Mendapatkan Bantuan BLT',
'Mendapatkan Bantuan Program Subsidi Listrik']]
y = data['Status Stunting']

# Mengubah variabel kategori menjadi numerik
```

```
X = pd.get_dummies(X)

# Membagi data menjadi data latihan dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Membuat model Gradientboosting dengan parameter default
gb = GradientboostingClassifier()

# Melatih model
gb.fit(X_train, y_train)

# Membuat prediksi pada data uji
y_pred = gb.predict(X_test)

# Menghitung akurasi
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Akurasi: {accuracy:.2f}')

# Menampilkan confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Menampilkan classification report
class_report = metrics.classification_report(y_test, y_pred,
target_names=['Severely Stunting', 'Stunting', 'Normal'])
print('Classification Report:')
print(class_report)
```

Pengolahan data menggunakan googlecolab
https://colab.research.google.com/ dengan syntax

```
# Membagi data menjadi data latihan dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Menentukan parameter grid untuk dicari
param_grid = {
   'learning_rate': [0.01, 0.1, 0.05],
   'n_estimators': [100, 200, 300],
   'max_depth': [3, 4, 5]
}

# Membuat model Gradientboosting
gb = GradientboostingClassifier()

# Melakukan pencarian grid
grid_search = GridSearchCV(estimator=gb,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Menampilkan parameter terbaik
print("Best parameters found: ", grid_search.best_params_)

# Menggunakan model terbaik untuk prediksi
best_gb = grid_search.best_estimator_
y_pred = best_gb.predict(X_test)
```

XGBoost

Extreme Gradientboosting (*XGBoost*) merepresentasikan suatu algoritma pembelajaran mesin yang mengimplementasikan optimasi ekstensif terhadap *Gradientboosting Decision Tree*s. Metodologi ini mengintegrasikan konsep ensemble learning dengan pendekatan boosting yang terstruktur, dimana multiple weak learners ditransformasikan secara sistematis menjadi strong learners melalui proses iteratif. Karakteristik distingtif *XGBoost* terletak pada kapabilitasnya dalam mengoptimalkan komputasi paralel dan utilisasi memori yang efisien, memungkinkan pemrosesan dataset dengan dimensionalitas dan kompleksitas tinggi.

Dalam perspektif matematis, *XGBoost* mengimplementasikan regularisasi yang mengkombinasikan L1 (Lasso) dan L2 (Ridge) untuk mengontrol kompleksitas model dan mencegah overfitting. Algoritma ini mengaplikasikan second-order gradient statistics dalam proses optimasi objective function, memberikan konvergensi yang lebih cepat dibandingkan dengan traditional *Gradientboosting*. Framework *XGBoost* juga mengintegrasikan mekanisme penanganan missing values dan split finding yang sophisticated, meningkatkan robustness model dalam menangani data real-world yang imperfect.

XGBoost menyediakan interface yang kompatibel dengan scikit-learn melalui wrapper class seperti XGBClassifier dan XGBRegressor, memfasilitasi integrasi seamless dengan ekosistem pembelajaran mesin Python. Implementasinya melibatkan kalibrasi berbagai hyperparameter crucial, termasuk

learning rate, maximum depth of trees, number of estimators, dan parameter regularisasi, yang dapat dioptimasi melalui teknik cross-validation atau bayesian optimization. Karakteristik ini memungkinkan fine-tuning model untuk mencapai performa optimal dalam task klasifikasi maupun regresi.

Signifikansi *XGBoost* dalam domain pembelajaran mesin dimanifestasikan melalui adopsinya yang ekstensif dalam kompetisi data science dan aplikasi industrial. Superioritas algoritmik dalam aspek computational efficiency dan predictive accuracy telah menjadikannya pilihan predominan untuk pemodelan prediktif dengan structured data. Melalui implementasi teknik-teknik seperti feature importance scoring dan early stopping, *XGBoost* tidak hanya memberikan performa prediktif yang excellent tetapi juga insight valuable mengenai kontribusi relatif setiap fitur terhadap output model.

Langkah-langkah utama dalam cara kerja *XGBoost*:

1. Inisialisasi Model:
   Model pertama dibuat, biasanya sebuah pohon keputusan yang sederhana (stump).
2. Menghitung Residual (Kesalahan):
   Setelah model pertama dilatih, prediksi dibuat dan residual (kesalahan) dihitung sebagai perbedaan antara nilai aktual dan nilai prediksi.
3. Membangun Model Baru untuk Residual:
   Model baru dilatih untuk memprediksi residual dari model sebelumnya. Model ini bertujuan untuk mengurangi kesalahan yang ada.
4. Menggabungkan Model:
   Model yang baru dilatih ditambahkan ke model sebelumnya dengan cara menggabungkan prediksi mereka.

Proses ini berlanjut hingga jumlah model yang ditentukan tercapai atau kesalahan tidak lagi berkurang secara signifikan.

5. Optimasi Fungsi Kehilangan:
   XGBoost mengoptimalkan fungsi kehilangan (loss function) dengan menggunakan metode gradien penurunan (gradient descent). Pada setiap iterasi, model baru dibangun untuk mengarahkan penurunan gradien dari fungsi kehilangan.

Keunggulan *XGBoost*

- Efisiensi Kecepatan: *XGBoost* dioptimalkan untuk kecepatan dan efisiensi, baik dalam hal pelatihan maupun prediksi.
- Kinerja Tinggi: *XGBoost* memberikan akurasi yang sangat tinggi karena memanfaatkan algoritma boosting yang kuat.
- Kemampuan Penanganan Missing Values: *XGBoost* dapat secara otomatis menangani nilai yang hilang dalam dataset.
- Regularisasi: *XGBoost* memiliki fitur regularisasi yang membantu mengurangi overfitting.
- Paralelisme: *XGBoost* mendukung pelatihan model secara paralel, mempercepat proses pelatihan.
- Skalabilitas: *XGBoost* dapat digunakan pada dataset besar dan dapat diskalakan dengan baik.

Studi Kasus: Memprediksi Status Stunting Anak di Indonesia Menggunakan *XGBoost*

Dengan menggunakan studi kasus sebelumnya, buatlah klasifikasi stunting menggunakan algoritma *XGBoost*

Pengolahan data menggunakan googlecolab
https://colab.research.google.com/ dengan syntax

```
# Import pustaka yang diperlukan
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from XGBoost import XGBClassifier
from sklearn import metrics

# Mengimpor data dari file CSV
data = pd.read_csv('data_stunting_provinsi.csv')

# Menampilkan beberapa baris pertama data
print(data.head())

# Mendefinisikan fitur (X) dan variabel target (y)
X = data[['Provinsi', 'Tipe Wilayah', 'Kepemilikan Jaminan
Kesehatan', 'Pengetahuan Ayah tentang Stunting',
'Pengetahuan Ibu tentang Stunting', 'Jumlah ART', 'Jenis Sarana
Air Minum Utama', 'Pengolahan Air Minum Sebelum
Dikonsumsi', 'Jenis Sarana Air Selain Minum Utama',
'Kepemilikan Fasilitas Tempat Buang Air Besar', 'Ketersediaan
Tempat Cuci Tangan', 'Jenis Bahan Bangunan Utama Atap
Rumah', 'Jenis Bahan Bangunan Utama Langit-langit Rumah',
'Jenis Bahan Bangunan Utama Dinding', 'Status Penguasaan
Bangunan Tempat Tinggal', 'Mendapatkan Bantuan PKH',
'Mendapatkan Bantuan BPNT', 'Mendapatkan Bantuan BLT',
'Mendapatkan Bantuan Program Subsidi Listrik']]
y = data['Status Stunting']

# Mengubah variabel kategori menjadi numerik
```

```
X = pd.get_dummies(X)

# Membagi data menjadi data latihan dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Membuat model XGBoost dengan parameter default
xgb = XGBClassifier()

# Melatih model
xgb.fit(X_train, y_train)

# Membuat prediksi pada data uji
y_pred = xgb.predict(X_test)

# Menghitung akurasi
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Akurasi: {accuracy:.2f}')

# Menampilkan confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Menampilkan classification report
class_report = metrics.classification_report(y_test, y_pred,
target_names=['Severely Stunting', 'Stunting', 'Normal'])
print('Classification Report:')
print(class_report)
```

Pengolahan data menggunakan googlecolab
https://colab.research.google.com/ dengan syntax

```
# Menentukan parameter grid untuk dicari
param_grid = {
   'learning_rate': [0.01, 0.1, 0.2],
   'max_depth': [3, 4, 5],
   'n_estimators': [100, 200, 300],
   'subsample': [0.7, 0.8, 0.9],
   'colsample_bytree': [0.7, 0.8, 0.9]
}
# Membuat model XGBoost
xgb = XGBClassifier()

# Melakukan pencarian grid
grid_search = GridSearchCV(estimator=xgb,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
# Menampilkan parameter terbaik
print("Best parameters found: ", grid_search.best_params_)

# Menggunakan model terbaik untuk prediksi
best_xgb = grid_search.best_estimator_
y_pred = best_xgb.predict(X_test)
```

Stacking Ensemble

Stacking Ensemble adalah teknik ensemble yang menggabungkan beberapa model dasar (*base models)* dan menggunakan model meta (*meta-learner*) untuk membuat prediksi akhir. Teknik ini bertujuan untuk meningkatkan kinerja model dengan menggabungkan kekuatan berbagai algoritma.

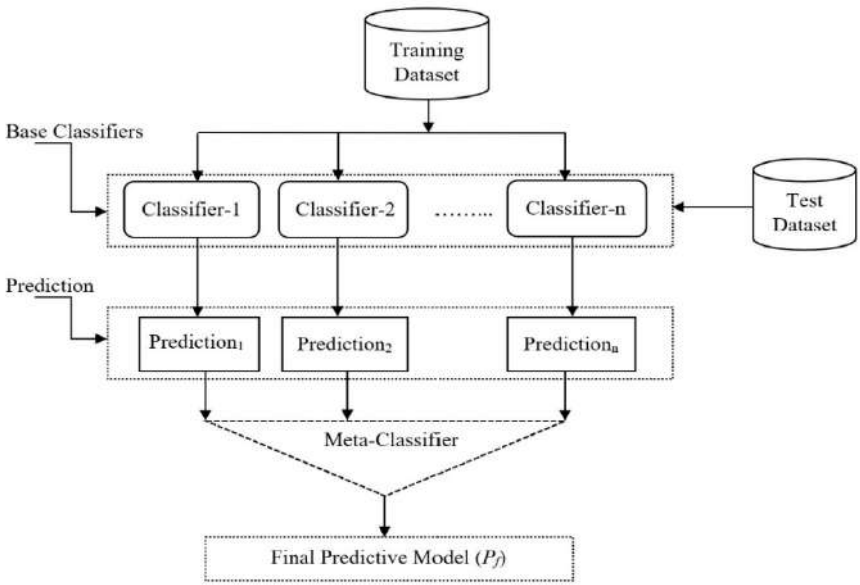


Figure 1 The General Architecture Of Stacking Ensemble Learning.

Stacking Ensemble Learning merepresentasikan suatu metodologi pembelajaran mesin yang mengintegrasikan multiple base classifiers dengan meta-classifier dalam struktur hierarkis untuk optimalisasi performa prediktif. Paradigma ini

mengimplementasikan konsep heterogeneous ensemble, dimana diversifikasi algoritma pembelajaran pada level base classifiers dikombinasikan dengan meta-learning pada level superior.

Fundamental teoretis dari stacking terletak pada kapabilitasnya dalam mengeksploitasi karakteristik komplementer dari berbagai algoritma pembelajaran, memungkinkan capturing patterns yang lebih komprehensif dibandingkan dengan single model. Dalam perspektif arsitektural, *Stacking Ensemble* mengorganisasi proses pembelajaran dalam dua level distinktif: base-level yang terdiri dari multiple base classifiers, dan meta-level yang mengimplementasikan meta-classifier. Base classifiers, yang dapat mencakup berbagai algoritma seperti *Decision Tree*s, support vector machines, dan neural networks, dilatih menggunakan dataset original untuk menghasilkan predictions yang akan diutilisasi sebagai input features untuk meta-classifier.

Metodologi training dalam *Stacking Ensemble* melibatkan prosedur cross-validation yang sophisticated untuk generating meta-features. Implementasi k-fold cross-validation pada level base classifiers memungkinkan konstruksi dataset meta-level yang reliable dan unbiased, mencegah overfitting yang potensial terjadi dalam proses stacking. Signifikansi prosedur ini terletak pada kapabilitasnya dalam memastikan meta-classifier menerima predictions yang representative dari true generalization performance base classifiers. Meta-classifier, yang juga dikenal sebagai level-1 learner, kemudian mengoptimalkan kombinasi dari predictions tersebut untuk menghasilkan final prediction yang lebih akurat dan robust.

Dalam konteks aplikatif, *Stacking Ensemble* learning telah mendemonstrasikan superioritas performa dalam berbagai domain, terutama dalam kasus dimana single model atau homogeneous ensemble methods menunjukkan limitasi. Analogous dengan konsep boosting dalam hal sequential improvement, stacking memberikan framework yang lebih fleksibel untuk mengkombinasikan diverse learning algorithms. Melalui implementasi teknik regularisasi dan model selection yang appropriate pada kedua level, *Stacking Ensemble* dapat menghasilkan model yang robust dan generalisable, menjadikannya pilihan viable untuk task klasifikasi yang kompleks dalam real-world applications. Keunggulan stacking terletak pada kemampuannya untuk mengadaptasi dan mengoptimalkan kontribusi setiap base classifier berdasarkan karakteristik spesifik dari domain problem yang dihadapi.

Gradientboosting membangun model secara berurutan, di mana setiap model baru mencoba untuk mengoreksi kesalahan yang dibuat oleh model sebelumnya. Langkah-langkah utama dalam proses Stacking ensembla, yakni:

1. Pemilihan dan Pelatihan Model Dasar (Base Models)

   Pemilihan Base Models: Pilih beberapa algoritma machine learning yang akan digunakan sebagai base models. Contoh base models yang umum digunakan meliputi: *Decision Tree* , *Random Forest*, Nearest Neighbors (KNN), *Gradientboosting*, *XGBoost*. Pelatihan Base Models: Latih setiap base model menggunakan data latihan. Setiap base model mempelajari pola dalam data dan menghasilkan prediksi.

2. Membangun dan Melatih Model Meta (Meta Learner)

Mengumpulkan Prediksi Base Models: Gunakan base models yang telah dilatih untuk membuat prediksi pada data latihan. Prediksi ini akan digunakan sebagai fitur input untuk meta-learner. Selain itu juga dilakukan pelatihan meta learner: Latih meta-learner menggunakan prediksi dari base models sebagai fitur dan target asli sebagai label. Meta-learner belajar menggabungkan prediksi dari base models untuk menghasilkan prediksi akhir yang lebih akurat.

3. Membangun Model *Stacking Ensemble*

Menggabungkan Base Models dan Meta Learner: Buat model stacking yang menggabungkan base models dan meta-learner. Pada tahap ini, prediksi dari base models digunakan sebagai input untuk meta-learner. Kemudian dilakukan Validasi Silang menggunakan teknik validasi silang (cross-validation) untuk menguji kinerja model stacking dan memastikan bahwa model tidak overfitting.

4. Optimasi Hyperparameter (Opsional)

Menentukan Parameter Grid: Tentukan parameter grid untuk setiap base model dan meta-learner yang ingin dioptimalkan. Kemudian dilakukan Pencarian Grid menggunakan GridSearchCV untuk mencari kombinasi parameter terbaik yang meningkatkan kinerja model. Proses ini melibatkan pelatihan ulang model dengan berbagai kombinasi parameter dan memilih yang terbaik berdasarkan metrik evaluasi.

Kelebihan *Gradientboosting*

1. Peningkatan Akurasi

   Menggabungkan Kekuatan Model Berbeda: *Stacking Ensemble* menggabungkan berbagai model dasar yang memiliki kekuatan dan kelemahan yang berbeda, sehingga dapat menangkap berbagai pola dalam data yang mungkin tidak dapat ditangkap oleh satu model saja. Pengurangan Bias dan Varians: Dengan menggabungkan beberapa model, *Stacking Ensemble* dapat mengurangi bias dan varians, yang menghasilkan model yang lebih akurat dan robust.

2. Fleksibilitas

   Berbagai Jenis Algoritma: *Stacking Ensemble* dapat menggabungkan model dari berbagai jenis algoritma, termasuk regresi, pohon keputusan, SVM, dan lain-lain. Model meta dapat berupa model yang berbeda dari base models, seperti menggunakan regresi logistik sebagai meta-learner untuk menggabungkan prediksi dari pohon keputusan dan model berbasis ensemble lainnya.

3. Kemampuan Generalisasi yang Lebih Baik.

   Stacking Ensemble cenderung lebih tahan terhadap overfitting dibandingkan model individual karena menggabungkan prediksi dari beberapa model yang berbeda. Dengan menggabungkan beberapa model, *Stacking Ensemble* dapat meningkatkan kemampuan generalisasi pada data baru yang tidak terlihat selama pelatihan.

4. Robustness

   Mengurangi Kelemahan Model Individual: Model individual mungkin memiliki kelemahan dalam menangani jenis data tertentu, tetapi dengan menggabungkan beberapa model, kelemahan ini dapat diimbangi oleh kekuatan model lain.

5. Kemampuan untuk Menggunakan Model yang Telah Dilatih

   Base models yang telah dilatih dapat digunakan kembali sebagai bagian dari ensemble, yang menghemat waktu dan sumber daya komputasi. Penyempurnaan Model yang Ada: Model meta dapat digunakan untuk menyempurnakan prediksi dari model dasar yang ada, meningkatkan kinerja keseluruhan.

6. Kompleksitas yang Dapat Diatur

   Kontrol atas Kompleksitas: Anda dapat mengatur kompleksitas model ensemble dengan memilih jenis dan jumlah base models serta model meta, yang memungkinkan penyesuaian sesuai kebutuhan dan sumber daya yang tersedia.

Kekurangan *Gradientboosting*

1. Kompleksitas Komputasi:

   Pelatihan model stacking pada dataset besar dapat memakan waktu berjam-jam atau bahkan berhari-hari, memerlukan penggunaan GPU atau cluster komputasi untuk mempercepat proses.

2. Kesulitan dalam Pemilihan Model:

   Memilih base models yang optimal memerlukan eksperimen dengan berbagai kombinasi model dan parameter, yang bisa menjadi proses yang membosankan dan memakan waktu.

3. Potensi Overfitting:

   Jika tidak dilakukan dengan validasi silang yang tepat, model meta bisa belajar untuk menyesuaikan prediksi base models secara berlebihan, menghasilkan performa yang buruk pada data uji.

4. Kesulitan dalam Interpretasi

   Dalam aplikasi medis atau keuangan, di mana keputusan model harus dapat dijelaskan, interpretasi hasil dari model stacking bisa menjadi tantangan besar.

Studi Kasus: Memprediksi Status Stunting Anak di Indonesia Menggunakan *Stacking Ensemble*

Dengan menggunakan studi kasus sebelumnya, buatlah klasifikasi stunting menggunakan algoritma *Stacking Ensemble* dengan menggunakan base algoritma yang telah dipelajari sebelumnya

Pengolahan data menggunakan googlecolab
https://colab.research.google.com/ dengan syntax

```
# Import pustaka yang diperlukan
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientboostingClassifier, StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from XGBoost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

# Mengimpor data dari file CSV
data = pd.read_csv('data_stunting_provinsi.csv')

# Menampilkan beberapa baris pertama data
print(data.head())

# Mendefinisikan fitur (X) dan variabel target (y)
X = data[['Provinsi', 'Tipe Wilayah', 'Kepemilikan Jaminan
Kesehatan', 'Pengetahuan Ayah tentang Stunting',
'Pengetahuan Ibu tentang Stunting', 'Jumlah ART', 'Jenis Sarana
Air Minum Utama', 'Pengolahan Air Minum Sebelum
Dikonsumsi', 'Jenis Sarana Air Selain Minum Utama',
'Kepemilikan Fasilitas Tempat Buang Air Besar', 'Ketersediaan
Tempat Cuci Tangan', 'Jenis Bahan Bangunan Utama Atap
Rumah', 'Jenis Bahan Bangunan Utama Langit-langit Rumah',
'Jenis Bahan Bangunan Utama Dinding', 'Status Penguasaan
Bangunan Tempat Tinggal', 'Mendapatkan Bantuan PKH',
```

```
'Mendapatkan Bantuan BPNT', 'Mendapatkan Bantuan BLT',
'Mendapatkan Bantuan Program Subsidi Listrik']]
y = data['Status Stunting']

# Mengubah variabel kategori menjadi numerik
X = pd.get_dummies(X)

# Membagi data menjadi data latihan dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standarisasi fitur (opsional, tapi disarankan untuk beberapa
model)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Membuat model dasar (base models)
base_models = [
   ('dt', DecisionTreeClassifier(random_state=42)),
   ('rf', RandomForestClassifier(n_estimators=100,
random_state=42)),
   ('knn', KNeighborsClassifier(n_neighbors=5)),
   ('gb', Gradientboosting Classifier(n_estimators=100,
random_state=42)),
   ('xgb', XGBClassifier(n_estimators=100, random_state=42))
]

# Membuat model meta (meta-learner)
meta_model = LogisticRegression()

# Membuat model stacking
```

```
stacking_model = StackingClassifier(estimators=base_models,
final_estimator=meta_model, cv=5)

# Melatih model stacking
stacking_model.fit(X_train, y_train)

# Membuat prediksi pada data uji
y_pred = stacking_model.predict(X_test)

# Menghitung akurasi
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Akurasi: {accuracy:.2f}')

# Menampilkan confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Menampilkan classification report
class_report = metrics.classification_report(y_test, y_pred,
target_names=['Severely Stunting', 'Stunting', 'Normal'])
print('Classification Report:')
print(class_report)
```

Optimalisasi Parameter

```
# Import pustaka yang diperlukan
# Menentukan parameter grid untuk model dasar dan model
meta
param_grid = {
   'dt__max_depth': [3, 5, 7],
```

```
    'rf__n_estimators': [50, 100, 200],
    'rf__max_depth': [3, 5, 7],
    'knn__n_neighbors': [3, 5, 7],
    'gb__n_estimators': [50, 100, 200],
    'gb__learning_rate': [0.01, 0.1, 0.2],
    'xgb__n_estimators': [50, 100, 200],
    'xgb__learning_rate': [0.01, 0.1, 0.2],
    'final_estimator__C': [0.1, 1, 10]
}

# Melakukan pencarian grid
grid_search = GridSearchCV(estimator=stacking_model,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Menampilkan parameter terbaik
print("Best parameters found: ", grid_search.best_params_)

# Menggunakan model terbaik untuk prediksi
best_stacking_model = grid_search.best_estimator_
y_pred = best_stacking_model.predict(X_test)
```

DAFTAR PUSTAKA

M. Kumar, S. Singhal, S. Shekhar, B. Sharma, and G. Srivastava, "Optimized *Stacking Ensemble* learning model for breast cancer detection and classification using machine learning," Sustainability, vol. 14, no. 21, p. 13998, 2022.

N. Hidayati and A. Hermawan, "K-Nearest Neighbor (K-NN) algorithm with Euclidean and Manhattan in classification of student graduation," Journal of Engineering and Applied Technology, vol. 2, no. 2, pp. 86–91, 2021.

S. Malik, R. Harode, and A. S. Kunwar, "*XGBoost*: A deep dive into boosting (introduction documentation)," Simon Fraser University: Burnaby, BC, Canada, 2020.

W. Dong, Y. Huang, B. Lehane, and G. Ma, "*XGBoost* algorithm-based prediction of concrete electrical resistivity for structural health monitoring," Autom Constr, vol. 114, p. 103155, Jun. 2020, doi: 10.1016/j.autcon.2020.103155.


CPU