Bank Customer Churn Prediction using Random Forest and Decision Tree Algorithm

Elizabeth Hanov, 23031554055
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
elizabeth.23055@mhs.unesa.ac.id

Aghnia Alya Amarilla, 23031554102
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
aghnia.23102@mhs.unesa.ac.id

Nashita Erha Fitri, 23031554116
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
nashita.23116@mhs.unesa.ac.id

Dosen Pengampu:

Dr. Elly Matul Imah, M.Kom.
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
ellymatul@unesa.ac.id

Dinda Galuh Guminta, S.Stat., M.Stat.
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
dindaguminta@unesa.ac.id

***Abstract*—Prediksi *churn* pelanggan telah menjadi tugas penting di sektor perbankan untuk mengidentifikasi pelanggan yang kemungkinan akan menghentikan hubungannya dengan bank. Proyek ini berfokus pada pemanfaatan teknik *Machine Learning*, khususnya algoritma *Random Forest* dan *Decision Tree*, untuk memprediksi *churn* pelanggan. Data yang digunakan adalah dataset publik yang diambil dari Kaggle yang mencakup informasi 10.000 nasabah bank. Proses penelitian melibatkan beberapa tahapan yaitu pengumpulan data, *preprocessing*, pemodelan, prediksi, evaluasi dan deployment. Hasil evaluasi memperlihatkan bahwa model *Random Forest* memberikan performa terbaik dengan akurasi 78% sedangkan *Decision Tree* memiliki akurasi 69%. Kesimpulan menunjukkan bahwa *Random Forest* lebih unggul dalam memprediksi *churn* pelanggan dibandingkan *Decision Tree*, sehingga dapat digunakan untuk strategi pemasaran yang lebih efektif dalam industri perbankan.**

Keywords—Prediksi, Bank, Random Forest, Decision Tree, Churn, Retained

I. Latar Belakang

Dalam era digital yang semakin maju, sektor perbankan berada dalam fase perubahan, di mana dengan meningkatnya jumlah lembaga keuangan baru, mereka perlu menyesuaikan layanan mereka dengan strategi yang lebih futuristik. Laporan Perbankan Ritel Global tahun 2019 memperkirakan bahwa sekitar 66,8 persen pelanggan perbankan saat ini berencana untuk beralih ke lembaga keuangan baru yang menawarkan layanan serupa. Fenomena ini dapat memberikan dampak negatif bagi sektor perbankan. Hal ini mungkin disebabkan oleh kegagalan dalam layanan perbankan serta perubahan preferensi di kalangan pelanggan. Oleh karena itu, untuk mengurangi kerugian akibat kehilangan pelanggan, penting untuk menganalisis faktor-faktor yang dapat mempengaruhi keputusan untuk berpindah bank. Meskipun masalah internal dapat diidentifikasi, mungkin sulit untuk secara langsung memahami pola preferensi pelanggan dan merumuskan solusi yang tepat. Oleh karena itu, memprediksi tren ini sangat penting karena berkaitan erat dengan keberhasilan bank.

Kehilangan pelanggan, atau *churn*, dapat memberikan dampak yang signifikan terhadap pendapatan dan pertumbuhan bank. Oleh karena itu, melakukan prediksi terhadap *churn* pelanggan sangat penting, karena hal ini dapat membantu mencegah potensi kehilangan pelanggan sekaligus meningkatkan layanan bank, serta memberikan keuntungan bagi institusi tersebut.

II. Landasan Teori

A. Prediksi

Prediksi adalah proses memperkirakan nilai atau hasil dari suatu peristiwa berdasarkan pola yang ditemukan dalam data historis. Dalam konteks ilmu komputer dan machine learning, prediksi digunakan untuk memberikan gambaran tentang kemungkinan kejadian di masa depan dengan memanfaatkan algoritma dan model matematis. Proses prediksi membutuhkan data yang telah dianalisis dan diproses sebelumnya untuk menghasilkan model yang dapat belajar dari pola dalam data.

Dalam konteks *Bank Customer Churn Prediction*, proses prediksi bertujuan untuk mengklasifikasikan pelanggan bank ke dalam dua kategori: *churn* atau tidak *churn*. Dengan

menggunakan algoritma *Decision Tree* dan *Random Forest*, model mampu mempelajari pola dari data historis pelanggan untuk memberikan hasil prediksi yang akurat.

B. Algoritma *Random Forest*

Random Forest adalah algoritma pembelajaran mesin berbasis ensemble learning yang digunakan untuk tugas klasifikasi dan regresi. Algoritma ini bekerja dengan membangun sekumpulan pohon keputusan (Decision Trees) dan menggabungkan hasil prediksinya untuk meningkatkan akurasi dan stabilitas model. Untuk mengatasi overfitting yang menjadi kelemahan utama Decision Tree, Random Forest menggunakan dua teknik utama, yaitu **bagging (bootstrap aggregating)** dan pemilihan subset acak dari fitur. Dalam bagging, data pelatihan diambil secara acak dengan penggantian untuk membangun setiap pohon, sehingga setiap pohon dilatih pada subset data yang berbeda. Selain itu, pada setiap node, pemilihan atribut terbaik dilakukan hanya pada subset acak dari fitur, yang membantu mengurangi korelasi antar pohon dan meningkatkan keanekaragaman (diversity).

Proses kerja Random Forest dimulai dengan melakukan bootstrap sampling untuk setiap pohon, diikuti dengan pembangunan pohon keputusan menggunakan subset acak fitur pada setiap node. Setelah semua pohon selesai dibangun, hasil prediksi dari setiap pohon digabungkan. Untuk klasifikasi, hasil akhir ditentukan melalui voting mayoritas, sedangkan untuk regresi, digunakan rata-rata dari semua prediksi pohon. Secara matematis, untuk klasifikasi, prediksi akhir dihitung sebagai $\hat{y} = mode\{T_1(x), T_2(x),..., T_k(x)\}$, sedangkan untuk regresi, digunakan formula $\hat{y} = \frac{1}{k}\sum_{i=1}^{k} T_i(x)$, di mana $T_i(x)$ adalah prediksi pohon ke-i untuk data x.

Algoritma ini menggunakan metrik evaluasi seperti **Entropy** $(H(S) = -\sum_{i=1}^{n} p_i \log_2(p_i))$, **Information Gain**, dan **Gini Index** $(Gini(S) = 1 - \sum_{i=1}^{n} p_i^2)$ untuk menentukan pembagian terbaik pada setiap node pohon. Random Forest memiliki sejumlah keunggulan, termasuk kemampuan untuk mengurangi overfitting, stabilitas terhadap noise, dan fleksibilitas untuk menangani data kategorikal maupun numerik. Selain itu, algoritma ini dapat memberikan metrik pentingnya fitur (feature importance) yang membantu dalam analisis data. Namun, Random Forest juga memiliki kelemahan, seperti kompleksitas komputasi yang lebih tinggi dibandingkan Decision Tree dan sulitnya interpretasi karena banyaknya pohon yang terlibat.

Dengan langkah-langkah seperti bootstrap sampling, pembangunan pohon dengan subset fitur acak, dan agregasi hasil, Random Forest dapat diimplementasikan "from scratch" untuk menghasilkan model yang kuat dan stabil. Aplikasi Random Forest meliputi berbagai bidang, seperti deteksi penipuan, prediksi penyakit, analisis sentimen, dan prediksi harga, menjadikannya salah satu algoritma yang sangat populer dalam pembelajaran mesin.

C. Algoritma *Decision Tree*

Decision Tree adalah salah satu metode *machine learning* yang termasuk dalam kategori *supervised learning*, digunakan untuk menyelesaikan masalah klasifikasi dan regresi. *Decision Tree* bekerja dengan membangun model berbentuk pohon berdasarkan atribut-atribut input untuk memprediksi nilai target. Dalam struktur pohon ini, setiap node mewakili atribut tertentu, cabang menggambarkan keputusan berdasarkan atribut tersebut, dan daun *(leaf)* menunjukkan hasil akhir atau prediksi. Proses pembangunan *Decision Tree* melibatkan pemilihan atribut yang paling optimal untuk membagi data menjadi subset yang lebih homogen, dengan menggunakan metrik tertentu seperti *Entropy, Information Gain, Gini Index,* atau *Reduction in Variance.*

Entropy (H) digunakan untuk mengukur tingkat ketidakpastian dalam data, dengan rumus:

$$H(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

di mana p_i adalah probabilitas dari kelas i. Berdasarkan entropy, Information Gain (IG) dihitung untuk menentukan atribut terbaik dengan mengukur pengurangan ketidakpastian setelah pembagian:

$$IG(S, A) = H(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} H(S_v)$$

di mana S_v adalah subset data setelah pembagian berdasarkan atribut A. Selain itu, Gini Index sering digunakan untuk mengukur impurity atau ketidakmurnian dalam data dengan rumus:

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2$$

Sedangkan untuk kasus regresi, Reduction in Variance digunakan untuk mengukur pengurangan variansi dalam data numerik:

$$Reduction = Variance(parent) - \sum_{i=1}^{k} \frac{|S_i|}{|S|} Variance(S_i)$$

Proses pembentukan Decision Tree dimulai dari root node dengan data penuh, dilanjutkan dengan pemilihan atribut terbaik berdasarkan metrik di atas, pembagian data ke subset sesuai atribut tersebut, dan pembangunan cabang-cabang secara rekursif hingga mencapai kondisi

tertentu, seperti homogenitas data, tidak ada fitur yang tersisa, atau kedalaman maksimum yang telah ditentukan. Untuk mencegah overfitting, pohon keputusan sering kali menggunakan teknik **pruning**, baik secara pre-pruning, yaitu menghentikan pembangunan pohon lebih awal berdasarkan kriteria tertentu, maupun post-pruning, yaitu memangkas node yang dianggap tidak signifikan setelah pohon selesai dibangun.

Decision Tree memiliki kelebihan berupa interpretasi yang mudah, kemampuan menangani data kategorikal maupun numerik, serta tidak memerlukan praproses data yang kompleks. Namun, metode ini juga memiliki kelemahan, seperti rentan terhadap overfitting dan sensitivitas terhadap perubahan kecil dalam data. Dengan memahami konsep-konsep ini, Decision Tree dapat diimplementasikan "from scratch" melalui langkah-langkah perhitungan metrik evaluasi, pembagian data, pembangunan pohon secara rekursif, dan penerapan pruning. Metode ini memiliki aplikasi luas dalam berbagai bidang, termasuk klasifikasi (misalnya diagnosis medis, pengklasifikasian dokumen) dan regresi (misalnya prediksi harga atau nilai).

III. Desain Sistem dan Model

Untuk memastikan hasil prediksi *churn* yang dapat diandalkan, efektif, dan efisien, desain sistem proyek ini mencakup beberapa tahapan penting. Setiap tahap dirancang dengan mempertimbangkan kebutuhan data, algoritma, dan implementasi dalam konteks operasional perbankan.

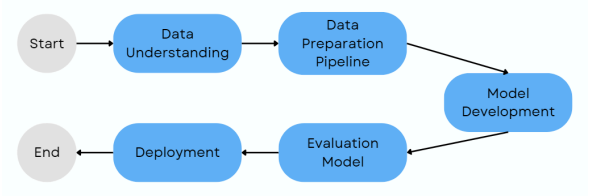
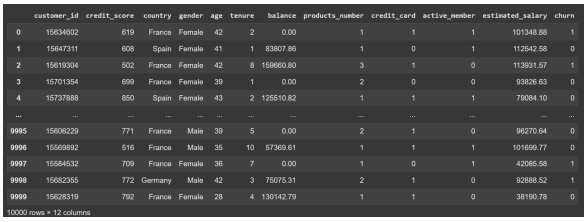


Fig. 1. Diagram Alur

1. *Data Understanding*

Proyek ini menggunakan *Bank Customer Churn Dataset* yang diperoleh dari website *Kaggle*, yang dapat diakses melalui URL: Bank Customer Churn Dataset. Dataset ini menyediakan berbagai informasi terkait profil nasabah, seperti kepemilikan kartu kredit, status keanggotaan, serta data demografis. Informasi ini bertujuan untuk mengidentifikasi pola perilaku nasabah, khususnya terkait keputusan untuk tetap menjadi pelanggan atau berhenti menggunakan layanan bank.

Variabel target dalam dataset berupa nilai biner, dengan nilai 1 menandakan nasabah telah meninggalkan bank, sedangkan nilai 0 menunjukkan nasabah masih aktif. Secara keseluruhan, dataset ini terdiri dari 12 fitur utama yang menggambarkan data pribadi dan riwayat interaksi nasabah dengan bank. Penjelasan rinci mengenai setiap fitur dapat dilihat pada Tabel 1.



	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

Fig. 2. Dataset

TABLE I. Deskripsi Dataset

Nama Fitur	Deskripsi Fitur
customer_id	ID unik untuk identifikasi nasabah bank
credit_score	Skor kredit pelanggan
country	Negara asal pelanggan
gender	Jenis kelamin pelanggan
age	Usia pelanggan
tenure	Jumlah tahun nasabah telah menjadi pemegang rekening bank
balance	Saldo rekening nasabah bank
products_number	Jumlah produk bank yang digunakan nasabah (rekening Tabungan, mobile banking, internet banking, dll)
credit_card	Tipe data bilangan biner untuk mengetahui apakah nasabah memiliki kartu kredit di bank atau tidak
active_member	Tipe data bilangan biner untuk mengetahui apakah nasabah merupakan anggota aktif bank atau bukan
estimated_salary	Perkiraan gaji nasabah dalam dolar
churn	Tipe data bilangan biner 1 jika nasabah telah meninggalkan bank selama suatu periode dan 0 jika nasabah dipertahankan

2. *Data Preparation Pipeline*

Sebelum implementasi model dilakukan, data diproses melalui tahap *Data Preparation Pipeline* untuk memastikan formatnya sesuai dengan algoritma *machine learning*. Dalam proyek ini, semua fitur dalam dataset dimanfaatkan untuk analisis dan prediksi *churn* pelanggan. Tahap ini dirancang secara menyeluruh untuk menjamin kualitas dan konsistensi data sebelum model dikembangkan.

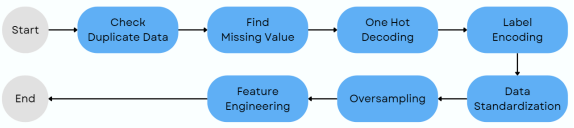


Fig. 3. Diagram Alur *Data Preparation Pipeline*

2.1. *Check Duplicate Data*

Langkah pertama adalah memeriksa apakah terdapat data duplikat dalam dataset. Data duplikat dapat

menyebabkan bias dalam model, karena informasi yang sama dihitung lebih dari sekali. Jika ada duplikat, baris tersebut dihapus untuk memastikan bahwa setiap data unik.

2.2. *Find Missing Value*

Langkah ini bertujuan untuk mengidentifikasi nilai yang hilang (*missing values*) dalam dataset. Nilai yang hilang dapat muncul akibat kesalahan dalam proses pengumpulan data. Setelah ditemukan, nilai tersebut dapat diatasi dengan teknik imputasi, seperti menggunakan median, rata-rata, atau modus, tergantung pada karakteristik data. Jika tidak terdapat nilai yang hilang, tahap pengisian nilai kosong dengan median tidak diperlukan.

2.3. *One Hot Encoding*

One hot encoding digunakan untuk mengkonversi variabel kategoris menjadi format numerik biner yang dapat diproses oleh algoritma *machine learning*. Misalnya, variabel "*country*" dengan kategori "France", "Germany", dan "Spain" akan dikonversi menjadi tiga kolom biner. (0 untuk France, 1 untuk Germany, dan 2 untuk Spain).

2.4. *Label Encoding*

Langkah ini bertujuan untuk mengonversi data kategorik menjadi angka menggunakan label encoding. Biasanya, langkah ini digunakan untuk variabel dengan urutan logis atau kategori biner, misalnya, "gender" (1 untuk laki-laki, 0 untuk perempuan).

2.5. *Data Standardization*

Data numerik distandarisasi agar memiliki skala yang sama. Teknik ini mengubah data ke distribusi dengan rata-rata 0 dan standar deviasi 1.

2.6. *Oversampling*

Teknik *oversampling* digunakan untuk mengatasi ketidakseimbangan kelas dalam dataset. Ketidakseimbangan kelas terjadi jika jumlah data dalam satu kelas (misalnya, *churn*) jauh lebih sedikit dibandingkan kelas lainnya (*retained*). *SMOTE (Synthetic Minority Over-sampling Technique)* adalah salah satu metode yang sering digunakan untuk menciptakan data sintetis pada kelas minoritas.

2.7. *Feature Engineering*

Langkah ini melibatkan pembuatan fitur baru atau transformasi fitur yang sudah ada untuk meningkatkan kinerja model. Contohnya termasuk menciptakan fitur interaksi antar variabel, normalisasi log untuk distribusi data yang mencolok, atau ekstraksi informasi temporal dari data waktu.

3. *Modeling* dan *Prediction*

3.1. *Decision Tree*

Pada langkah ini, Model *Decision Tree* dilatih menggunakan Gini Index sebagai kriteria untuk mengukur kualitas split. Hyperparameter seperti kedalaman maksimum dan jumlah minimum sampel per daun yang digunakan adalah Kedalaman Maksimum : 10 dan Minimum Sampel per Daun 15.

3.2. *Random Forest*

Pada model ini, Model *Random Forest* dilatih dengan membuat ansambel dari 4 decision tree. Model ini menggunakan Gini Index dan melakukan bootstrapping dengan 800 sampel untuk meningkatkan keragaman pada *decision tree*. Jumlah estimator yang digunakan adalah 4, begitu pula dengan kedalaman maksimumnya berjumlah 4, dan fitur maksimumnya adalah ditetapkannya integer 2.

3.3. *Prediction*

Pada tahap prediksi, sistem memanfaatkan model yang telah dilatih untuk mengklasifikasikan pelanggan ke dalam kategori *churn* atau *retained*. Proses dimulai dengan memasukkan data pelanggan melalui antarmuka pengguna, di mana data tersebut mencakup parameter seperti skor kredit, negara, jenis kelamin, usia, tenure, saldo rekening, jumlah produk yang digunakan, status kartu kredit, status keanggotaan aktif, dan estimasi gaji. Sistem menggunakan dua model, yaitu Random Forest dan Decision Tree, yang masing-masing memberikan prediksi berdasarkan pola yang telah dipelajari dari data yang sudah ada. *Random Forest* menghasilkan prediksi dengan menggabungkan keputusan dari beberapa pohon melalui metode voting mayoritas untuk meningkatkan akurasi, sedangkan *Decision Tree* membuat keputusan menggunakan satu pohon berdasarkan aturan pemisahan data. Hasil prediksi disajikan secara langsung kepada pengguna dalam bentuk kategori *churn* untuk pelanggan yang

diprediksi akan meninggalkan bank, dan *retained* untuk pelanggan yang diprediksi tetap menggunakan layanan bank. Sistem ini dirancang untuk memberikan hasil yang mudah dipahami dan mendukung pengambilan keputusan yang lebih baik oleh pihak bank.

4. *Model Evaluation*

Tahap evaluasi model bertujuan untuk memastikan bahwa model yang dibuat mampu memberikan hasil yang dapat diandalkan dalam memprediksi *churn* pelanggan. Metrik evaluasi seperti akurasi, precision, recall, dan F1-score digunakan untuk mengevaluasi kinerja model secara keseluruhan. Akurasi menunjukkan persentase prediksi yang benar, precision dan recall melengkapi evaluasi ini dengan mengukur seberapa akurat model dalam memprediksi *churn* yang benar-benar terjadi dan seberapa baik model mampu mendeteksi seluruh pelanggan yang *churn*, sementara F1-score memastikan keseimbangan antara precision dan recall untuk mengevaluasi performa model secara lebih komprehensif. Evaluasi ini juga mencakup analisis confusion matrix untuk memvisualisasikan distribusi prediksi yang benar dan salah, sehingga memudahkan identifikasi kelemahan model secara lebih rinci.

5. *Deployment*

Tahap deployment bertujuan untuk mengintegrasikan model yang telah dibangun ke dalam aplikasi yang dapat digunakan oleh pengguna secara langsung. Model ini diimplementasikan melalui aplikasi berbasis website menggunakan *Streamlit*, yang dirancang untuk menyediakan antarmuka yang sederhana dan mudah digunakan.

Aplikasi ini memungkinkan pengguna untuk memasukkan data dan melihat hasil prediksi *churn*. Selain itu, aplikasi ini juga memungkinkan aksesibilitas yang luas, karena dapat dijalankan langsung melalui browser tanpa memerlukan instalasi perangkat lunak tambahan.

IV. Implementasi dan Hasil Performa Model

1. Implementasi Algoritma

A. Random Forest

Implementasi algoritma Random Forest pada kode ini menggunakan pendekatan ensemble berbasis pohon keputusan. Proses dimulai dengan fungsi **bootstrapping**, yang membuat dataset bootstrap dari data pelatihan dengan mengambil sampel acak sejumlah tertentu. Setiap dataset bootstrap digunakan untuk membangun pohon keputusan individu melalui fungsi **decision_tree_algorithm**, yang mendukung pengambilan subset acak fitur (random subspace) untuk meningkatkan keragaman pohon. Parameter seperti jumlah maksimum kedalaman pohon (**dt_max_depth**) dan jumlah fitur yang dipilih (**n_features**) dapat dikustomisasi.

Hutan dibangun dengan fungsi **random_forest_algorithm**, yang menghasilkan sejumlah pohon keputusan (**n_trees**) dengan dataset bootstrap masing-masing. Setelah hutan terbentuk, prediksi dilakukan dengan fungsi **random_forest_predictions**, di mana setiap pohon dalam hutan memberikan prediksi terhadap data uji. Prediksi akhir diperoleh dengan metode voting mayoritas di antara hasil prediksi dari semua pohon.

Algoritma ini memperkuat model dengan memanfaatkan variasi dataset bootstrap dan subset fitur, sehingga mengurangi risiko overfitting dan meningkatkan akurasi prediksi pada data yang belum pernah dilihat. Selain itu, penggunaan entropi dalam pembentukan pohon memastikan pemisahan data yang optimal pada setiap simpul.

```python
import numpy as np
import pandas as pd
import random
# 1. Decision Tree helper functions

# 1.1 Data pure?
def check_purity(data):

    label_column = data[:, -1]
    unique_classes = np.unique(label_column)
    if len(unique_classes) == 1:
        return True
    else:
        return False

# 1.2 Classify
def classify_data(data):
    label_column = data[:, -1]
    unique_classes, counts_unique_classes = np.unique(label_column, return_counts=True)
    index = counts_unique_classes.argmax()
    classification = unique_classes[index]
    return classification

# 1.3 Potential splits?
def get_potential_splits(data, random_subspace):
    potential_splits = {}
    _, n_columns = data.shape
    column_indices = list(range(n_columns - 1))    # excluding the last column which is the label
    if random_subspace and random_subspace <= len(column_indices):
        column_indices = random.sample(population=column_indices, k=random_subspace)
    for column_index in column_indices:
        values = data[:, column_index]
        unique_values = np.unique(values)
        potential_splits[column_index] = unique_values
    return potential_splits

# 1.4 Lowest Overall Entropy?
def calculate_entropy(data):
    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)
    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))
    return entropy

def calculate_overall_entropy(data_below, data_above):
    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n
    overall_entropy =  (p_data_below * calculate_entropy(data_below)
                      + p_data_above * calculate_entropy(data_above))
    return overall_entropy
```

```python
def determine_best_split(data, potential_splits):
    overall_entropy = 9999
    for column_index in potential_splits:
        for value in potential_splits[column_index]:
            data_below, data_above = split_data(data, split_column=column_index, split_value=value)
            current_overall_entropy = calculate_overall_entropy(data_below, data_above)
            if current_overall_entropy <= overall_entropy:
                overall_entropy = current_overall_entropy
                best_split_column = column_index
                best_split_value = value
    return best_split_column, best_split_value


# 1.5 Split data
def split_data(data, split_column, split_value):
    split_column_values = data[:, split_column]
    type_of_feature = FEATURE_TYPES[split_column]
    if type_of_feature == "continuous":
        data_below = data[split_column_values <= split_value]
        data_above = data[split_column_values >  split_value]
    else:
        data_below = data[split_column_values == split_value]
        data_above = data[split_column_values != split_value]
    return data_below, data_above
```

```python
# 2. Decision Tree Algorithm
def decision_tree_algorithm(df, counter=0, min_samples=2, max_depth=5, random_subspace=None):
    if counter == 0:
        global COLUMN_HEADERS, FEATURE_TYPES
        COLUMN_HEADERS = df.columns
        FEATURE_TYPES = determine_type_of_feature(df)
        data = df.values
    else:
        data = df
    if (check_purity(data)) or (len(data) < min_samples) or (counter == max_depth):
        classification = classify_data(data)

        return classification
    else:
        counter += 1
        potential_splits = get_potential_splits(data, random_subspace)
        split_column, split_value = determine_best_split(data, potential_splits)
        data_below, data_above = split_data(data, split_column, split_value)
        if len(data_below) == 0 or len(data_above) == 0:
            classification = classify_data(data)
            return classification
        feature_name = COLUMN_HEADERS[split_column]
        type_of_feature = FEATURE_TYPES[split_column]
        if type_of_feature == "continuous":
            question = "{} <= {}".format(feature_name, split_value)
        else:
            question = "{} = {}".format(feature_name, split_value)
        sub_tree = {question: []}
        yes_answer = decision_tree_algorithm(data_below, counter, min_samples, max_depth, random_subspace)
        no_answer = decision_tree_algorithm(data_above, counter, min_samples, max_depth, random_subspace)
        if yes_answer == no_answer:
            sub_tree = yes_answer
        else:
            sub_tree[question].append(yes_answer)
            sub_tree[question].append(no_answer)
        return sub_tree
```

```python
# 3. Make predictions
# 3.1 One example
def predict_example(example, tree):
    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split(" ")
    if comparison_operator == "<=":
        if example[feature_name] <= float(value):
            answer = tree[question][0]
        else:
            answer = tree[question][1]
    else:
        if str(example[feature_name]) == value:
            answer = tree[question][0]
        else:
            answer = tree[question][1]
    if not isinstance(answer, dict):
        return answer
    else:
        residual_tree = answer
        return predict_example(example, residual_tree)
def decision_tree_predictions(test_df, tree):
    predictions = test_df.apply(predict_example, args=(tree,), axis=1)
    return predictions
```

```python
def bootstrapping(train_df, n_bootstrap):
    bootstrap_indices = np.random.randint(low=0, high=len(train_df), size=n_bootstrap)
    df_bootstrapped = train_df.iloc[bootstrap_indices]
    return df_bootstrapped

def random_forest_algorithm(train_df, n_trees, n_bootstrap, n_features, dt_max_depth):
    forest = []
    for i in range(n_trees):
        df_bootstrapped = bootstrapping(train_df, n_bootstrap)
        tree = decision_tree_algorithm(df_bootstrapped, max_depth=dt_max_depth, random_subspace=n_features)
        forest.append(tree)
    return forest

def random_forest_predictions(test_df, forest):
    df_predictions = {}
    for i in range(len(forest)):
        column_name = "tree_{}".format(i)
        predictions = decision_tree_predictions(test_df, tree=forest[i])
        df_predictions[column_name] = predictions

    df_predictions = pd.DataFrame(df_predictions)
    random_forest_predictions = df_predictions.mode(axis=1)[0]
    return random_forest_predictions
```

Fig. 4. Code Algoritma Random Forest

B. Decision Tree

Implementasi algoritma Decision Tree pada kode ini mencakup pembuatan model pohon keputusan berbasis rekursif. Data diproses untuk menentukan tipe fitur (kontinu atau kategorikal) melalui fungsi **type_of_cols**, sementara semua kemungkinan split untuk setiap fitur dihitung menggunakan **get_splits**. Pembagian data dilakukan oleh **split_data** berdasarkan nilai split.

Kualitas split diukur menggunakan entropi melalui fungsi **entropy** dan **entropy_data**. Fitur dan split terbaik ditentukan oleh fungsi **best_split**. Pohon dibangun secara rekursif melalui **_tree_builder**, yang membagi data hingga mencapai kondisi berhenti, seperti kedalaman maksimum atau jumlah sampel minimum. Pohon hasil akhir disimpan dalam atribut **tree**.

Prediksi dilakukan dengan menavigasi pohon menggunakan **_predict**, yang memeriksa kondisi pada setiap simpul hingga mencapai simpul daun. Fungsi **predict** menghasilkan prediksi untuk seluruh data uji. Implementasi ini menangani data kontinu dan kategorik serta menggunakan entropi sebagai kriteria pemisahan.

```python
from collections import Counter
import numpy as np

class decision_Tree:

    def __init__(self):
        self.no_of_unique=6
        self.counter=0
        self.min_samples=15
        self.max_depth=10
    def type_of_cols(self,data):
        col_type=[]
        for col in data.columns[:-1]:
            no_of_col_unique= data[col].nunique()
            if data[col].dtypes==object or no_of_col_unique < self.no_of_unique:
                col_type.append("categorical")
            else:
                col_type.append("continuous")
        return col_type
    def get_splits(self,data):
        splits = {}
        feature_type=self.type_of_cols(data)
        for column_index in range(data.shape[1]-1):
            values = data.iloc[:, column_index]
            unique_values = np.unique(values)
            type_of_feature = feature_type[column_index]
            if type_of_feature == "continuous":
                splits[column_index] = []
                for index in range(len(unique_values)):
                    if index != 0:
                        current_value = unique_values[index]
                        previous_value = unique_values[index - 1]
                        split = (current_value + previous_value) / 2
                        splits[column_index].append(split)
            elif len(unique_values) > 1:
                splits[column_index] = unique_values
        return splits
    def split_data(self,data, column, value):
        feature_type=self.type_of_cols(data)
        split_values = data.iloc[:,column]
        type_of_feature = feature_type[column]
        if type_of_feature == "continuous":
            left = data[split_values <= value]
            right = data[split_values >  value]
        else:
            left = data[split_values == value]
            right = data[split_values != value]
    def entropy(self,data):
        prob=list(dict(data.iloc[:, -1].value_counts(normalize=True)).values())
        entropy = sum(prob* -np.log2(prob))
        return entropy
```

```python
    def entropy_data(self,left,right):
        n = len(left) + len(right)
        p_left = len(left) / n
        p_right = len(right) / n
        entropy_ =  (p_left * self.entropy(left)+ p_right *self.entropy(right))
        return entropy_
    def best_split(self,data,splits):
        entropy = 99999
        for col in splits:
            for val in splits[col]:
                left, right = self.split_data(data, column=col, value=val)
                current_entropy = self.entropy_data(left,right)
                if current_entropy <= entropy:
                    entropy = current_entropy
                    best_column = col
                    best_split = val
        return best_column, best_split
    def _tree_builder(self,df):
        column=df.columns
        feature_type=self.type_of_cols(df)
        data = df
        if  (df.iloc[:,-1].nunique()==1) or (len(data) < self.min_samples) or (self.counter == self.max_depth):
            classes= Counter(df.iloc[:,-1]).most_common(1)[0][0]
            return classes
        else:
            self.counter += 1
            splits = self.get_splits(data)
            split_column, split_value = self.best_split(data,splits)
            left, right = self.split_data(data, split_column, split_value)
            feature_name = column[split_column]
            type_of_feature = feature_type[split_column]
            if type_of_feature == "continuous":
                question = "{} <= {}".format(feature_name, split_value)
            else:
                question = "{} = {}".format(feature_name, split_value)
            mytree = {question: []}
            ans_yes = self._tree_builder(left)
            ans_no = self._tree_builder(right)
            if ans_yes == ans_no:
                mytree = ans_yes
            else:
                mytree[question].append(ans_yes)
                mytree[question].append(ans_no)
            return mytree
    def fit(self,x,y):
        x["output"]=y
        self.tree=self._tree_builder(x)
        return self.tree
```

```
def _predict(sf,dx,tree):
    root_node = list(tree.keys())[0]
    column,operator,split=root_node.split(" ")
    if operator == "<=":
        if dx[column] <= float(split):
            result = tree[root_node][0]
        else:
            result = tree[root_node][1]
    else:
        if str(dx[column]) == split:
            result = tree[root_node][0]
        else:
            result = tree[root_node][1]
    if type(result)!=dict:
        return result
    else:
        return sf._predict(dx,result)
def predict(self,X_test):
    s=[]
    for i in range(X_test.shape[0]):
        s.append(self._predict(X_test.iloc[i],self.tree))
    return s
```

Fig. 5. Code Algoritma Decision Tree

2. Hasil Performa Model

A. Random Forest

Hasil performa model Random Forest menunjukkan evaluasi berdasarkan confusion matrix dan metrik utama seperti precision, recall, f1-score, serta akurasi keseluruhan. Confusion matrix menunjukkan bahwa model memprediksi **1.4K sampel kelas 0** dengan benar (True Negatives) dan **910 sampel kelas 1** dengan benar (True Positives). Namun, terdapat **160 kesalahan prediksi kelas 0 menjadi kelas 1** (False Positives) dan **670 kesalahan prediksi kelas 1 menjadi kelas 0**(False Negatives).

Untuk kelas 0, model memiliki precision sebesar **0.73**, recall **0.87**, dan f1-score **0.79**, sedangkan untuk kelas 1, precision mencapai **0.83**, recall **0.68**, dan f1-score **0.75**. Secara keseluruhan, model memiliki akurasi **77%**, dengan rata-rata presisi dan recall masing-masing sebesar **0.78** dan **0.77**.

Hasil ini menunjukkan performa model yang baik dalam mendeteksi **kelas 0**, meskipun performa pada **kelas 1** sedikit lebih rendah.

	precision	recall	f1-score	support
0	0.73	0.87	0.79	1607
1	0.83	0.68	0.75	1578
accuracy			0.77	3185
macro avg	0.78	0.77	0.77	3185
weighted avg	0.78	0.77	0.77	3185

Fig. 6. Classification Report

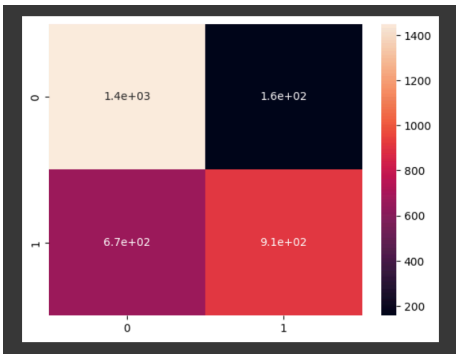


Fig. 7. Confusion Matrix

B. Decision Tree

Hasil performa model menunjukkan evaluasi berdasarkan confusion matrix dan metrik utama seperti precision, recall, f1-score, serta akurasi keseluruhan. Confusion matrix menunjukkan bahwa model memprediksi **2.7K sampel kelas 0** dengan benar (True Negatives) dan **0 sampel kelas 1** dengan benar (True Positives). Namun, terdapat **0 kesalahan prediksi kelas 0 menjadi kelas 1** (False Positives) dan **640 kesalahan prediksi kelas 1 menjadi kelas 0** (False Negatives).

Untuk kelas 0, model memiliki precision sebesar **0.70**, recall **0.72**, dan f1-score **0.71**, sedangkan untuk kelas 1, precision **0.70**, recall **0.68**, dan f1-score bernilai **0.69**. Secara keseluruhan, model memiliki akurasi **70%**, dengan rata-rata presisi dan recall masing-masing sebesar **0.70** dan **0.70**.

Hasil ini menunjukkan bahwa model cukup baik dalam mendeteksi **kelas 0** dan **kelas 1**, sehingga performa keseluruhan model cukup seimbang dalam mendeteksi kedua kelas.

	precision	recall	f1-score	support
0	0.70	0.72	0.71	1633
1	0.70	0.68	0.69	1553
accuracy			0.70	3186
macro avg	0.70	0.70	0.70	3186
weighted avg	0.70	0.70	0.70	3186

Fig. 8. Classification Report

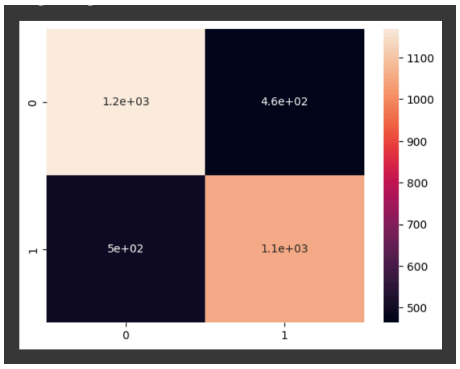


Fig. 9. Confusion Matrix

V. Deployment

Tampilan awal aplikasi *Customer Churn Prediction* ditunjukkan pada Gambar 4 di bawah ini. Aplikasi ini dirancang untuk memudahkan pengguna dalam memasukkan data dan melihat hasil prediksi *churn*. Aplikasi dapat diakses melalui URL: Customer Churn Prediction App.

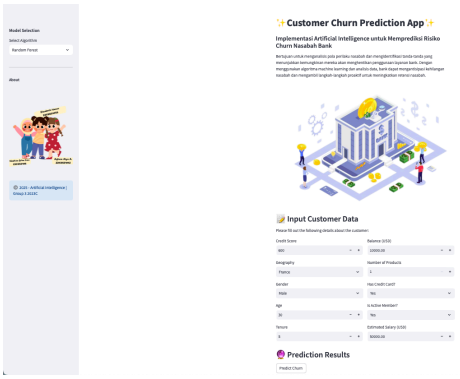


Fig. 10. Tampilan Awal Aplikasi

VI. Kesimpulan

Berdasarkan hasil evaluasi, algoritma *Random Forest* menunjukkan performa terbaik dalam memprediksi *churn* pelanggan bank dengan tingkat akurasi sebesar 78%, *F1-Score* 0.77, *Precision* 0.78, dan *Recall* 0.77. Algoritma *Decision Tree* menunjukkan performa paling rendah dengan akurasi sebesar 69%, *F1-Score* 0.69, *Precision* 0.69, dan *Recall* 0.69, menunjukkan bahwa algoritma ini kurang optimal untuk dataset ini dibandingkan algoritma *Random Forest.*

Acknowledgment

Penyusunan laporan dan penyelesaian proyek ini tidak lepas dari dukungan dan masukan berbagai pihak. Dengan tulus, kami mengucapkan terima kasih kepada Ibu Dr. Elly Matul Imah, M.Kom. dan Ibu Dinda Galuh Guminta, S.Stat., M.Stat. selaku dosen mata kuliah *Artificial Intelligence*, atas bimbingan dan arahan yang diberikan selama proses pengerjaan proyek ini. Tidak lupa, kami juga mengucapkan terima kasih kepada rekan-rekan yang telah memberikan masukan dan dukungan moral, sehingga proyek ini dapat diselesaikan dengan baik.

Referensi

[1] He Zhu, “Bank Customer Churn Prediction with Machine Learning Methods,” *Proceedings of the 2nd International Conference on Financial Technology and Business Analysis*, pp. 23-29, DOI: 10.54254/2754-1169/69/20230773, 2023.

[2] Shiyunyang Zhao, “Customer Churn Prediction Based on the Decision Tree and Random Forest Model,” *BCP Business & Management*, vol. 44, pp. 339-344, 2023.

[3] Pahul Preet Singh, Fahim Islam Anik, Rahul Senapati, Arnav Sinha, Nazmus Sakib, dan Eklas Hossain, “Investigating customer churn in banking: a machine learning approach and visualization app for data science and management,” *Data Science and Management*, vol. 7, pp. 7–16, 2024.

[4] W. Zhang, "Bank Customer Churn Analysis and Prediction," in *Proc. MSIEID 2022*, Chongqing, People's Republic of China, Dec. 9–11, 2022. Copyright © 2023 EAI. DOI: 10.4108/eai.9-12-2022.2327608.

Lampiran

1. Kontribusi Anggota

TABLE II. Kontribusi Anggota

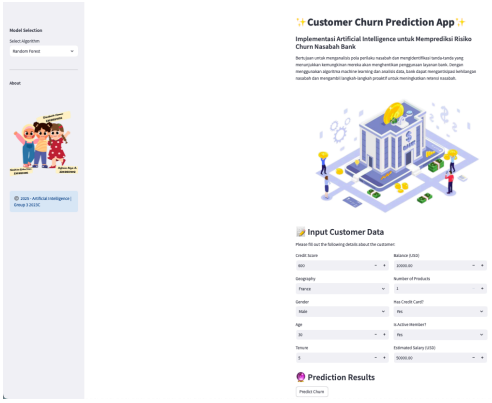
Nama	NIM	Kontribusi
Elizabeth Hanov	23031554055	Mencari dataset, membuat kode modeling, membuat kode streamlit, membuat laporan, dan mencari referensi jurnal.
Aghnia Alya Amarilla	23031554102	Mencari dataset, membuat kode modeling, membuat kode streamlit, membuat laporan, dan mencari referensi jurnal.
Nashita Erha Fitri	23031554116	Mencari dataset, membuat kode modeling, membuat kode streamlit, membuat laporan, dan mencari referensi jurnal.

2. Listing Code

Kode yang digunakan dalam proyek ini dapat diakses melalui URL: Project AI_Group 3_2023C. Folder tersebut berisi dataset dan seluruh tahapan mulai dari preprocessing data, pembangunan model, evaluasi, hingga persiapan deployment.

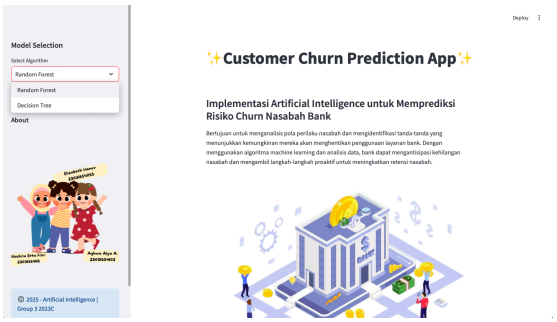
3. Tangkapan Layar UI setelah Deployment

a. Tampilan Awal



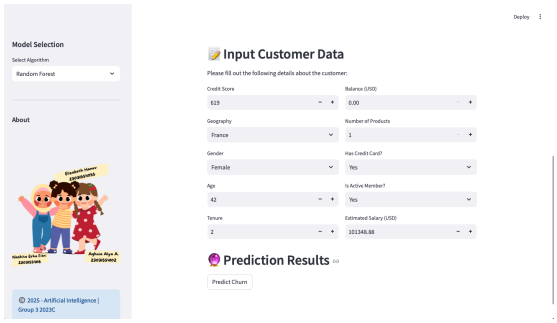
Ketika user sudah berhasil membuka link, maka tampilan awal akan seperti gambar diatas. Untuk mengetahui hasil prediksi, user akan diminta untuk memasukkan *Credit Score, Geography, Gender, Age, Tenure, Balance, Number of Products, Has Credit Card?, Is Active Member?, dan Estimated Salary.* Kemudian User juga diharuskan untuk memilih algoritma apa yang ingin dipakai untuk melakukan prediksi.

b. Tampilan Memilih Algoritma



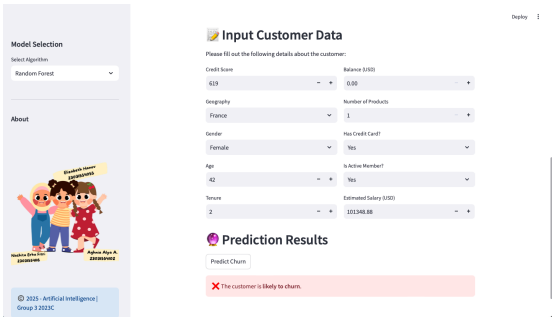
Pada dropdown list tersebut, user dapat memilih algoritma yang akan digunakan untuk melakukan prediksi. Dua opsi algoritma yang tersedia dalam daftar dropdown adalah Random Forest dan Decision Tree.

c. Tampilan ketika User sudah Input Data dan Menggunakan Algoritma *Random Forest.*



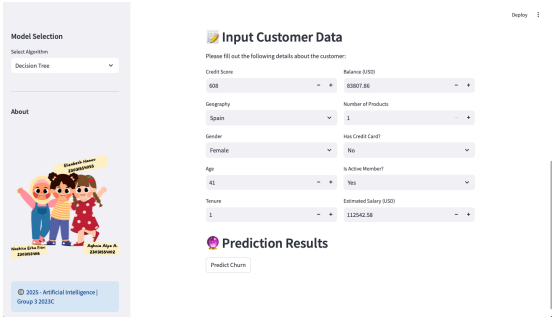
Pada tampilan ini user telah memasukkan data yang dibutuhkan untuk melakukan prediksi serta memilih *Random Forest* sebagai algoritma yang akan digunakan.

d. Tampilan Saat sudah Berhasil Melakukan Prediksi Menggunakan Algoritma *Random Forest*



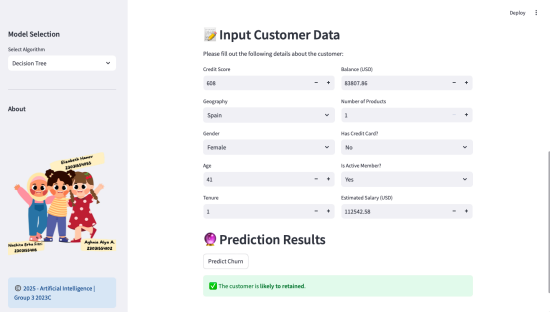
Dapat dilihat bahwa user telah berhasil melakukan prediksi dan hasilnya adalah customer akan *churn.* Dengan *credit score* sebesar 619 dimana user berasal dari negara *France* berjenis kelamin perempuan, berumur 42 tahun dengan catatan 2 tahun menjadi pemegang rekening bank yang memiliki *balance* senilai $0.00 USD dan 1 produk bank yang digunakan juga memiliki kartu kredit sebagai anggota aktif bank dengan perkiraan gaji sebesar $101348.99 USD.

e. Tampilan ketika User sudah Input Data dan Menggunakan Algoritma *Decision Tree*



Pada tampilan ini dilakukan uji coba untuk melakukan prediksi serta memilih *Random Forest* sebagai algoritma yang akan digunakan dengan memasukkan data yang dibutuhkan.

f. Tampilan Saat sudah Berhasil Melakukan Prediksi Menggunakan Algoritma *Decision Tree*



Dari hasil prediksi menggunakan algoritma *Decision Tree* dengan data yang diinput; *credit score* senilai 608, berasal dari negara *Spain*, berjenis kelamin perempuan, berumur 41 tahun dengan catatan sudah 1 tahun menjadi pemegang rekening bank yang memiliki *balance* senilai 83870,86, customer tersebut tidak *churn* atau dengan kata lain *retained.*