Optimasi *Hyperparameter Tuning* dengan Iterative Deepening A* (IDA*)

Filbert Engyo - 13523163
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: filbert.engyo7@gmail.com, 13523163@std.stei.itb.ac.id

***Abstract*—Hyperparameter tuning merupakan langkah krusial dalam meningkatkan performa model *machine learning* dengan menentukan kombinasi nilai hyperparameter terbaik sebelum proses pelatihan dimulai. Penyesuaian hyperparameter seperti *learning rate*, kekuatan regularisasi, jumlah fitur, dan parameter pada model ensemble seperti *random forest* dan *gradient boosting* sangat memengaruhi kemampuan model dalam melakukan generalisasi. Dalam penelitian ini, dilakukan eksperimen *hyperparameter tuning* pada model XGBoost menggunakan algoritma *Iterative Deepening A** (IDA*), yang memadukan strategi eksplorasi mendalam dari *depth-first search* dengan efisiensi heuristik dari algoritma A*. IDA* memanfaatkan fungsi evaluasi berbasis performa model dan pembatasan kedalaman untuk mengarahkan proses pencarian secara efisien dan terkontrol, sehingga menghindari eksplorasi konfigurasi yang tidak menjanjikan dan potensi *overfitting*. Kinerja IDA* dibandingkan dengan metode konvensional RandomizedSearchCV guna mengevaluasi efektivitasnya dalam menjelajahi ruang *hyperparameter* dan meningkatkan akurasi model. Hasil perbandingan menunjukkan bahwa IDA* mampu menjadi alternatif yang efisien dan terarah dalam proses *tuning hyperparameter*.**

***Keywords*—Hyperparameter tuning, Model Machine Learning, XGBoost, Iterative Deepening A* (IDA*), Heuristik, Learning rate**

I. PENDAHULUAN

Hyperparameter tuning adalah proses memilih kombinasi *hyperparameter* terbaik untuk meningkatkan performa model *machine learning*. Berbeda dengan parameter model yang dipelajari selama pelatihan, *hyperparameter* ditentukan sebelum proses pelatihan dimulai dan memiliki pengaruh besar terhadap perilaku serta akurasi model. Contoh hyperparameter antara lain *learning rate*, jumlah pohon pada *random forest*, dan tingkat regularisasi dalam regresi. Proses tuning dilakukan dengan mengeksplorasi berbagai nilai *hyperparameter* guna menemukan kombinasi yang memberikan hasil terbaik pada data validasi.

Untuk model regresi, *hyperparameter* yang sering disesuaikan meliputi tingkat regularisasi, learning rate (terutama untuk metode yang menggunakan *gradient descent*), serta jumlah fitur yang digunakan. Sementara itu, pada model seperti *decision tree* dan *ensemble*, parameter penting termasuk kedalaman maksimum pohon, jumlah pohon, jumlah minimum sampel untuk memisahkan node, *learning rate* untuk *boosting*, serta rasio subsample dari data pelatihan. Selain itu, pemilihan jumlah fitur maksimum untuk pemisahan dan penerapan regularisasi juga penting untuk mencegah *overfitting*. Penyesuaian *hyperparameter* yang tepat diperlukan untuk menyeimbangkan kompleksitas model dan kemampuannya dalam melakukan generalisasi.

Proses pencarian kombinasi *hyperparameter* terbaik umumnya dilakukan melalui eksplorasi sistematis terhadap ruang parameter. Teknik seperti *grid search* dan random search merupakan metode konvensional yang digunakan untuk menyampel ruang tersebut secara terstruktur maupun acak. Namun, pendekatan cerdas seperti *Iterative Deepening A** (IDA*) juga dapat diterapkan untuk tuning. IDA* menggabungkan pendekatan pencarian mendalam dari depth-first search dengan efisiensi heuristik dari algoritma A*. Dalam konteks *tuning*, IDA* menggunakan fungsi heuristik berbasis performa model (seperti akurasi atau error pada data validasi) dan membatasi kedalaman eksplorasi agar tidak terlalu banyak mengevaluasi konfigurasi yang kurang menjanjikan.

Dengan memanfaatkan fungsi biaya yang dirancang berdasarkan performa model, IDA* secara bertahap memperluas ruang pencarian melalui peningkatan batas kedalaman. Pendekatan ini memungkinkan eksplorasi konfigurasi hyperparameter secara efisien, sambil menjaga kualitas tuning. Strategi ini juga lebih terarah dibandingkan pencarian acak dan membantu menghindari *overfitting* dengan membatasi eksplorasi pada konfigurasi yang tidak potensial.

Dalam penelitian ini, digunakan dataset yang telah melalui proses pembersihan dan pra-pemrosesan menyeluruh sebagai dasar untuk eksperimen *tuning hyperparameter* menggunakan model XGBoost. Algoritma *Iterative Deepening A** (IDA*) diterapkan untuk menemukan kombinasi hyperparameter yang optimal, dengan memanfaatkan pendekatan heuristik dan pembatasan kedalaman untuk memprioritaskan konfigurasi yang paling menjanjikan. Efektivitas IDA* kemudian dievaluasi dengan membandingkannya terhadap metode RandomizedSearchCV. Perbandingan ini bertujuan menunjukkan keunggulan IDA* dalam menjelajahi ruang *hyperparameter* secara efisien dan meningkatkan performa model secara keseluruhan.

II. DASAR TEORI

A. Algoritma *Iterative Deepening Search (IDS)*

Iterative Deepening Search (IDS) adalah metode pencarian yang menggabungkan kelebihan dari Breadth-First Search (BFS) dan Depth-First Search (DFS). Teknik ini melakukan serangkaian DFS dengan batas kedalaman yang terus meningkat hingga solusi ditemukan. Pendekatan ini menjamin completeness dan optimality seperti BFS, tetapi dengan kompleksitas ruang yang rendah seperti DFS. Meskipun simpul-simpul atas akan dibangkitkan berulang kali, hal ini dianggap efisien karena sebagian besar solusi berada di level bawah dalam pohon pencarian.

Pada setiap iterasi, dilakukan DFS hingga kedalaman tertentu. Jika tujuan belum ditemukan, kedalaman batas dinaikkan satu level. Proses ini terus dilakukan hingga solusi ditemukan. Meskipun simpul-simpul pada level atas dievaluasi ulang, waktu yang terbuang relatif kecil karena sebagian besar simpul berada di kedalaman bawah dari pohon status.

B. Algoritma *A**

A* adalah algoritma pencarian berbasis graf yang terkenal karena efisiensi dan optimalitasnya. Ia menggunakan fungsi evaluasi yang memperhitungkan dua hal: biaya aktual untuk mencapai node tertentu (g(n)) dan estimasi biaya dari node tersebut ke tujuan (h(n), disebut heuristik).

$$f(n) = g(n) + h(n)$$

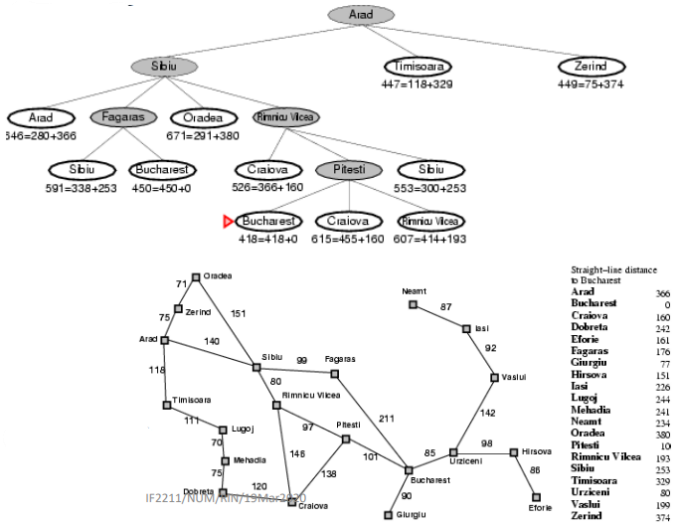
Dengan rincian dibawah:

- g(n): biaya dari start ke node n (path cost sejauh ini).
- h(n): estimasi biaya dari node n ke tujuan (heuristik).
- f(n): estimasi total biaya dari start ke goal melalui n.

Heuristik yang digunakan harus memenuhi persyaratan yaitu:

- *Admissible*: Tidak pernah melebih-lebihkan biaya sesungguhnya ke tujuan. Ini menjamin optimalitas.
- *Consistent* (Monotonik): $h(n) \leq cost(n, n') + h(n')$ untuk semua n dan n'. Ini menjamin A* tidak akan memproses ulang node yang sama.

Mekanisme kerjanya adalah A* menyimpan semua simpul yang telah dikunjungi beserta nilai f(n)-nya, dan memperluas simpul dengan nilai $f(n)$ terkecil lebih dahulu. Ini menjadikan A* sebagai kompromi antara pencarian cepat dan pencarian tepat, karena mengejar solusi yang seolah-olah menjanjikan berdasarkan heuristik dan informasi biaya yang telah diketahui.



Gambar 1. Ilustrasi Penerapan Algortima A*

Sumber:[1]

C. Algoritma IDA*

IDA* muncul sebagai bentuk evolusi dari A*, mengadopsi pemikiran strategisnya dalam pemanfaatan fungsi f(n), namun menambahkan filosofi penghematan sumber daya melalui teknik pendalaman iteratif. Dalam IDA*, pencarian tidak langsung menuju solusi dengan memproses semua simpul berdasarkan urutan f(n) seperti pada A*, melainkan menetapkan batas f(n) tertentu dan menelusuri simpul menggunakan strategi penelusuran dalam (DFS), sejauh nilai f(n) tidak melebihi batas tersebut.

Pada setiap iterasi, IDA* memperluas metode pencariannya, mengangkat batas f(n) berdasarkan nilai minimum yang sebelumnya tidak tercapai. Dengan cara ini, pencarian tetap berjalan terarah dan efisien, sambil secara bertahap mendekati solusi optimal. Kombinasi antara strategi penelusuran mendalam dan fungsi evaluasi menjadikan IDA* sangat adaptif untuk permasalahan yang besar namun memiliki keterbatasan dalam memori atau penyimpanan.

IDA* memandang eksplorasi ruang solusi bukan sebagai perlombaan cepat, melainkan proses bertahap yang disiplin, yang mengandalkan informasi heuristik namun tidak melupakan keterbatasan sumber daya. Algoritma ini seolah berkata, "Aku akan mencapainya, perlahan tapi dengan keyakinan dan panduan yang masuk akal."

D. *Machine Learning*

Machine learning merupakan bagian dari kecerdasan buatan yang menitikberatkan pada penciptaan algoritma dan model statistik yang memungkinkan komputer menyelesaikan tugas-tugas tertentu tanpa harus diprogram secara langsung. Sistem ini memperoleh kemampuan dari data yang diberikan, dengan mempelajari pola-pola dan kemudian mengambil keputusan berdasarkan hasil pembelajaran tersebut. Secara teori, bidang ini terbagi ke dalam beberapa konsep inti, yaitu:

1. *Supervised Learning*

Konsep ini melibatkan pelatihan model pada dataset yang sudah dilabeli, artinya data memiliki fitur input dan label target yang sesuai. Model belajar untuk memprediksi output berdasarkan data input. Teknik yang umum digunakan meliputi regresi linier, regresi logistik, support vector machines, dan jaringan saraf.

2. *Unsupervised Learning*

Konsep ini melibatkan pelatihan model pada data tanpa label. Tujuannya adalah untuk menemukan pola tersembunyi atau struktur intrinsik dalam data. Teknik yang digunakan termasuk clustering dan reduksi dimensi.

3. *Reinforcement Learning*

Konsep ini melatih agen untuk membuat serangkaian keputusan dengan memberikan penghargaan pada perilaku yang diinginkan. Agen belajar mencapai tujuan dalam lingkungan yang tidak pasti dan mungkin kompleks. Teknik yang digunakan antara lain *Q-learning* dan metode *policy gradient.*

4. *Semi-Supervised Learning*

Konsep ini menggabungkan data berlabel dan tak berlabel untuk meningkatkan akurasi pembelajaran. Pendekatan ini memanfaatkan sebagian kecil data berlabel untuk memahami struktur data tak berlabel dengan lebih baik.

Dalam proses pemodelan data pada *machine learning*, langkah awal yang dilakukan adalah pra-pemrosesan data. Tahapan ini melibatkan pembersihan dan transformasi data mentah agar layak untuk dianalisis, seperti menangani data yang hilang, pencilan, serta mengubah skala nilai numerik dan mengonversi data kategorikal ke bentuk numerik. Setelah itu, data biasanya dibagi ke dalam tiga bagian: data pelatihan, validasi, dan pengujian, agar model dapat dievaluasi secara menyeluruh dan adil.

Setelah data siap, langkah berikutnya adalah seleksi dan rekayasa fitur, yang bertujuan untuk memilih fitur paling informatif serta menciptakan fitur baru guna meningkatkan kualitas prediksi. Ini bisa mencakup pembuatan kombinasi antar variabel atau pengubahan fitur menjadi bentuk yang lebih representatif. Selanjutnya, dilakukan pemilihan algoritma dan pelatihan model. Jenis model yang dipilih tergantung pada karakteristik data dan tujuan analisis. Beberapa algoritma yang umum digunakan adalah regresi linier, pohon keputusan, random forest, SVM, dan jaringan saraf. Proses pelatihan melibatkan penyesuaian parameter model berdasarkan data pelatihan guna meminimalkan kesalahan prediksi.

E. *Hyperparameter Tuning*

Hyperparameter merupakan konfigurasi eksternal yang harus ditentukan sebelum proses pelatihan model dimulai. Contohnya adalah learning rate pada algoritma *gradient descent*, jumlah pohon pada *random forest*, serta kedalaman maksimum dari decision tree. Pemilihan nilai hyperparameter ini sangat mempengaruhi kecepatan konvergensi model, kemampuan generalisasi, serta kinerja keseluruhannya. Oleh karena itu, proses penyetelan hyperparameter sangat penting karena secara langsung memengaruhi kompleksitas, kapasitas model, serta kekuatan prediktif dan efisiensinya.

Beberapa pendekatan umum yang digunakan untuk melakukan tuning hyperparameter antara lain:

- *Grid Search*

  Metode ini melakukan pencarian secara menyeluruh pada kombinasi nilai *hyperparameter* yang telah ditentukan sebelumnya. Meskipun metode ini cukup sederhana dan dapat memberikan hasil yang baik, *grid search* cenderung memerlukan waktu dan sumber daya komputasi yang besar, terutama jika digunakan pada dataset yang besar dan model yang kompleks.

- *Random Search*

  Alih-alih mencoba semua kemungkinan kombinasi, *random search* secara acak memilih sejumlah konfigurasi *hyperparameter* dari ruang pencarian yang telah ditentukan. Penelitian menunjukkan bahwa pendekatan ini seringkali lebih efisien dibanding *grid search*, karena mampu menemukan konfigurasi yang cukup baik dalam waktu yang lebih singkat.

- *Bayesian Optimization*

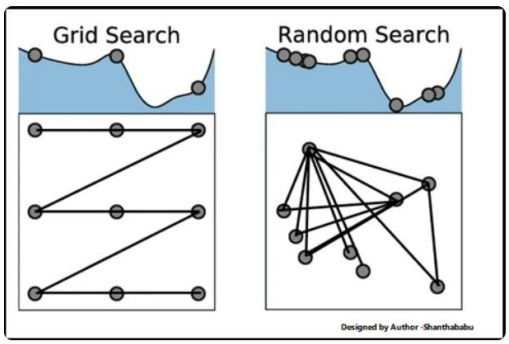
  Pendekatan ini memanfaatkan model probabilistik untuk memperkirakan performa berbagai konfigurasi *hyperparameter*, kemudian memilih kandidat yang paling menjanjikan untuk dievaluasi. Metode ini menyeimbangkan antara eksplorasi dan eksploitasi dalam menjelajahi ruang pencarian *hyperparameter*.

- *Evolutionary Algorithms*

  Algoritma optimisasi yang terinspirasi dari proses seleksi alam. Algoritma ini bekerja dengan cara mengembangkan populasi solusi kandidat melalui proses mutasi, *crossover*, dan seleksi secara bertahap.

- *Gradient-Based Optimization*

  Beberapa teknik lanjutan menggunakan informasi gradien dalam mengoptimalkan *hyperparameter*. Metode ini hanya dapat digunakan jika *hyperparameter* yang ingin disesuaikan bersifat terdiferensiasi, dan umumnya cukup efisien untuk model-model tertentu.



Gambar 2. Visualisasi *Grid Search* & *Random Search*
Sumber: [6]

Kinerja dari masing-masing konfigurasi hyperparameter biasanya dievaluasi menggunakan teknik cross-validation pada validation set. Metode evaluasi yang digunakan dapat berupa akurasi, presisi, *recall*, F1 *score*, atau *mean squared error*, tergantung pada jenis masalah yang dihadapi, apakah klasifikasi atau regresi. Pemilihan metrik ini akan menjadi acuan dalam proses pencarian kombinasi *hyperparameter* terbaik.

III. IMPLEMENTASI

Secara garis besar, langkah implementasi akan diawali akan dimulai dengan pengolahan data awal seperti pembersihan (cleansing) dan langkah persiapan lain agar data dapat optimal untuk digunakan dalam *hyperparameter tuning*. Lalu, hasil dari data yang sudah optimal akan diolah dengan RandomizeCV, GridCV, dan IDA* yang kemudian akan dibandingkan hasil akurasi setelah dan sebelum dilakukan tuning antar satu sama lain.

A. *Dataset*

Dataset yang digunakan untuk pengujian *hyperparameter tuning* ini adalah data sewaan hotel yang memiliki 94.546 baris dan 30 kolom. Kolom mencakup informasi yaitu ID sewa, nama hotel, durasi penyewaan, waktu kedatangan, kategori hari penginapan (hari kerja atau hari libur), jumlah orang per kategori (dewasa, anak, bayi), makanan, asal negara, sekmen pasar, saluran distribusi , apakah pengunjung pernah menginap sebelumya, pernah melakukan pembatalan , penyewaan sebelumnya yang tidak dibatalkan , tipe kamar yang dipesan, tipe kamar yang diberikan, perubahan pada pesanan, tipe deposit, agen, perusahaan, waktu untuk menunggu panggilan, tipe pelanggan, alamat rumah, jumlah mobil yang diperlukan untuk parkir, status penyewaan. Berdasarkan dataset ini, akan diuji seberapa mungkin seseorang membatalkan penyewaannya.

	id	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_w
0	0	Resort Hotel	312	2017	March	10	5	2	
1	1	City Hotel	2	2015	December	51	18	0	
2	2	City Hotel	41	2016	March	14	31	0	
3	3	Resort Hotel	228	2016	August	36	29	2	
4	4	City Hotel	128	2017	May	19	13	0	
...	...	...	...	...	...	...	...	...	...
94541	94541	City Hotel	26	2016	October	40	1	2	
94542	94542	City Hotel	269	2016	November	48	24	0	
94543	94543	City Hotel	302	2015	August	33	15	2	
94544	94544	City Hotel	53	2017	June	25	19	1	
94545	94545	City Hotel	86	2015	October	40	3	2	

94546 rows × 30 columns

Gambar 3. Tabel Rincian Dataset
(Sumber: kode penulis)

B. *Pra-pemrosesan Data*

1) *Split Data*

Pra-pemrosesan data diawalai dengan pemisahan data dengan rasio 3:2 yaitu untuk 60% dataset akan menjadi *train* yang akan dijadikan pelatihan untuk model, sedangkan sisa 40% akan dijadikan *base* yang akan menjadi basis perbandingan untuk performa model terhadap data yang sebenarnya.

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1,test_size = 0.6)
for train_indices,val_indices in split.split(train,train[["reservation_status"]]):
    trainset = train.loc[train_indices]
    valset = train.loc[val_indices]

numeric_columns = [col for col in train.select_dtypes(include=['int64', 'float64']).columns if train[col].nunique() > 2]
```
0.6s

Gambar 4. Kode untuk Split Data
(Sumber: kode penulis)

2) *Data Cleansing*

Setelah dilakukan pra-pemrosesan data dengan dipisah, lalu data dibersihkan dari segala kekurangannya yang mencakup nilai kosong agar data bisa lebih mengompres data menjadi lebih kecil secara menyeluruh yang dapat meringankan proses pelatihan, tapi ada kondisi yang perlu dipertahankan yaitu untuk kolom ‘agent’ dan ‘country’ memiliki peran signifikan dalam keseluruhan data, sehingga menghilangkan mereka akan menghilangkan sebagian besar dari keseluruhan data, sehingga harus dipertahankan. Ini dilakukan untuk mencegah penghapusan data secara berlebihan yang dimana, permasalahan data kotor yang ada akan dibawa ke pra-pemrosesan selanjutnya.

```
def cleanse(df):
    cleaned_df = df.copy()
    cleaned_df = cleaned_df.replace("", pd.NA)
    cols_to_check = [col for col in cleaned_df.columns if col != 'company' and col != 'agent']
    cleaned_df = cleaned_df.dropna(subset=cols_to_check)
    return cleaned_df
trainset = cleanse(trainset)
```
0.0s

Gambar 5. Kode untuk Pembersihan Data
(Sumber: kode penulis)

3) *Impute Data*

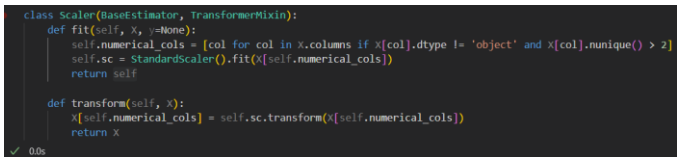
Bagian permasalahan tadi mengenai bagian kosong yang disebabkan oleh ketergantungan kolom ‘agent’ dan ‘country’, akan diatasi dengan *impute* untuk menghubungkan data-data yang kosong dengan diisi kembali atau digantikan untuk membentuk dataset yang lengkap terisi secara menyeluruh dan bisa digunakan untuk pelatihan model tanpa mungkin terjadi ketidaklengkapan data.

```
class Impute(BaseEstimator, TransformerMixin):
    def __init__(self, strategy='most_frequent'):
        self.strategy = strategy
    def fit(self, X, y=None):
        self.imputer_dict = {}
        num_cols = X.select_dtypes(include=['float64', 'int64']).columns
        cat_cols = X.select_dtypes(exclude=['float64', 'int64']).columns
        for col in num_cols:
            num_imputer = SimpleImputer(missing_values=np.nan, strategy=self.strategy)
            num_imputer.fit(X[col].values.reshape(-1, 1))
            self.imputer_dict[col] = num_imputer
        for col in cat_cols:
            cat_imputer = SimpleImputer(missing_values=np.nan, strategy=self.strategy)
            cat_imputer.fit(X[col].values.reshape(-1, 1))
            self.imputer_dict[col] = cat_imputer
        return self
    def transform(self, X):
        new_X = X.copy()
        for col, imputer in self.imputer_dict.items():
            if col in new_X.columns:
                new_X[col] = imputer.transform(new_X[col].values.reshape(-1, 1)).ravel()
            else:
                pass
        return new_X
```
0.0s

Gambar 6. Kode untuk *Impute* Data
(Sumber: kode penulis)

4) *Scaling Data*

Bagian ini akan melakukan standardisasi akan rentang data independen yang bervariasi dalam suatu dataset agar data yang digunakan untuk pelatihan model tidak menimbulkan keambiguan karena pelatihan model *machine learning* itu sendiri sangat rentan terhadap skala.



```
class Scaler(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        self.numerical_cols = [col for col in X.columns if X[col].dtype != 'object' and X[col].nunique() > 2]
        self.sc = StandardScaler().fit(X[self.numerical_cols])
        return self

    def transform(self, X):
        X[self.numerical_cols] = self.sc.transform(X[self.numerical_cols])
        return X
```

0.0s

Gambar 7. Kode untuk *Scaling* Data
(Sumber: kode penulis)

5) Drop Data

Bagian ini akan melakukan *drop* atau penghapusan kolom dengan data yang terlalu banyak nilai kosong dan/atau berisi informasi yang tidak relevan untuk pelatihan model.

```
class Drop(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.drop(["reserved_room_type","assigned_room_type","id","country"], axis=1, errors="ignore")
        return X
```

0.0s

Gambar 8. Kode untuk *Drop* Data
(Sumber: kode penulis)

6) Encode Data

Bagian ini akan mengubah format data dalam bentuk kode yang bisa diberikan kepada model untuk memprediksi dan/atau menjawab pertanyaan yang berkaitan dengan dataset yang diberikan.

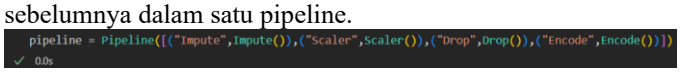
```
class Encode(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        self.encoders = {}
        self.categories = {
            'arrival_date_month': [
                'January', 'February', 'March', 'April', 'May', 'June',
                'July', 'August', 'September', 'October', 'November', 'December'
            ],
            'hotel': ["Resort Hotel", "City Hotel"],
            'meal': ["HB", "BB", "FB", "SC", "Undefined"],
            'market_segment': ["Groups", "Online TA", "Offline TA/TO", "Direct", "Aviation", "Corporate", "Complementary", "Undefined"],
            'distribution_channel': ['TA/TO', 'Direct', 'Corporate', 'GDS', 'Undefined'],
            'deposit_type': ["No Deposit", "Non Refund", "Refundable"],
            'customer_type': ['Transient-Party', 'Transient', 'Contract', 'Group']
        }
        for column, categories in self.categories.items():
            try:
                encoder = OneHotEncoder(categories=[categories], drop=None, sparse_output=False, handle_unknown='ignore')
            except TypeError:
                encoder = OneHotEncoder(categories=[categories], drop=None, sparse=False, handle_unknown='ignore')
            encoder.fit(X[[column]])
            self.encoders[column] = encoder
        return self

    def transform(self, X):
        encoded_X = X.copy()
        for column, encoder in self.encoders.items():
            matrix = encoder.transform(X[[column]])
            column_names = encoder.get_feature_names_out([column])
            for i in range(len(matrix.T)):
                encoded_X[column_names[i]] = matrix.T[i]
            encoded_X.drop(column, axis=1, inplace=True)
        return encoded_X
```

Gambar 9. Kode untuk *Encode* Data
(Sumber: kode penulis)

C. Pemrosesan Data

Untuk mengawali pemrosesan, diperlukan pembentukan *pipeline* yang dengan memanggil seluruh pra-proses sebelumnya dalam satu pipeline.



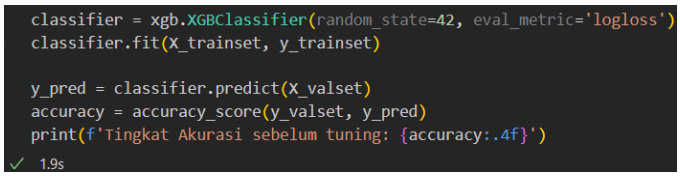
```
pipeline = Pipeline([("Impute",Impute()),("Scaler",Scaler()),("Drop",Drop()),("Encode",Encode())])
```

0.0s

Gambar 10. Kode Deklarasi *Pipeline*
(Sumber: kode penulis)

Untuk mempermudah pengujian, proses dibagi menjadi beberapa kasus yang akan dibandingkan. Pertama adalah tanpa *tuning* sebagai acuan, lalu kasus kedua adalah hasil optimasi *tuning* dengan RandomizeSearchCV, lalu kasus ketiga adalah hasil optimasi *tuning* dengan GridSearchCV, dan kasus yang keempat atau kasus yang terakhir adalah hasil optimasi *tuning* dengan IDA*. Acuan ini diperlukan untuk membuktikan apakah proses tuning berhasil meningkatkan akurasi atau tidak dan seberapa besar perbedaan tingkat akurasinya.

Kasus pertama menggunakan XGBoost dengan pengaturan *hyperparameter* bawaan. Model ini berfungsi sebagai titik awal atau acuan untuk mengevaluasi kinerja XGBoost tanpa melakukan penyesuaian hyperparameter. Dengan menilai performa model dalam kondisi default, kita dapat memperoleh tolok ukur yang berguna untuk dibandingkan dengan model-model lain yang telah melalui proses tuning. Baseline ini memberikan gambaran seberapa besar pengaruh optimasi hyperparameter terhadap peningkatan performa model.



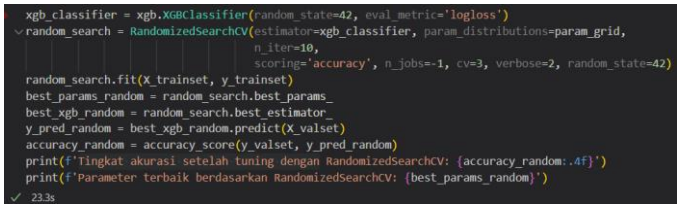
```
classifier = xgb.XGBClassifier(random_state=42, eval_metric='logloss')
classifier.fit(X_trainset, y_trainset)

y_pred = classifier.predict(X_valset)
accuracy = accuracy_score(y_valset, y_pred)
print(f'Tingkat Akurasi sebelum tuning: {accuracy:.4f}')
```

1.9s

Gambar 11. Kode untuk Kasus Sebelum *Tuning*
(Sumber: kode penulis)

Pada kasus kedua, model XGBoost digunakan dengan hyperparameter yang diperoleh melalui metode RandomizedSearchCV dari pustaka scikit-learn. Metode ini bekerja dengan memilih kombinasi hyperparameter secara acak dari ruang parameter yang telah ditentukan, lalu mengevaluasi sejumlah konfigurasi tersebut. Berbeda dengan pencarian *grid* yang menguji semua kemungkinan, pendekatan ini lebih efisien karena dapat menemukan kombinasi yang optimal dalam waktu yang lebih singkat. Oleh karena itu, RandomizedSearchCV sering menjadi pilihan praktis dalam proses tuning hyperparameter, terutama ketika ruang parameter sangat luas atau waktu komputasi terbatas.

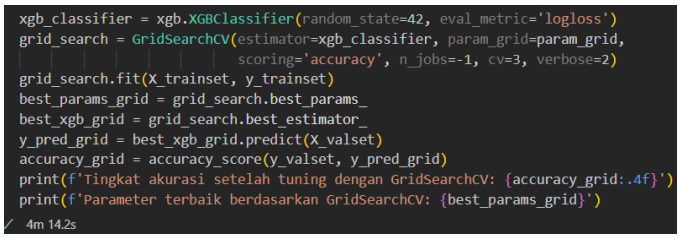


```
xgb_classifier = xgb.XGBClassifier(random_state=42, eval_metric='logloss')
random_search = RandomizedSearchCV(estimator=xgb_classifier, param_distributions=param_grid,
                                   n_iter=10,
                                   scoring='accuracy', n_jobs=-1, cv=3, verbose=2, random_state=42)
random_search.fit(X_trainset, y_trainset)
best_params_random = random_search.best_params_
best_xgb_random = random_search.best_estimator_
y_pred_random = best_xgb_random.predict(X_valset)
accuracy_random = accuracy_score(y_valset, y_pred_random)
print(f'Tingkat akurasi setelah tuning dengan RandomizedSearchCV: {accuracy_random:.4f}')
print(f'Parameter terbaik berdasarkan RandomizedSearchCV: {best_params_random}')
```

23.3s

Gambar 12. Kode untuk *Kasus Tuning Random*
(Sumber: kode penulis)

Pada kasus ketiga menggunakan model XGBoost dengan hyperparameter yang diperoleh melalui GridSearchCV, sebuah alat dari scikit-learn untuk seleksi model. Dalam metode ini, pencarian dilakukan secara menyeluruh terhadap seluruh kombinasi yang mungkin dari nilai-nilai hyperparameter yang telah ditentukan. Pendekatan ini mengevaluasi setiap konfigurasi secara sistematis untuk menemukan kombinasi terbaik yang memberikan kinerja optimal pada model. Meskipun memerlukan waktu komputasi yang lebih besar, metode ini memastikan bahwa seluruh ruang parameter telah dieksplorasi secara lengkap.



```
xgb_classifier = xgb.XGBClassifier(random_state=42, eval_metric='logloss')
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid,
                           scoring='accuracy', n_jobs=-1, cv=3, verbose=2)
grid_search.fit(X_trainset, y_trainset)
best_params_grid = grid_search.best_params_
best_xgb_grid = grid_search.best_estimator_
y_pred_grid = best_xgb_grid.predict(X_valset)
accuracy_grid = accuracy_score(y_valset, y_pred_grid)
print(f'Tingkat akurasi setelah tuning dengan GridSearchCV: {accuracy_grid:.4f}')
print(f'Parameter terbaik berdasarkan GridSearchCV: {best_params_grid}')
```

4m 14.2s

Gambar 13. Kode untuk *Kasus Tuning Grid*
(Sumber: kode penulis)

Pada kasus keempat, *tuning hyperparameter* dilakukan menggunakan pendekatan IDA* (Iterative Deepening A*), yaitu

metode pencarian yang menggabungkan keunggulan dari pencarian depth-first (penelusuran berdasarkan kedalaman) dengan efisiensi pencarian berbasis heuristik. Pendekatan ini digunakan untuk menemukan kombinasi hyperparameter terbaik pada model klasifikasi, dalam hal ini menggunakan RandomForestClassifier sebagai dasar evaluasi skor, lalu hasil tuning diaplikasikan pada model XGBoost.

Metode IDA* bekerja dengan melakukan pencarian bertahap hingga kedalaman tertentu, yang meningkat secara iteratif. Pada setiap level kedalaman, pencarian dilakukan terhadap kombinasi nilai hyperparameter berdasarkan urutan yang ditentukan dalam param_grid. Untuk setiap konfigurasi, fungsi evaluasi menghitung akurasi rata-rata menggunakan cross-validation, dan digunakan sebagai biaya aktual (actual *cost*).

Untuk mempercepat pencarian dan menghindari eksplorasi konfigurasi yang kurang menjanjikan, digunakan heuristik *Manhattan Distance* terhadap posisi nilai parameter dibandingkan posisi tengah grid-nya. Heuristik ini menjadi estimasi cost tambahan untuk menentukan seberapa "jauh" konfigurasi saat ini dari yang dianggap ideal. Nilai heuristik ditambahkan ke negative score untuk membentuk total *f-cost* seperti dalam algoritma A*.

Selain itu, skenario ini menerapkan batas waktu (timeout) selama 60 detik, yang menghentikan pencarian bila sudah terlalu lama. Ini memberikan kontrol atas waktu komputasi yang diperlukan.

Setelah kombinasi terbaik ditemukan, model XGBoost dilatih ulang menggunakan parameter hasil tuning, kemudian dievaluasi pada dataset validasi untuk mengukur tingkat akurasinya.

```python
import time

def manhattan_heuristic(params, param_grid):
    distance = 0
    for key, value in params.items():
        grid = list(param_grid[key])
        mid_index = len(grid) // 2
        try:
            distance += abs(grid.index(value) - mid_index)
        except ValueError:
            distance += mid_index
    return distance

def calc_cost(params, X, y):
    rf = RandomForestClassifier(**params, random_state=42).fit(X, y)
    scores = cross_val_score(rf, X, y, cv=3, scoring=make_scorer(accuracy_score))
    return np.mean(scores)

def ida(param_grid, X, y, max_depth=5, timeout=60):
    best_score = float("-inf")
    best_params = {}
    start_time = time.time()

    def ids(params, keys, depth, threshold):
        nonlocal best_score, best_params
        # Cek timeout
        if time.time() - start_time > timeout:
            raise TimeoutError("Pencarian parameter dihentikan karena melebihi batas waktu.")
        if depth > threshold or not keys:
            score = calc_cost(params, X, y)
            f_cost = -score + manhattan_heuristic(params, param_grid)
            if score > best_score:
                best_score = score
                best_params = params.copy()
                print(f"Parameter baru berdasarkan IDA*: {best_params}, dengan Hasil: {best_score:.4f}")
            return f_cost

        next_key = keys[0]
        min_f = float("inf")
        for value in param_grid[next_key]:
            new_params = params.copy()
            new_params[next_key] = value
            f = ids(new_params, keys[1:], depth + 1, threshold)
            min_f = min(min_f, f)
        return min_f

    try:
        for depth in range(1, max_depth + 1):
            print(f"Depth limit: {depth}")
            ids({}, list(param_grid.keys()), 0, depth)
    except TimeoutError as e:
        print(str(e))

    return best_params

best_params_ida = ida(param_grid, X_trainset, y_trainset, max_depth=5, timeout=60)
best_xgb_ida = xgb.XGBClassifier(**best_params_ida, random_state=42, eval_metric='logloss')
best_xgb_ida.fit(X_trainset, y_trainset)
y_pred_ida = best_xgb_ida.predict(X_valset)
accuracy_ida = accuracy_score(y_valset, y_pred_ida)
print(f'Tingkat akurasi setelah tuning dengan IDA*: {accuracy_ida:.4f}')
print(f'Parameter terbaik berdasarkan IDA*: {best_params_ida}')
```

1m 4.6s

Gambar 14. Kode untuk *Kasus Tuning IDA**
(Sumber: kode penulis)

IV. PENGUJIAN & ANALISIS

Pengujian dilakukan dengan pengaturan konfigurasi yang didasarkan pada parameter XGBoost itu sendiri untuk menentukan performa model.

```python
param_grid = {
    'n_estimators': [80, 120, 200],
    'max_depth': [4, 5, 10],
    'learning_rate': [0.03, 0.07, 0.15],
    'subsample': [0.6, 0.75, 0.9],
    'colsample_bytree': [0.65, 0.85, 1.0]
}
```

0.0s

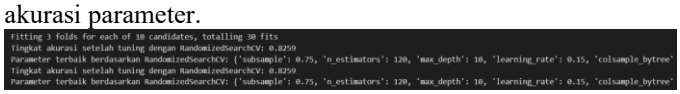
Gambar 14. Deklarasi Basis Parameter
(Sumber: kode penulis)

Berdasarkan kasus pertama, tanpa adanya proses tuning didapatkan hasil akurasi 0.8204 atau 82.04% yang tergolong dalam memuaskan karena termasuk dalam rentang persentase akurasi 70-90%.

```
Tingkat Akurasi sebelum tuning: 0.8204
```

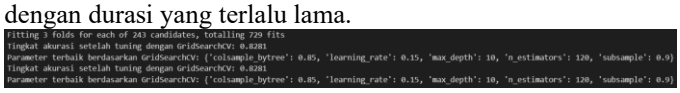
Gambar 14. Tingkat Akurasi Sebelum *Tuning*
(Sumber: kode penulis)

Untuk hasil kasus kedua yaitu setelah *tuning* dengan *RandomizeSearchCV* yang membutuhkan waktu kurang lebih 23 detik memberikan hasil tingkat akurasi sebesar 0.8259 atau 82.59% yang memastikan *tuning* berhasil meningkatkan akurasi parameter.



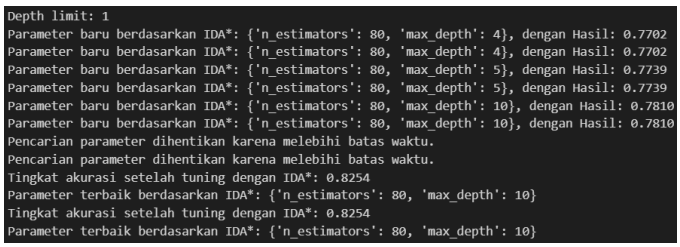
Gambar 15. Tingkat Akurasi Setelah Tuning Random
(Sumber: kode penulis)

Untuk hasil kasus ketiga yaitu setelah *tuning* dengan *GridSearchCV* yang membutuhkan waktu kurang lebih 4 menit 14 detik memberikan hasil tingkat akurasi sebesar 0.8281 atau 82.81% yang memastikan *tuning* berhasil meningkatkan akurasi parameter yang memastikan *exhaustive search* memberikan akurasi lebih optimal, tetapi tidak sebanding dengan durasi yang terlalu lama.



Gambar 16. Tingkat Akurasi Setelah *Tuning* Grid
(Sumber: kode penulis)

Untuk hasil kasus ketiga yaitu setelah *tuning* dengan *IDA** yang membutuhkan waktu kurang lebih 1 menit 4 detik memberikan hasil tingkat akurasi sebesar 0.8254 atau 82.54% yang memastikan *tuning* berhasil meningkatkan akurasi parameter yang memastikan *exhaustive search* memberikan akurasi lebih optimal dibandingkan dengan *random search*, tetapi ada kendala ketika ia tidak berhasil menemukan param baru bisa terjebak dalam loop sehingga harus dihentikan dengan durasi.



Gambar 17. Tingkat Akurasi Setelah *Tuning* IDA*
(Sumber: kode penulis)

V. KESIMPULAN

Berdasarkan seluruh hasil eksperimen, dapat disimpulkan bahwa tuning hyperparameter berperan penting dalam meningkatkan kinerja model machine learning, termasuk dalam kasus XGBoost. Pendekatan IDA* terbukti menjadi alternatif yang layak terhadap metode tuning konvensional, dengan memberikan hasil akurasi yang kompetitif dan waktu komputasi yang relatif efisien. Dibandingkan RandomizedSearchCV, IDA* mampu mengarahkan proses pencarian lebih sistematis melalui pemanfaatan heuristik, dan lebih hemat waktu daripada GridSearchCV yang membutuhkan eksplorasi menyeluruh. Dengan demikian, IDA* dapat dianggap sebagai metode yang efektif untuk hyperparameter tuning, khususnya ketika efisiensi dan keterbatasan sumber daya menjadi pertimbangan utama.

VI. APPENDIX

Link Github Repository:
https://github.com/filbertengyo/Optimasi_Hyperparameter_Tuning_dengan_IDA

ACKNOWLEDGMENT

Pertama-tama, penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini tepat waktu. Adapun tujuan penulisan makalah ini adalah sebagai bentuk pemenuhan tugas mata kuliah IF2211 Strategi Algoritma.

Dalam penyusunan ini penulis ingin berterima kasih kepada berbagai pihak yang telah mendukung dan mendorong pembuatan makalah ini. Oleh karena itu, saya menyampaikan terima kasih kepada:

1. Seluruh dosen pengampu mata kuliah Strategi Algoritma yang telah memberikan bimbingan dan dorongan untuk membuat makalah ini dan telah menyiapkan bahan ajar yang juga digunakan dalam makalah ini.
2. Orang tua yang senantiasa memberikan dukungan secara moral maupun material kepada anak-anaknya sehingga bisa seperti saat ini.

Demikian ucapan terima kasih penulis kepadaorang-orang yang mendukung dalam proses pembuatan makalah ini. Semoga dengan adanya makalah ini dapat memberikan gambaran dan sedikit pemikiran tentang masalah yang disampaikan.

REFERENCES

[1] R. Munir, “Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 2,” [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf. [Diakses pada 21 Juni 2025].

[2] R. Munir, “Penentuan rute (Route/Path Planning) - Bagian 1,” [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf. [Diakses pada 21 Juni 2025].

[3] R. Munir, “Penentuan rute (Route/Path Planning) - Bagian 2,” [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf. [Diakses pada 21 Juni 2025].

[4] A. Primanita, et al.. “Comparison of A* and Iterative Deepening A* algorithms for non-player character in Role Playing Game,” [Online]. Available: https://ieeexplore.ieee.org/document/8167134/similar. [Diakses pada 21 Juni 2025].

[5] A. Geron. “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition,” [Online]. Available: http://14.139.161.31/OddSem-0822-1122/Hands-On_Machine_Learning_with_Scikit-Learn-Keras-and-TensorFlow-2nd-Edition-Aurelien-Geron.pdf. [Diakses pada 21 Juni 2025].

[6] S. Pandian. “A comprehensive Guide on Hyperparameter Tuning and its Techniques,” [Online]. Available: https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/. [Diakses pada 21 Juni 2025]

[7] B. Priya. “Hyperparameter Tuning: GridSearchCV and RandomizedSearchCV, Explained,” [Online]. Available: https://www.kdnuggets.com/hyperparameter-tuning-gridsearchcv-and-randomizedsearchcv-explained. [Diakses pada 21 Juni 2025].

[8] W. Yusda. “Uniform Cost Search as a Strategy for Hyperparameter Optimization,” [Online].

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Makalah/Makalah-IF2211-Stima-2024%20(18).pdf. [Diakses pada 21 Juni 2025].

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, 24 Juni 2025

Filbert Engyo - 13523163