

LAPORAN CLO-3 ASSESSMENT—TEAM-BASED PROJECT

MATA KULIAH PENGANTAR KECERDASAN BUATAN



IF-45-05

Kelompok FineFine

Fathan Zhafiri Arshimny	1301213300	Membuat Powerpoint, merevisi laporan
Hanrocky Halim	1301213446	Membuat model ML
Steven Christian	1301210417	Melakukan analisis

Tim Kami mengerjakan tugas ini dengan cara yang tidak melanggar aturan perkuliahan dan kode etik akademisi. Jika melakukan plagiarisme atau jenis pelanggaran lainnya, maka Tim kami bersedia diberi nilai E untuk Mata Kuliah ini.

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM

BANDUNG

2023

I. PENDAHULUAN

A. Ikhtisar

Aritmia adalah gangguan irama detak jantung, baik terlalu cepat, terlalu lambat, ataupun tidak teratur. Aritmia dapat terjadi karena beberapa kondisi, seperti: hipertensi, diabetes, gangguan elektrolit, kelainan katup jantung, penyakit jantung bawaan, dan berbagai kondisi lainnya. Gaya hidup yang tidak sehat juga dapat memicu terjadinya aritmia, seperti kurang tidur, merokok, dan mengonsumsi minuman beralkohol [1].

Ada beberapa jenis aritmia, di antaranya adalah atrial fibrilasi, AV blok, supraventrikular takikardi, dan ventrikel fibrilasi. Aritmia bisa terjadi tanpa gejala. Hal ini bisa menyebabkan penderitanya tidak menyadari bahwa mereka menderita aritmia. Beberapa gejala aritmia antara lain: jantung berdetak lebih cepat, lebih lambat, pusing, pingsan, sesak napas, serta nyeri pada dada [1].

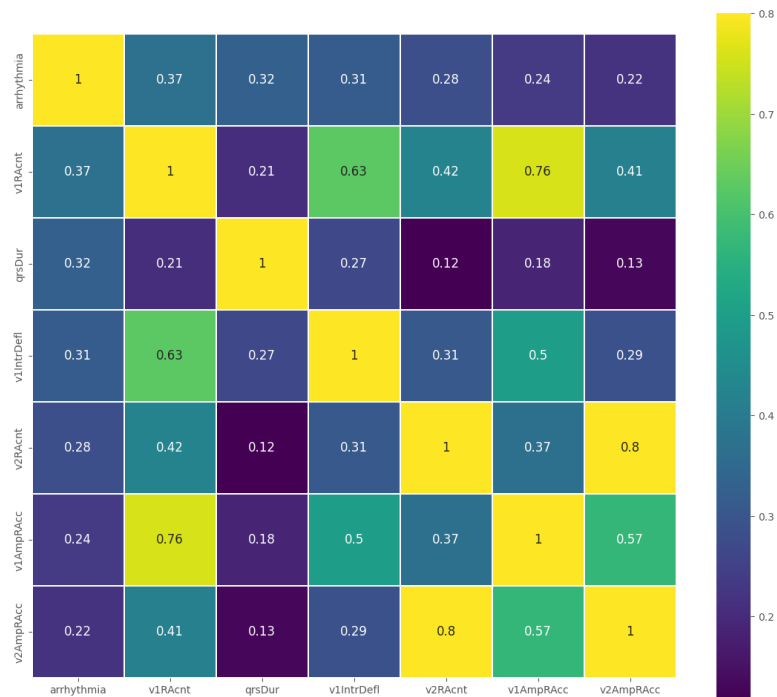
Banyaknya jenis aritmia menimbulkan pertanyaan apakah bisa seseorang membuat sebuah program komputer yang dapat mengklasifikasikan berbagai jenis aritmia. Program tersebut dapat digunakan sebagai salah satu asisten bagi tenaga kesehatan untuk melakukan diagnosis kepada pasien. Oleh karena itu, diperlukan sebuah algoritma pembelajaran mesin yang dapat digunakan untuk menganalisis jenis aritmia berdasarkan ciri-ciri yang dimilikinya.

Kami menggunakan himpunan data yang disediakan pada tautan [UCI Machine Learning Repository: Arrhythmia Data Set](#) dalam melakukan pembuatan program pengklasifikasian jenis aritmia ini. Data tersebut berasal dari sebuah studi yang dilakukan oleh H. Altay Guvenir untuk menentukan jenis aritmia berdasarkan data rekaman elektrokardiogram. Pada himpunan data tersebut, terdapat 279 atribut data yang terdiri dari informasi umum pasien, seperti umur, jenis kelamin, tinggi badan, dan berat badan, serta data rekaman elektrokardiogram yang berisi nilai-nilai rekaman ritme jantung, seperti interval QRS, interval PR, interval QT, jumlah detak jantung per menit, dan lain sebagainya.

B. Hasil Pengolahan Himpunan Data

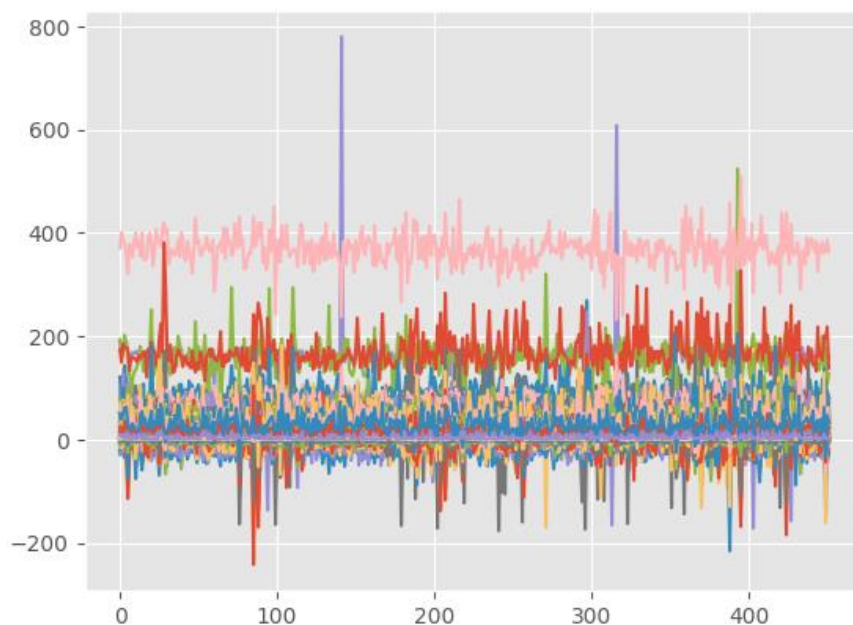
Berdasarkan hasil penelaahan yang kami dapatkan melalui penganalisisan data secara eksploratif, kami mendapatkan bahwa himpunan data tersebut terdiri dari 452 baris data yang masing-masingnya memiliki 279 atribut data. Terdapat pula atribut bertipe data nonnumerik yang setelah kami selidiki Terdapat beberapa data dengan *missing values* yang kami lakukan penggantian (*replacement*) dengan nilai yang paling banyak muncul (*most frequent value*) dari kolom tersebut.

Kami juga mencari korelasi antara setiap atribut data dengan kolom yang akan diprediksi, yaitu kolom 'arrhythmia.' Kami menemukan bahwa tujuh atribut dengan korelasi terkuat tidak memiliki korelasi di atas 37%.



Gambar 1. Tujuh Atribut dengan Korelasi Tertinggi

Selain itu, melalui pembuatan plot, kami mendapat wawasan mengenai persebaran data dari masing-masing atribut.



Gambar 2. Persebaran Data

Kami menyimpulkan bahwa tidak ada satu atribut yang berkorelasi tinggi dan terdapat beberapa kolom yang memiliki *missing values*. Kolom-kolom tersebut terutama di kolom ke-10, 11, 12, 13, dan 14 sehingga kolom tersebut dapat di-*drop*.

II. DASAR TEORI

A. Algoritma yang Dipilih

Algoritma pertama yang kami pilih adalah decision tree algorithm. Algoritma ini bekerja dengan memecah himpunan data menjadi subhimpunan yang lebih kecil berdasarkan kesamaan/relevansi atribut. Algoritma ini akan mengetes atribut-atribut data di setiap simpul keputusan (decision nodes), memecah himpunan data melalui cabang-cabang setiap kemungkinan-yang-ada. Jalur yang dilalui oleh elemen X tersebut akan membawa elemen tersebut dari akar ke simpul daun dari pohon keputusan [2]. Simpul daun yang dicapai adalah prediksi label/kelas/hasil klasifikasi untuk elemen X tersebut [3].

Algoritma yang kami buat melakukan penentuan label dengan menghitung nilai Gini dari data yang ada. Nilai Gini adalah ukuran ketidakmurnian suatu data. Semakin kecil nilai Gini, semakin kecil pula ketidakmurniannya; semakin murni datanya. Nilai Gini dihitung dengan rumus sebagai berikut.

$$Gini = 1 - \sum_{i=1}^j P(i)^2$$

Dengan $P(i)$ merepresentasikan perbandingan antara semua klasifikasi yang sesuai (*pass*) dengan total klasifikasi yang dilakukan. Saat nilai Gini sudah didapatkan, decision tree algorithm dapat menentukan titik pemisahan (*splitting point*) paling optimal untuk memaksimalkan kinerja klasifikasi.

Algoritma kedua yang kami pilih adalah k-nearest neighbors algorithm. Algoritma ini bekerja dengan menyimpan data latih sehingga data baru yang belum diberi label dapat dicarikan labelnya dengan membandingkan data tersebut dengan data latih yang paling mirip, yaitu yang memiliki pola paling dekat dengan data latih [2]. Data latih disimpan dalam ruang pola n dimensi, dengan n adalah banyaknya jumlah atribut yang dimiliki data [3]. Berikut adalah implementasi dari k-nearest neighbor algorithm.

Algoritma yang kami buat melakukan perhitungan kedekatan pola data terhadap data latih dengan mengurutkan jaraknya. Jarak tersebut kami hitung dengan perhitungan jarak Euclidean dan jarak Manhattan sebagai berikut.

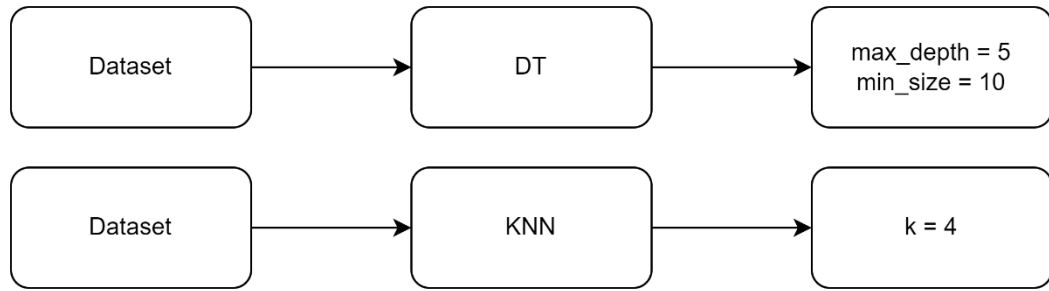
$$Euclidean Distance = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

$$Manhattan Distance = \sum_{i=1}^n |x_i - y_i|$$

Jarak Euclidean digunakan untuk menghitung jarak terpendek antara dua titik, yaitu jarak garis lurus antara kedua titik. Sementara itu, jarak Manhattan digunakan untuk menghitung perbedaan absolut di antara koordinat kedua titik.

B. Metode Kerja

Kami mengerjakan proyek ini dengan membuat algoritma dan menggunakan *library*. Hal ini kami lakukan untuk membandingkan hasil kerja kami yang menggunakan algoritma buatan kami dengan *library* yang sudah disediakan pada beberapa modul Python. Untuk algoritma yang kami buat sendiri, berikut adalah arsitektur model dari algoritma yang kami pilih.



Gambar 1. Arsitektur Model Decision Tree dan K-Nearest Neighbor

Himpunan data akan kami olah menggunakan decision tree algorithm dan k-nearest neighbors algorithm dengan menggunakan parameter-parameter yang digambarkan di atas. Untuk decision tree algorithm, kami menggunakan parameter kedalaman pohon maksimal yang bernilai lima (5) dan ukuran minimal pemecahal (*splitting*) data bernilai sepuluh (10). Sementara itu, untuk k-nearest neighbors algorithm, kami menggunakan parameter banyaknya tetangga yang dicek bernilai empat (4).

Dengan menggunakan metode yang telah dijelaskan di atas, hasil/keluaran dari algoritma yang kami pilih adalah nilai keakuratan prediksinya. Hasil akurasi prediksi yang semakin besar berarti algoritma yang kami buat sesuai dan cocok dengan data yang disajikan. Akan tetapi, kami jugaantisipasi dengan kemungkinan *overfitting* data sehingga kami hati-hati dalam memilih parameter-parameter yang kami gunakan untuk setiap algoritma.

Berikut adalah penjelasan dari algoritma decision tree algorithm yang kami buat.

```
[28] # Decision Tree Algorithm
def decision_tree(train, test, max_depth, min_size):
    tree = build_tree(train, max_depth, min_size)
    predictions = []
    for row in test:
        prediction = predict(tree, row)
        predictions.append(prediction)
    return predictions
```

Gambar 2. Potongan Kode Decision Tree Algorithm

Pada potongan kode di atas, sebuah pohon baru dibuat. Data latih yang ada kemudian dihitung indeks Gininya untuk menentukan bentuk pohon yang dibuat. Kemudian, dilakukan prediksi terhadap data yang ada. Data tersebut kemudian diklasifikasi dengan menggunakan pohon yang sudah dibuat. Untuk lebih lengkapnya, dapat dilihat pada tautan Google Colab yang dilampirkan di subbab C di bawah.

Berikut adalah penjelasan dari algoritma k-nearest neighbor algorithm yang kami buat.

```
[41] # Evaluasi model dengan menggunakan data fold
def evaluate_euclidean(fold, k):
    test, train = fold # what data we should use here
    X_train, y_train = train.drop(['arrhythmia'], axis=1), train['arrhythmia'] # what column we want to drop
    X_test, y_test = test.drop(['arrhythmia'], axis=1), test['arrhythmia'] # what column we want to drop
    X_train = norm(X_train) # normalize
    X_test = norm(X_test) # normalize
    y_preds = []
    for row in range(X_test.shape[0]):
        y_preds.append(knn_euclidean(X_train, y_train, X_test.iloc[row], k)) # the data, the data label, X_test.iloc[row], assign to choose how many point that are close

    return (acc(y_preds, y_test))

# Evaluasi model dengan menggunakan data fold
def evaluate_manhattan(fold, k):
    test, train = fold # what data we should use here
    X_train, y_train = train.drop(['arrhythmia'], axis=1), train['arrhythmia'] # what column we want to drop
    X_test, y_test = test.drop(['arrhythmia'], axis=1), test['arrhythmia'] # what column we want to drop
    X_train = norm(X_train) # normalize
    X_test = norm(X_test) # normalize
    y_preds = []
    for row in range(X_test.shape[0]):
        y_preds.append(knn_manhattan(X_train, y_train, X_test.iloc[row], k)) # the data, the data label, X_test.iloc[row], assign to choose how many point that are close

    return (acc(y_preds, y_test))
```

Gambar 3. Potongan Kode K-Nearest Neighbor Algorithm

Pada potongan kode di atas, dilakukan pemisahan (*splitting*) terhadap data, lalu data tersebut dinormalisasi. Kemudian dilakukan prediksi; prediksi dilakukan dengan dua metode, yaitu dengan menggunakan jarak Euclidean dan menggunakan jarak Manhattan. Keluarannya adalah akurasi dari prediksi model tersebut.

Untuk pengerjaan dengan menggunakan *library*, berikut adalah penjelasan dari *library* yang digunakan dan penggunaannya untuk decision tree algorithm.

```
▶ # Splitting the Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
selector = SelectKBest(score_func=f_classif, k=50) # Select top 10 features
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

optuna.logging.set_verbosity(optuna.logging.WARNING)

def objective(trial):
    # Define the search space for hyperparameters
    max_depth = trial.suggest_int("max_depth", 1, 10)
    criterion = trial.suggest_categorical("criterion", ["gini", "entropy"])

    # Initialize and train the Decision Tree classifier
    dt = DecisionTreeClassifier(max_depth=max_depth, criterion=criterion)
    dt.fit(X_train_selected, y_train)

    # Make predictions on the test set
    y_pred = dt.predict(X_test_selected)

    # Calculate accuracy as the evaluation metric
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy

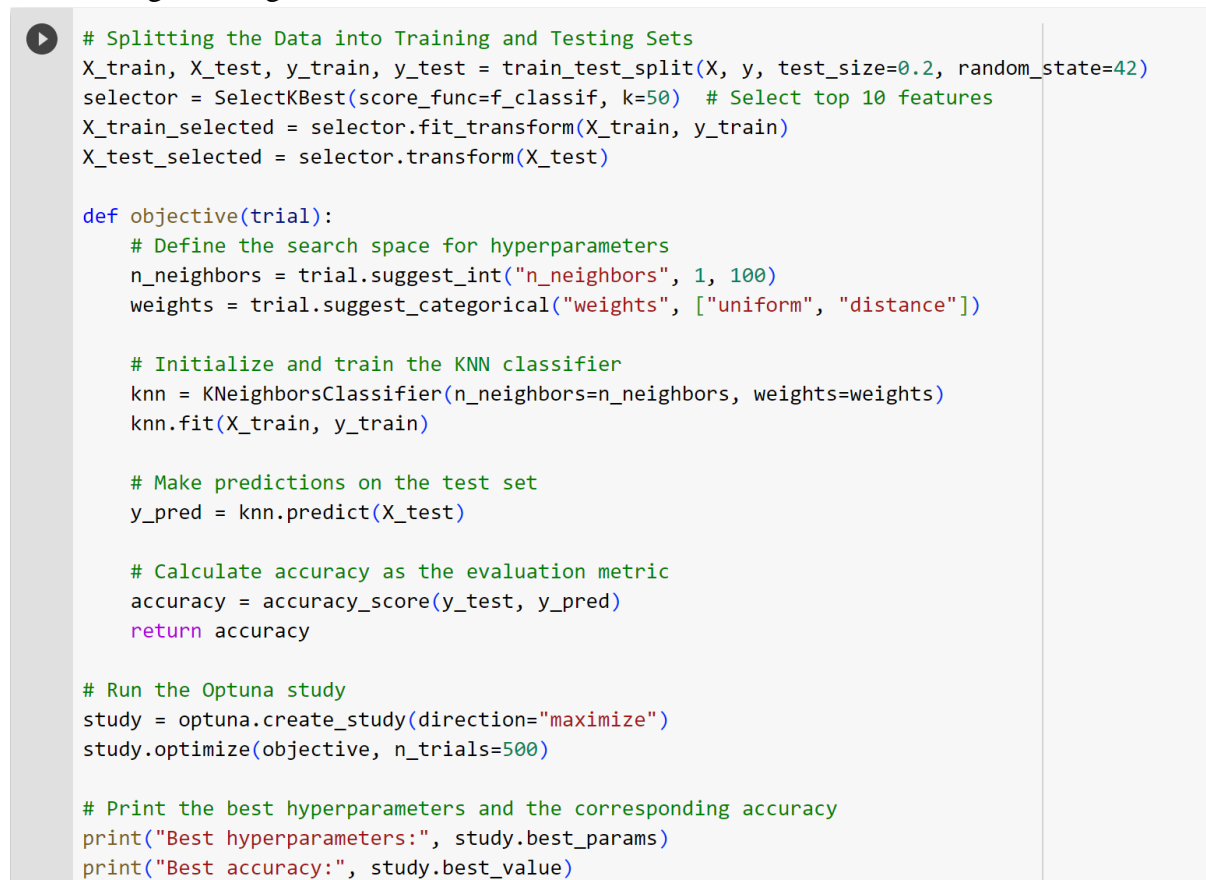
# Run the Optuna study
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100)

# Print the best hyperparameters and the corresponding accuracy
print("Best hyperparameters:", study.best_params)
print("Best accuracy:", study.best_value)
```

Gambar 4. Implementasi Decision Tree Algorithm Menggunakan *Library*

Pada kode di atas, modul Optuna digunakan untuk melakukan optimisasi *hyperparameter* sehingga akurasi yang didapatkan lebih tinggi. Selanjutnya, data dipisah menjadi data latih dan data tes. Setelah itu, dilakukan pencarian terhadap 50 atribut terbaik yang dapat meningkatkan akurasi. Kemudian, dilakukan coba-coba (*trial and error*) terhadap banyaknya tetangga yang dimasukkan ke dalam perhitungan dan pendekatan pembobotan jarak ketetanggaan yang dilakukan. Terdapat dua pendekatan, yaitu: Gini yang menggunakan tingkat ketidakmurnian suatu data dan Entropy yang menggunakan tingkat ketidakteraturan suatu data. Kode mengembalikan *hyperparameter* dan akurasi terbaik dari coba-coba yang dilakukan.

Berikut adalah penjelasan dari *library* yang digunakan dan penggunaannya untuk k-nearest neighbors algorithm.



```
# Splitting the Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
selector = SelectKBest(score_func=f_classif, k=50) # Select top 10 features
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

def objective(trial):
    # Define the search space for hyperparameters
    n_neighbors = trial.suggest_int("n_neighbors", 1, 100)
    weights = trial.suggest_categorical("weights", ["uniform", "distance"])

    # Initialize and train the KNN classifier
    knn = KNeighborsClassifier(n_neighbors=n_neighbors, weights=weights)
    knn.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = knn.predict(X_test)

    # Calculate accuracy as the evaluation metric
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy

# Run the Optuna study
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=500)

# Print the best hyperparameters and the corresponding accuracy
print("Best hyperparameters:", study.best_params)
print("Best accuracy:", study.best_value)
```

Gambar 5. Implementasi K-Nearest Neighbors Algorithm Menggunakan *Library*

Pada kode di atas, modul Optuna digunakan untuk melakukan optimisasi *hyperparameter* sehingga akurasi yang didapatkan lebih tinggi. Selanjutnya, data dipisah menjadi data latih dan data tes. Setelah itu, dilakukan pencarian terhadap 50 atribut terbaik yang dapat meningkatkan akurasi. Kemudian, dilakukan coba-coba (*trial and error*) terhadap banyaknya tetangga yang dimasukkan ke dalam perhitungan dan pendekatan pembobotan jarak ketetanggaan yang dilakukan. Terdapat dua pendekatan, yaitu: Uniform yang memberikan bobot yang seragam terhadap titik-titik yang bertetangga dan Distance yang menggunakan jarak sesungguhnya sebagai bobot yang diberikan. Kode mengembalikan *hyperparameter* dan akurasi terbaik dari coba-coba yang dilakukan.

C. Tautan-Tautan

Berikut adalah tautan Google Colab hasil pengerjaan proyek ini:
https://colab.research.google.com/drive/1ivPbFCwv_N8Wpff86T0MFv3pG81w42RU?usp=sharing.

Berikut aset salindia dan video rekaman presentasi pengerjaan proyek ini:
<https://drive.google.com/drive/folders/1FqjnyglwS21GFF0jiBcWhOas66ZeQkWt>.

III. EVALUASI DAN DISKUSI

A. Cara Pengukuran Performa Model dan Hasil

Performa model pembelajaran mesin yang kami buat diukur dengan menggunakan penilaian akurasi yang masing-masingnya didefinisikan sebagai berikut.

- a. Decision tree algorithm tanpa library: membandingkan total hasil prediksi yang tepat dengan seluruh data prediksi.
- b. K-nearest neighbors algorithm tanpa library: membandingkan total hasil prediksi yang tepat dengan seluruh data prediksi pada masing-masing bagian (fold), lalu menghitung rata-rata dari ketiga bagian data tersebut.
- c. Decision tree dengan library: melalui modul Optuna, mengembalikan nilai terbaik dari coba-coba (*trial and error*) yang dilakukan.
- d. K-nearest neighbors algorithm dengan library: melalui modul Optuna, mengembalikan nilai terbaik dari coba-coba (*trial and error*) yang dilakukan.

Hasil, yaitu akurasi yang kami dapatkan adalah sebagai berikut.

- a. Decision tree algorithm tanpa library: 67,0%.
- b. Decision tree dengan library: 73,0%.
- c. K-nearest neighbors algorithm tanpa library dengan jarak Euclidean: 60,2%.
- d. K-nearest neighbors algorithm tanpa library dengan jarak Manhattan: 60,5%.
- e. K-nearest neighbors algorithm dengan library: 58,2%.

B. Analisis

Berdasarkan hasil yang kami peroleh, kami menemukan bahwa pada himpunan data yang diprediksi dengan decision tree algorithm lebih akurat daripada yang diprediksi dengan k-nearest neighbors algorithm. Berdasarkan hasil yang ditemukan pula, terlihat bahwa penggunaan *library* dalam decision tree algorithm meningkatkan akurasi. Hal yang paling mungkin menjadi penyebab terjadinya hal tersebut adalah karena penggunaan *hyperparameter tuning* dan optimasi dari modul tersebut yang baik. Sementara itu, pada k-nearest neighbors algorithm, penggunaan library justru menghasilkan akurasi yang lebih rendah daripada algoritma yang kami buat sendiri. Hal yang paling mungkin menjadi penyebabnya adalah perbedaan parameter dan/atau implementasi dari kedua pendekatan tersebut.

IV. KESIMPULAN DAN SARAN

A. Kesimpulan

Kesimpulan yang dapat kami berikan dari proyek ini adalah himpunan data (*dataset*) yang diberikan lebih cocok/sesuai diolah menggunakan decision tree algorithm daripada k-nearest neighbors algorithm. Hal tersebut dibuktikan dengan hasil yang kami dapatkan, yakni akurasi yang diberikan decision tree algorithm lebih baik daripada yang diberikan k-nearest neighbors algorithm secara keseluruhan.

B. Saran

Saran yang dapat kami berikan untuk peneliti selanjutnya adalah untuk lebih banyak melakukan eksplorasi data untuk mendapatkan wawasan lebih agar prapemrosesan data dapat dilakukan lebih efisien. Selain itu, algoritma yang kami buat sangat mungkin untuk lebih dioptimasi dengan mengeksplorasi penggunaan parameter lain yang lebih bervariasi.

Berikut adalah tautan untuk mengakses aset salindia, video presentasi, dan laporan akhir: <https://drive.google.com/drive/folders/1FqjnyglwS21GFF0jiBcWhOas66ZeQkWt>.

DAFTAR PUSTAKA

- [1] Alodokter, “Aritmia,” *Alodokter*, Apr. 29, 2015. <https://www.alodokter.com/aritmia>
- [2] D. T. Larose, *Discovering knowledge in data : an introduction to data mining*. Hoboken, N.J.: Wiley-Interscience, 2005.
- [3] J. Han, M. Kamber, and J. Pei, *Data mining : concepts and techniques*. Burlington, Ma: Elsevier, 2012.
- [4] H. A. Guvenir, B. Acar, G. Demiroz, and A. Cekin, “A supervised machine learning algorithm for arrhythmia analysis,” *Computers in Cardiology 1997*, vol. 1, no. 110, 1997, doi: <https://doi.org/10.1109/cic.1997.647926>.
- [5] S. Dash, “Decision Trees Explained — Entropy, Information Gain, Gini Index, CCP Pruning..,” *Medium*, Nov. 02, 2022. <https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c>
- [6] B. Charbuty and A. Abdulazeez, “Classification Based on Decision Tree Algorithm for Machine Learning,” *Journal of Applied Science and Technology Trends*, vol. 2, no. 01, pp. 20–28, Mar. 2021, doi: <https://doi.org/10.38094/jastt20165>.
- [7] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, “An introduction to decision tree modeling,” *Journal of Chemometrics*, vol. 18, no. 6, pp. 275–285, Jun. 2004, doi: <https://doi.org/10.1002/cem.873>.