Algoritma Pohon Keputusan Termodifikasi sebagai Alat Imputasi Multivariat Data Nonlinear

Muhammad Fathur Rizky - 13523105[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]rizkyfathur326@gmail.com, 13523105@std.stei.itb.ac.id

***Abstrak*—Imputasi data adalah langkah penting dalam pengolahan data untuk mengatasi masalah data yang hilang atau tidak lengkap. Kehilangan data dapat memengaruhi akurasi analisis dan model prediksi sehingga teknik imputasi yang efektif sangat diperlukan untuk memastikan validitas hasil analisis. Pohon keputusan yang sering digunakan dalam pemodelan prediktif dapat menangani hubungan nonlinear. Pada makalah ini, dijelaskan bahwa algoritma ini cukup andal dalam menghadapi kasus nilai hilang dengan rerata galat RMSE 4.95 dan efisien yang memiliki kompleksitas waktu $O(d^2n^2log_2n)$ dibandingkan teknik-teknik imputasi lainnya.**

***Kata kunci*—Imputasi Data, Pohon Keputusan, Nilai Hilang, Pemodelan Prediktif**

I. PENDAHULUAN

Imputasi data adalah salah satu langkah penting dalam pengolahan data yang digunakan untuk mengatasi masalah data yang hilang atau tidak lengkap. Kehilangan data dapat mempengaruhi akurasi analisis dan model prediksi sehingga imputasi yang efektif diperlukan untuk memastikan validitas hasil analisis. Salah satu metode yang sering digunakan dalam analisis prediktif adalah pohon keputusan yang dapat menangani data dengan hubungan nonlinear dan kompleks. Meskipun demikian, pada implementasi standar pohon keputusan, nilai *null* atau data yang hilang tidak diimplementasikan sehingga diperlukan teknik imputasi di luar proses pemodelan.

Pada makalah ini, dilakukan modifikasi pada algoritma pohon keputusan untuk memperhitungkan nilai *null* dalam proses pemisahan data. Modifikasi ini bertujuan untuk mengatasi kelemahan dalam pemisahan yang tidak memperhitungkan data hilang sehingga proses imputasi dapat dilakukan dengan lebih efektif. Dalam modifikasi ini, nilai *null* diperlakukan secara eksplisit dalam pemisahan data. Nilai hilang dipertimbangkan bersama dengan data yang memenuhi kriteria pemisahan tertentu. Dengan pendekatan ini, pohon keputusan dapat menghasilkan pemisahan yang lebih baik, termasuk untuk data yang memiliki nilai hilang, dan memungkinkan imputasi yang lebih akurat.

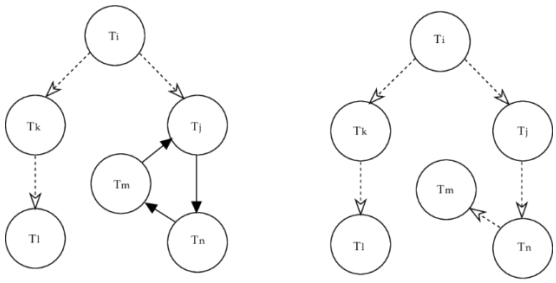
Pendekatan ini diharapkan dapat meningkatkan kinerja pohon keputusan dalam imputasi data sehingga model dapat mengisi nilai yang hilang dengan lebih tepat. Dengan demikian, modifikasi ini tidak hanya meningkatkan akurasi imputasi, tetapi juga memperluas kemampuan pohon keputusan dalam menangani dataset yang memiliki sejumlah nilai hilang dan meningkatkan kualitas data yang digunakan dalam analisis lebih lanjut.

II. DASAR TEORI

A. Graf

Graf adalah suatu struktur data diskret yang terdiri atas simpul-simpul dan sisi-sisi. Graf dinotasikan sebagai $G = (V, E)$ dengan V merupakan simpul (*vertex*) dan E merupakan tepi (*edge*) [1]. Graf digunakan untuk menggambarkan hubungan antara objek-objek yang memiliki karakteristik sebagai berikut.

- Simpul: Simpul adalah elemen dasar dalam graf yang merepresentasikan titik atau objek dalam suatu sistem. Setiap simpul biasanya diberi label atau nomor untuk membedakan satu simpul dengan yang lainnya.
- Tepi: Tepi adalah hubungan atau penghubung antara dua simpul. Setiap sisi menghubungkan sepasang simpul dan dapat berupa sisi tak terarah (*undirected*) atau sisi berarah (*directed*) tergantung pada jenis graf.
- Siklus: Siklus adalah jalur yang dimulai dan berakhir pada simpul yang sama, tanpa melewati sisi atau simpul lebih dari sekali. Graf yang mengandung siklus disebut graf berbentuk siklus, sedangkan graf yang tidak mengandung siklus disebut graf asiklik.



Gambar 1. Graf Siklis dan Asiklis

- Arah: Pada graf berarah (*digraph*), sisi memiliki arah, yang berarti hubungan antar simpul bersifat satu arah. Sedangkan pada graf tak terarah, sisi tidak memiliki arah, yang berarti hubungan antar simpul bersifat dua arah.

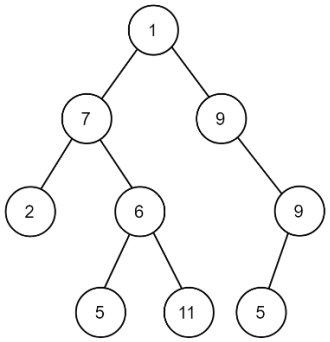
B. Pohon

Pohon adalah salah satu struktur data yang berbasis graf dengan karakteristik khusus berupa hubungan hierarkis antara simpul-simpulnya. Secara formal, pohon dapat didefinisikan sebagai graf berarah tanpa siklus (*Directed Acyclic*

Graph/DAG) yang memiliki satu simpul utama sebagai akar (root) dan cabang-cabang yang menghubungkan simpul lainnya. Simpul-simpul dalam pohon dibedakan menjadi beberapa jenis, yaitu akar (simpul paling atas yang menjadi titik awal struktur pohon), simpul daun (*leaf*) yang tidak memiliki anak, dan simpul internal yang memiliki anak namun bukan akar [2].

Komponen utama dari pohon meliputi simpul (*node*), tepi (*edge*), tingkat (level), kedalaman (*depth*), tinggi (*height*), dan subpohon (*subtree*). Tepi adalah hubungan antara dua simpul yang menggambarkan keterkaitan hierarkis antara orang tua (*parent*) dan anak (*child*). Tingkat (*level*) dihitung berdasarkan jarak suatu simpul dari akar, sedangkan kedalaman (*depth*) menunjukkan jarak dari akar ke simpul tertentu.

Salah satu varian dari pohon yang banyak digunakan adalah pohon biner. Pohon biner adalah jenis pohon dengan setiap simpul hanya dapat memiliki paling banyak dua anak, yang disebut sebagai anak kiri (*left child*) dan anak kanan (*right child*).



Gambar 2. Pohon Biner

C. Pohon Keputusan (Decision Tree)

Pohon keputusan adalah salah satu algoritma yang digunakan dalam pembelajaran mesin (*machine learning*) untuk tugas klasifikasi dan regresi. Algoritma ini membangun model berbentuk pohon yang digunakan untuk membuat keputusan berdasarkan fitur-fitur input. Setiap simpul dalam pohon mewakili keputusan atau pertanyaan mengenai atribut atau fitur, dan cabang-cabangnya mewakili hasil dari keputusan tersebut. Pada akhirnya, setiap daun pohon mewakili kelas atau nilai yang diprediksi [3].

Proses pembangkitan pohon keputusan dimulai dengan pemilihan atribut yang paling baik untuk membagi data pada setiap simpul. Pembagian ini dilakukan secara rekursif hingga seluruh data terklasifikasi atau mencapai kriteria berhenti tertentu. Pohon keputusan memiliki komponen utama sebagai berikut.

- Variansi: Variansi digunakan untuk mengukur variabilitas dalam data. Semakin tinggi variansi, semakin luas persebaran data tersebut. Variansi dapat dirumuskan sebagai berikut.

$$\mathrm{Var}(D) = \frac{1}{|D|}\sum_{i=1}^{|D|}(y_i - \bar{y})^2$$

  dengan y_i adalah nilai target pada data ke-i, $|D|$ adalah jumlah sampel dalam dataset D, dan $\bar{y}$ adalah nilai rata-rata dalam dataset.
- Information gain: Information gain adalah ukuran yang digunakan untuk memilih atribut yang paling baik dalam membagi dataset pada simpul tertentu. Information Gain dihitung berdasarkan pengurangan *parent variance* dengan *child variance*. Atribut dengan Information Gain tertinggi akan dipilih sebagai pembagi pada simpul tersebut. Secara matematis, information gain dirumuskan sebagai berikut.

$$\mathrm{IG}_{\mathrm{Var}} = \mathrm{Var}(D) - \left(\frac{|D_L|}{|D|}\mathrm{Var}(D_L) + \frac{|D_R|}{|D|}\mathrm{Var}(D_R)\right)$$

  Dengan $\mathrm{Var}(D)$ adalah varians sebelum pembagian, $\frac{|D_L|}{|D|}\mathrm{Var}(D_L)$ dan $\frac{|D_R|}{|D|}\mathrm{Var}(D_R)$ adalah varians rata-rata tertimbang dari subset data setelah pemisahan, dan IG_{Var} adalah selisih antara varians sebelum pemisahan dan setelah pemisahan. Semakin besar nilai IG, semakin baik pemisahan tersebut dalam mengurangi ketidakpastian dalam dataset.

Kompleksitas algoritma decision tree dapat dianalisis berdasarkan proses pembentukannya. Dalam mencari pemisah terbaik pada setiap simpul, algoritma melakukan iterasi sebanyak d kali untuk setiap fitur. Pada setiap iterasi tersebut, algoritma akan mengiterasi nilai unik pada sampel sebanyak n kali, dimana untuk setiap nilai unik dibutuhkan perhitungan entropi dengan kompleksitas $O(n)$. Dengan demikian, total kompleksitas untuk mencari pemisah terbaik pada satu simpul adalah $O(dn^2)$. Proses ini kemudian dilakukan secara rekursif dengan membagi sampel menjadi dua bagian pada setiap level pohon. Dalam kasus tree yang seimbang, jumlah level maksimum yang terbentuk adalah $\log_2 n$. Oleh karena itu, total kompleksitas waktu untuk pembangkitan pohon keputusan adalah $O(dn^2 \log_2 n)$ dengan d merepresentasikan jumlah fitur dan n merepresentasikan jumlah sampel dalam dataset.

D. Root Mean Squared Error

Root Mean Squared Error (RMSE) adalah salah satu metrik evaluasi yang sering digunakan untuk mengukur kinerja model regresi. RMSE menghitung rata-rata dari kuadrat selisih antara nilai yang sebenarnya (y_i) dan nilai yang diprediksi oleh model ($\hat{y}_i$). Rumus RMSE secara matematis dengan n adalah jumlah sampel dinyatakan sebagai berikut.

$$\mathrm{RMSE} = \sqrt{\sum_{i=1}^{n}\frac{(y_i - \bar{y})^2}{n}}$$

III. Implementasi Alat Imputasi Data Nonlinear

A. Penjelasan dan Penyiapan Dataset

Digunakan sebuah dataset performa murid dalam menghadapi ujian. Dataset ini merupakan dataset generatif dan ditujukan untuk kepentingan edukasi. Dataset ini terdiri atas 6607 baris dan 7 kolom tanpa nilai hilang (*null*). Berikut merupakan penjelasan mengenai tiap-tiap kolom pada dataset.

Tabel 1. Penjelasan Dataset

Nama kolom	Penjelasan
Hours_Studied	Jumlah jam yang dihabiskan untuk belajar per minggu.
Attendance	Persentase jumlah kehadiran

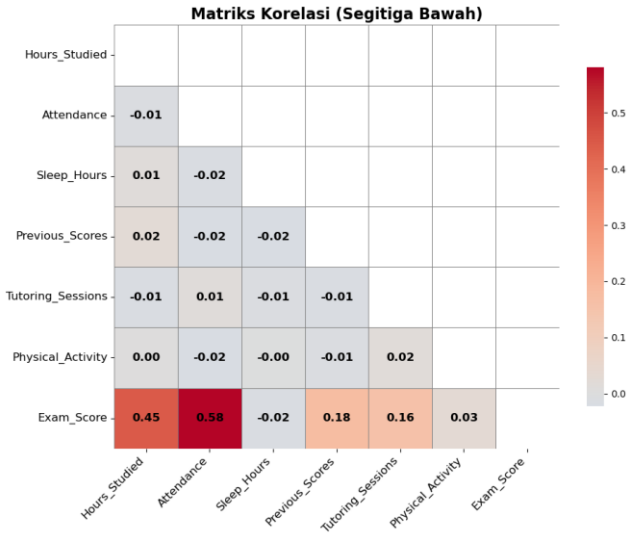
Sleep_Hours	Lama murid tidur (jam)
Previous_Scores	Skor ujian sebelumnya
Tutoring Sessions	Jumlah sesi bimbingan belajar yang dihadiri per bulan.
Physical_Activity	Rata-rata jumlah aktivitas fisik per minggu
Exam_Score	Nilai ujian akhir

Selain penjelasan, diberikan juga statistik dasar dari setiap kolom yang meliputi nilai minimum, mean, maksimum, dan standar deviasi. Dari tabel di bawah, terlihat bahwa kolom Previous_Scores mempunyai standar deviasi paling tinggi sehingga ini akan menjadi tantangan yang besar bagi model dalam melakukan imputasi.

Tabel 2. Statistik Dasar Dataset

Nama kolom	Statistik			
	Min	Mean	Max	Std
Hours_Studied	1	19.97	44	5.99
Attendance	60	79.98	100	11.55
Sleep_Hours	4	7.03	10	1.47
Previous_Scores	50	75.07	100	14.4
Tutoring Sessions	0	1.49	8	1.23
Physical_Activity	0	2.97	6	1.03
Exam_Score	55	67.24	101	3.89

Untuk membuktikan bahwa data yang digunakan adalah data dengan hubungan nonlinear, maka ditampilkan gambar matriks korelasi menggunakan koefisien pearson sebagai penghitung korelasi. Terlihat bahwa terdapat hubungan cukup linear antara Exam_Score dengan Hours_Studied dan Attendance, tetapi tidak linear dengan yang lainnya. Oleh karena itu, data yang diujikan sudah cukup representatif dengan topik yang ditulis.



Gambar 3. Matriks Korelasi (Segitiga Bawah)

Karena topik makalah ini bertujuan menangani kasus data hilang, maka akan diimplementasikan fungsi untuk merandomisasi indeks (i, j) agar dipilih sebagai nilai hilang. Penulis mengubah nilai menjadi *null* pada dua fitur untuk setiap barisnya. Berikut kode dalam bahasa Python.

```
def random_nan(_df: pd.DataFrame) ->
pd.DataFrame:
 df = _df.copy()
 for i in range(df.shape[0]):
  j = np.random.choice(df.shape[1], 2,
replace=False)
  df.iloc[i, j] = np.nan
 return df
```

Dari proses tersebut, dihasilkan data terbaru dengan 2/7 merupakan nilai hilang.

B. Implementasi Pohon Keputusan untuk Regresi

Setelah melakukan penyiapan dataset, selanjutnya adalah membuat algoritma pohon keputusan. Dalam membuat pohon keputusan, penulis menggunakan *library* tambahan, yakni pandas, numpy, dan matplotlib untuk visualisasi. Pertama-tama, dibuat kelas Node terlebih dahulu.

```
class Node:
 def __init__(self, f=None, t=None, l=None,
r=None, v=None):
  self.f = f  # fitur
  self.t = t  # batas pemisah
  self.l = l  # anak kiri
  self.r = r  # anak kanan
  self.v = v  # nilai pada daun
```

Setelah membuat kelas Node, selanjutnya adalah membuat kelas DecisionTreeRegressor. Kelas ini memiliki metode inisialisasi (`__init__`), `fit` untuk melakukan pembangkitan pohon, dan predict yang digunakan dalam melakukan prediksi/inferensi.

```
class DecisionTreeRegressor:
 def __init__(self,min_split=2,min_leaf=1,
max_depth=None):
  self.min_split = min_split
  self.min_leaf = min_leaf
  self.max_depth = max_depth
  self.root = None

 def fit(self, X, y):
  self.root = self._grow(X, y, d=0)
  return self

 def predict(self, X):
  return np.array([self._traverse(row,
self.root) for _, row in X.iterrows()])
```

Fungsi berikutnya adalah `_grow` yang bertanggung jawab untuk membangun pohon keputusan secara rekursif. Proses dimulai dengan memeriksa apakah jumlah data pada simpul saat ini kurang dari batas minimum (`min_split`), yang jika benar, fungsi akan berhenti dan mengembalikan simpul daun dengan nilai yang dihitung menggunakan `_leaf_val(y)`. Selanjutnya, fungsi mencari pemisahan terbaik pada dataset dengan memanggil `_best_split(X, y)` yang akan mengembalikan fitur (f) dan nilai pemisahan (t). Jika tidak ada pemisahan yang dapat dilakukan, fungsi akan mengembalikan simpul daun. Data kemudian dibagi menjadi dua subset: satu untuk nilai fitur yang lebih kecil dari t atau nilai fitur hilang dan satu lagi untuk nilai yang lebih besar atau sama dengan t. Setelah pemisahan, fungsi memeriksa apakah jumlah data pada salah satu sisi (kiri atau kanan) kurang dari batas minimal (`min_leaf`). Jika salah satu subset memiliki data yang terlalu sedikit, maka simpul tersebut menjadi daun. Jika kedua subset memiliki cukup data, fungsi `_grow` dipanggil kembali secara rekursif untuk membangun pohon keputusan pada kedua sisi. Pada akhirnya, fungsi ini mengembalikan simpul yang menyimpan informasi tentang fitur pemisah, nilai pemisah, serta anak-anak simpul kiri dan kanan yang telah dibangun melalui rekursi.

```
def _grow(self, X: pd.DataFrame, y:pd.Series,
d: int):
 n = len(y)

 if n < self.min_split or (self.max_depth is
not None and d >= self.max_depth):
  return Node(v=self._leaf_val(y))

 f, t = self._best_split(X, y)

 if f is None:
  return Node(v=self._leaf_val(y))

 l_idx = X[f].isna() | (X[f] < t)
 r_idx = ~l_idx

 if l_idx.sum() < self.min_leaf or
r_idx.sum() < self.min_leaf:
  return Node(v=self._leaf_val(y))

 l = self._grow(X[l_idx], y[l_idx], d + 1)
 r = self._grow(X[r_idx], y[r_idx], d + 1)
 return Node(f, t, l, r)
```

Fungsi selanjutnya adalah `_traverse` yang digunakan untuk memprediksi nilai X berdasarkan pohon keputusan yang sudah dibangkitkan. Proses ini dilakukan dengan membandingkan apakah

```
def _traverse(self, row: pd.Series, n: Node):
 if n.v is not None: return n.v
 if pd.isna(row[n.f]): # jika nilai hilang
  return self._traverse(row, n.l)
 if row[n.f] < n.t:
  return self._traverse(row, n.l)
 return self._traverse(row, n.r)
```

Fungsi selanjutnya adalah `_best_split` yang digunakan untuk mencari pemisah mana yang terbaik. Fungsi ini mempertimbangkan fungsi `_information_gain` sebagai parameter bagus atau tidaknya. Semakin besar nilai information gain untuk fitur yang dipilih, maka fitur tersebut memiliki bobot terbesar dalam pemisahan.

```
def _gain(self, y: pd.Series, X_col:
pd.Series, t):
    p_var = self._variance(y)

    l_idx = X_col.isna() | (X_col < t)
    r_idx = ~l_idx

    if l_idx.sum() == 0 or r_idx.sum() == 0:
        return 0

    n = len(y)
    n_l, n_r = l_idx.sum(), r_idx.sum()
    var_l, var_r = self._variance(y[l_idx]),
self._variance(y[r_idx])
    c_var = (n_l/n)*var_l+(n_r/n)*var_r

    return p_var - c_var
```

Fungsi berikutnya adalah `_leaf_val`. Fungsi ini mengeluarkan rata-rata nilai target di daun pada saat itu.

```
def _leaf_val(self, y: pd.Series):
 return y.mean()
```

Fungsi yang diberikan digunakan untuk menghitung information gain atau peningkatan informasi dalam pembentukan pohon keputusan. Fungsi ini terdiri dari dua bagian utama, yakni `_gain` dan `_variance`.

Fungsi _variance menghitung variansi dari data target, yang menggambarkan sejauh mana nilai-nilai target tersebar. Semakin besar variansi, semakin besar ketidakpastian dalam data tersebut.

Fungsi `_gain` menghitung peningkatan informasi yang diperoleh dengan membagi data berdasarkan suatu threshold (`t`) pada salah satu fitur (`X_col`). Langkah pertama adalah menghitung variansi seluruh data target (`p_var`). Data kemudian dibagi menjadi dua subset, berdasarkan apakah nilai fitur lebih kecil atau lebih besar dari threshold. Setelah itu, dihitung variansi masing-masing subset. Jika salah satu subset kosong, fungsi mengembalikan nilai gain 0 karena pembagian tersebut tidak memberikan informasi yang berguna. Jika kedua subset berisi data, dihitung rata-rata berimbang dari variansi kedua subset. Information gain dihitung dengan mengurangi variansi gabungan subset dari variansi awal. Nilai gain menunjukkan seberapa baik pembagian data tersebut dapat mengurangi ketidakpastian dalam target sehingga membantu dalam memilih pembagian terbaik dalam pohon keputusan.

```
def _gain(self, y: pd.Series, X_col:
pd.Series, t):
 p_var = self._variance(y)

 l_idx = X_col.isna() | (X_col < t)
 r_idx = ~l_idx

 if l_idx.sum() == 0 or r_idx.sum() == 0:
  return 0

 n = len(y)
 n_l, n_r = l_idx.sum(), r_idx.sum()
 v_l, v_r = self._variance(y[l_idx]),
self._variance(y[r_idx])
 c_var = (n_l / n) * v_l + (n_r / n) * v_r

 return p_var - c_var

def _variance(self, y: pd.Series):
 return y.var()
```

C. Pipelining dan Pengujian

Pada gambar di bawah ini, terlihat proses pembagian dataset menjadi dua subset: data pelatihan dan data pengujian. Proses dimulai dengan pengacakan data secara acak untuk memastikan bahwa pembagian data tidak mengandung bias. Selanjutnya, 80% dari total dataset dialokasikan untuk data pelatihan, yang digunakan untuk melatih model, sementara 20% sisanya digunakan sebagai data pengujian untuk mengevaluasi kinerja model yang telah dilatih. Pembagian ini penting untuk memastikan bahwa model dapat dievaluasi menggunakan data yang tidak digunakan dalam proses pelatihan, sehingga dapat memberikan gambaran yang lebih akurat mengenai kemampuan model dalam melakukan generalisasi pada data baru.

```
def _train_test_split(self, train:
pd.DataFrame, test_size=0.2) -> tuple:
 n = len(train)
 idx = np.arange(n)
 np.random.shuffle(idx)
 train_idx = idx[int(n * test_size):]
 test_idx = idx[:int(n * test_size)]
 return train.iloc[train_idx],
train.iloc[test_idx]
```

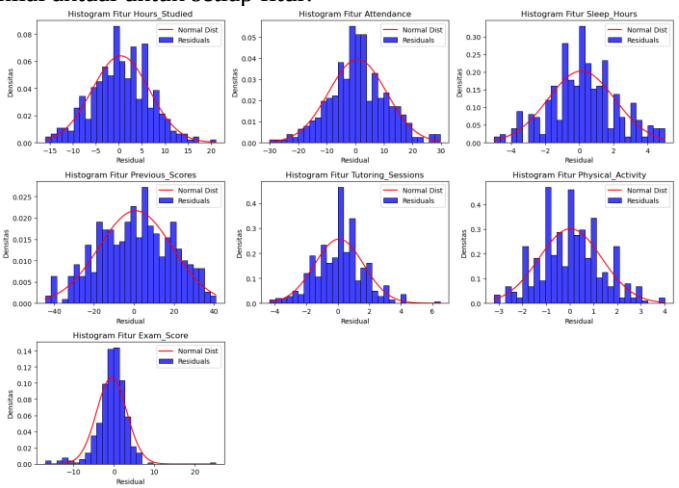
Pada bagian pengujian, penulis menggunakan metrik *root mean squared error* (RMSE) untuk setiap fitur sebagai indikator utama dalam evaluasi model. Metrik ini dipilih karena memiliki

kecocokan yang tinggi dan merupakan salah satu metode yang paling banyak digunakan dalam analisis galat dan mengingat kemampuannya untuk memberikan penalti yang lebih besar pada galat yang lebih besar, sehingga memberikan gambaran yang jelas tentang seberapa jauh prediksi menyimpang dari nilai sebenarnya.

```
error_list = [[] for _ in range(7)]
for col_idx, col in enumerate(test.columns):
    X_train, y_train =
train.drop(columns=col), train[col]
    y_train = y_train.dropna()
    X_train = X_train.loc[y_train.index]

    X_test, y_test = test.drop(columns=col),
test[col]
    nan_indices = y_test.index[y_test.isna()]
    X_test = X_test.loc[nan_indices]

    model = DecisionTreeRegressor()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_true = df.loc[nan_indices, col]

    squared_error = (y_pred - y_true.values)
** 2
    error_list[col_idx] = squared_error

rmse_per_feature = {
    test.columns[j]:
np.sqrt(np.mean(error_list[j])).astype(float)
for j in range(len(test.columns))
}
```

D. Evaluasi dan Analisis Galat

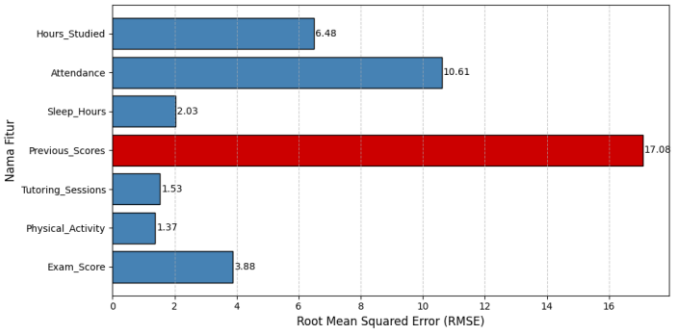
Pada gambar di bawah, terlihat tujuh histogram yang menggambarkan distribusi residual dari model Decision Tree untuk berbagai fitur yang berbeda, seperti Hours_Studied, Attendance, Sleep_Hours, Previous_Scores, Tutoring_Sessions, Physical_Activity, dan Exam_Score. Pada setiap histogram, dapat terlihat distribusi residual yang diwakili oleh batang berwarna biru dengan kurva distribusi normal yang di-*overlay* menggunakan garis merah. Distribusi residual ini mengindikasikan bagaimana perbedaan antara nilai prediksi dan nilai aktual untuk setiap fitur.



Gambar 4. Histogram Residual setiap Fitur

Sebagian besar fitur menunjukkan distribusi residual yang cukup mirip dengan distribusi normal. Hal ini mengindikasikan bahwa model pohon keputusan dapat memberikan prediksi yang cukup baik untuk fitur-fitur tersebut. Namun, ada beberapa histogram yang menunjukkan ketidaksimetrisan atau puncak tajam, seperti pada fitur Tutoring_Sessions, yang mengindikasikan adanya masalah dalam kemampuan model untuk menangkap variabilitas data dengan baik, atau menunjukkan adanya *overfitting*. Pada fitur-fitur seperti Attendance dan Sleep_Hours, distribusi residual menunjukkan variasi yang lebih besar, yang dapat menunjukkan bahwa model kesulitan menangkap pola yang lebih kompleks dalam data. Secara keseluruhan, distribusi residual ini memberikan wawasan penting mengenai seberapa baik model Decision Tree dalam memprediksi nilai pada masing-masing fitur, serta menunjukkan di mana model mungkin perlu disempurnakan.

Selanjutnya, ditampilkan RMSE setiap fitur agar dapat mengevaluasi lebih jelas. Terlihat bahwa fitur Previous_Scores memiliki RMSE tertinggi. Hal ini bisa saja terjadi mengingat nilai yang dihilangkan adalah 28% dari keseluruhan data sehingga akan mengurangi variabilitas prediksi model.



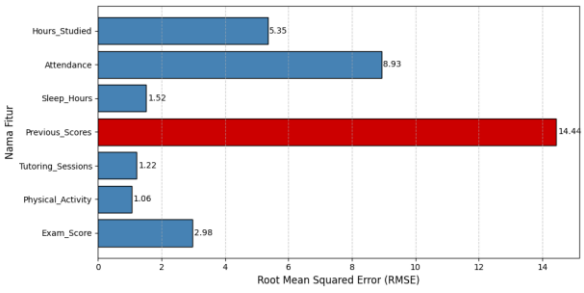
Gambar 5. RMSE Setiap Fitur

Walaupun terdapat kekurangan pada satu fitur yang sudah dijelaskan di atas, model ini sudah terbilang cukup baik berdasarkan skor RMSE dari fitur-fitur yang lain.

E. Hyperparameter Tuning

Pada implementasi ini, dilakukan proses *hyperparameter tuning* untuk menyempurnakan kinerja model dengan menggunakan pendekatan Grid Search. *Hyperparameter tuning* bertujuan untuk menemukan kombinasi parameter terbaik yang menghasilkan galat terkecil dalam proses prediksi. Parameter yang disesuaikan meliputi `max_depth`, `min_split`, dan `min_leaf`. Pada eksperimen ini, nilai `max_depth` divariasikan dari 1 hingga 8, `min_split` dari 2 hingga 4, dan `min_leaf` dari 1 hingga 4.

Proses Grid Search dilakukan dengan mencoba semua kombinasi parameter tersebut. Pada setiap iterasi, model dilatih menggunakan data latih dengan parameter tertentu, lalu diuji pada data uji untuk menghasilkan prediksi. Proses ini dilakukan sebanyak $8 \times 3 \times 4 = 96$ kali.



Gambar 6. RMSE setelah Hyperparameter Tuning

Hasil eksperimen menunjukkan bahwa kombinasi hyperparameter terbaik untuk model yang diuji adalah dengan `max_depth` sebesar 5, `min_split` sebesar 2, dan `min_leaf` sebesar 4. Kombinasi ini menghasilkan nilai rata-rata Root Mean Squared Error (RMSE) keseluruhan terkecil, yaitu 4.95. Dengan menggunakan kombinasi parameter ini, model dapat dioptimalkan untuk memberikan hasil prediksi yang lebih akurat, menjadikannya konfigurasi yang direkomendasikan untuk digunakan dalam pengujian atau implementasi lebih lanjut.

V. KESIMPULAN

Imputasi data nonlinear menggunakan pohon keputusan yang dimodifikasi dapat menjadi salah satu opsi dari sekian banyaknya teknik imputasi yang sudah ada. Dengan mempertimbangkan kompleksitas, pohon keputusan terbilang algoritma yang cukup efisien dengan kompleksitas total untuk mengimputasi semua fitur adalah $O(d^2n^2 \log_2 n)$ dan cukup *robust* dalam menghadapi kasus hubungan data nonlinear dengan rata-rata nilai galat secara keseluruhan 4.95.

VI. LAMPIRAN

Repositori Github: https://github.com/fathurwithyou/Matdis-MultivariateImputation

VII. PENUTUP

Ucapan terima kasih penulis berikan kepada Tuhan Yang Maha Esa, Institut Teknologi Bandung, Prodi Teknik Informatika, dan dosen mata kuliah IF2120 Matematika Diskrit, Bapak Rinaldi Munir. Diharapkan makalah ini dapat bermanfaat bagi pembaca dan dapat dikembangkan lebih lanjut. Seperti kata Mushashi dalam blog Rinaldi Munir, "*Hidup lebih dari harus sekedar tetap hidup. Masalahnya, bagaimana membuat hidup itu bermakna, memancarkan cahaya gemilang ke masa depan, sekalipun harus mengorbankan hidup itu sendiri. Bila ini sudah tercapai, maka tidak ada bedanya dengan berapa lama hidup itu.*"

REFERENCES

[1] Rinaldi Munir, "Graf (Bag.1)." Diakses: Dec. 25, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

[2] Rinaldi Munir, "Pohon (Bag. 1)." Diakses: Dec. 25, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/23-Pohon-Bag1-2024.pdf

[3] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986, doi: 10.1007/BF00116251.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 25 Desember 2023

Muhammad Fathur Rizky (13523105)