

## K-Nearest Neighbors (KNN)

Adinda Putri - 13523071

**K-Nearest Neighbors** merupakan salah satu model klasifikasi berdasarkan *instance-based learning*. Pertama-tama, model menyimpan training set. Kemudian, ketika model dihadapkan dengan *instance* baru atau test set misalnya, model membuat hubungan antara training set dengan *instance* baru tersebut untuk menentukan nilai fungsi target untuk *instance* tersebut. Asumsi dasar pada KNN adalah sebagai berikut:

- Semua *instance* berkorespondensi dengan titik-titik pada ruang  $n$ -dimensi, dengan  $n$  merepresentasikan jumlah fitur pada tiap *instance*.
- Jarak antartitik didefinisikan dalam Euclidean, Manhattan, atau Minkowski distance

### Cara Kerja

Pertama, KNN *Classifier* menyimpan training set. Saat prediksi, ketika model dihadapkan dengan *instance* baru (atau test set) untuk melakukan prediksi, model mencari  $K$  *instances* dari training set yang paling dekat dengan *instance* baru tersebut. Model ini lalu menetapkan class yang paling dekat untuk *instance* baru tersebut.

Pseudocode:

1. Simpan semua data training.
2. Ulangi langkah 3, 4, 5 untuk tiap data test.
3. Temukan  $K$  data training yang paling dekat ke data test.
4.  $y_{pred}$  untuk data test = class yang paling sering muncul di antara  $K$  tetangga terdekat dari data test.

Pemilihan nilai  $K$  yang optimal dilakukan dengan cara validasi error pada training set, seperti k-fold cross-validation.  $K$  juga dapat dipilih sebagai akar dari  $m$ , dengan  $m$  adalah jumlah training set. Pada eksperimen ini, dilakukan pencarian nilai  $K$  yang mengoptimalkan recall untuk masing-masing metode perhitungan jarak. Pencarian ini dilakukan dengan melakukan k-fold cross-validation untuk masing-masing nilai  $K$ , dengan jumlah fold sebanyak 10.. Didapatkan nilai  $K$  yang optimal untuk masing-masing metode perhitungan jarak adalah sebagai berikut

- Nilai K Optimal untuk metode 'euclidean' = 5



Gambar 1. Nilai K Optimal untuk metode 'Euclidean'

Sumber: Penulis

- Nilai K Optimal untuk metode 'manhattan' = 5



Gambar 2. Nilai K Optimal untuk metode ‘Manhattan’

Sumber: Penulis

Setelah didapatkan nilai K optimal, dilakukan evaluasi model dengan K-Fold Cross-Validation dengan jumlah fold yang sama, yaitu sebanyak 10, dengan nilai K optimal tersebut. Lalu, dilakukan evaluasi model dengan hold-out validation menggunakan nilai K optimal tersebut.

**Perbandingan hasil evaluasi model from KNN from Scratch dengan KNN dari Scikit-Learn:**

- **Evaluasi model dengan K-Fold Cross Validation untuk K = 5 dan metrik jarak euclidean:**
  - **KNN from Scratch:**

### K-Fold Cross-Validation

```

n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

evaluate_model_classification(knn_pipe, X_train, y_train, kf)

```

[52] ✓ 7.3s Python

```

... --- Mengevaluasi Model dengan K-Fold Cross Validation ---
Rata-rata Akurasi: 79.21%
Rata-rata Presisi: 79.65%
Rata-rata Recall: 83.30%
Rata-rata F1-Score: 81.30%
-----
--- Laporan Klasifikasi & Confusion Matrix ---
Laporan Klasifikasi:

```

	precision	recall	f1-score	support
0	0.79	0.74	0.77	336
1	0.79	0.83	0.81	400
accuracy			0.79	736
macro avg	0.79	0.79	0.79	736
weighted avg	0.79	0.79	0.79	736

Gambar 3.K-Fold Cross-Validation KNN Metrik Euclidean K=5

Sumber: Penulis

#### - KNN from Scikit-Learn

### K-Fold Cross Validation KNN dari Scikit-Learn

```

cnt = 0
n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

evaluate_model_classification(knn_scikit_pipe, X_train, y_train, kf)

```

[53] ✓ 4.4s Python

```

... --- Mengevaluasi Model dengan K-Fold Cross Validation ---
Rata-rata Akurasi: 79.21%
Rata-rata Presisi: 79.65%
Rata-rata Recall: 83.30%
Rata-rata F1-Score: 81.30%
-----
--- Laporan Klasifikasi & Confusion Matrix ---
Laporan Klasifikasi:

```

	precision	recall	f1-score	support
0	0.79	0.74	0.77	336
1	0.79	0.83	0.81	400
accuracy			0.79	736
macro avg	0.79	0.79	0.79	736
weighted avg	0.79	0.79	0.79	736

Gambar 4. K-Fold Cross-Validation KNN dari Sklearn Metrik Euclidean K=5

Sumber: Penulis

#### - Evaluasi model dengan K-Fold Cross Validation untuk K = 5 dan metrik jarak manhattan:

##### - KNN from Scratch

```
Evaluasi Model untuk K tertentu dengan K-Fold Cross-Validation

evaluate_model_classification(knn_pipe, X_train, y_train, kf)

[ss] ✓ 7.2s Python

... --- Mengevaluasi Model dengan K-Fold Cross Validation ---
Rata-rata Akurasi: 80.30%
Rata-rata Presisi: 80.75%
Rata-rata Recall: 84.02%
Rata-rata F1-Score: 82.24%
-----
--- Laporan Klasifikasi & Confusion Matrix ---
Laporan Klasifikasi:
      precision    recall  f1-score   support

      0       0.80      0.76      0.78       336
      1       0.81      0.84      0.82       400

 accuracy          0.80          0.80          0.80          736
 macro avg          0.80          0.80          0.80          736
 weighted avg          0.80          0.80          0.80          736
```

Gambar 5.K-Fold Cross-Validation KNN Metrik Manhattan K=5

Sumber: Penulis

#### - KNN from Scikit-Learn

```
K-Fold Cross Validation KNN dari Scikit-Learn

n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

evaluate_model_classification(knn_scikit_pipe, X_train, y_train, kf)

[se] ✓ 4.7s Python

... --- Mengevaluasi Model dengan K-Fold Cross Validation ---
Rata-rata Akurasi: 80.30%
Rata-rata Presisi: 80.75%
Rata-rata Recall: 84.02%
Rata-rata F1-Score: 82.24%
-----
--- Laporan Klasifikasi & Confusion Matrix ---
Laporan Klasifikasi:
      precision    recall  f1-score   support

      0       0.80      0.76      0.78       336
      1       0.81      0.84      0.82       400

 accuracy          0.80          0.80          0.80          736
 macro avg          0.80          0.80          0.80          736
 weighted avg          0.80          0.80          0.80          736
```

Gambar 6. K-Fold Cross-Validation KNN dari Sklearn Metrik Manhattan K=5

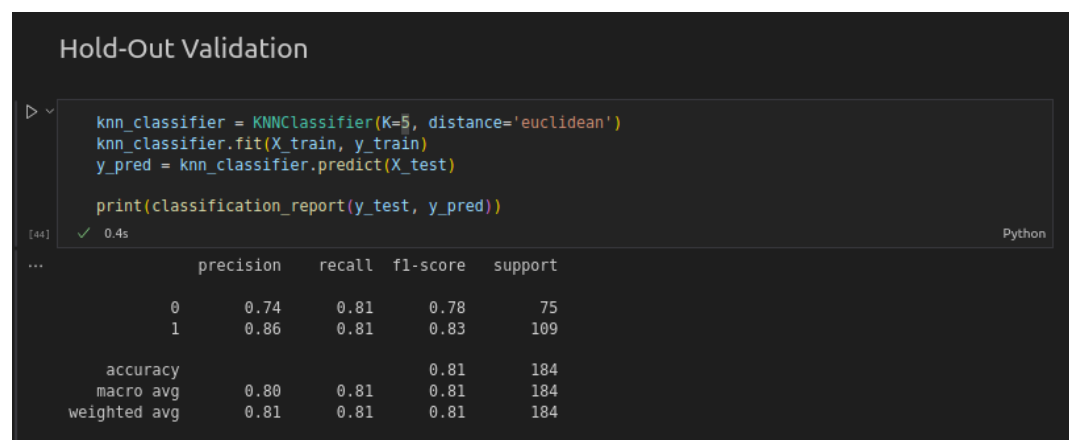
Sumber: Penulis

Terlihat bahwa hasil yang sama didapatkan untuk masing-masing model KNN. perbedaannya adalah dalam waktu eksekusinya. Perbedaan eksekusi ini terjadi karena adanya perbedaan implementasi pada algoritma KNN. Pada KNN from scratch, perhitungan jarak dilakukan

dengan *vectorized* brute force. Akan tetapi, pada implementasi predict-nya, digunakan for loop untuk masing-masing baris data pada test data (belum teroptimasi).

Sementara itu, terdapat perbedaan hasil untuk masing-masing metode perhitungan jarak. Dari hasil, terlihat bahwa hasil lebih baik didapatkan metode manhattan. Hal ini dapat terjadi karena pengaruh outliers yang besar pada perhitungan dengan euclidean. Manhattan distance lebih baik karena tidak ada dominasi dari fitur tertentu dan setiap dimensi berkontribusi linear, bukan kuadratik, sesuai dengan karakteristik alami dari fitur yang cocok diukur secara individual bukan geometris.

- **Evaluasi model dengan Hold-Out Validation untuk K = 5 dan metrik jarak euclidean:**
  - **KNN from Scratch**



```
Hold-Out Validation

knn_classifier = KNNClassifier(K=5, distance='euclidean')
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)

print(classification_report(y_test, y_pred))
```

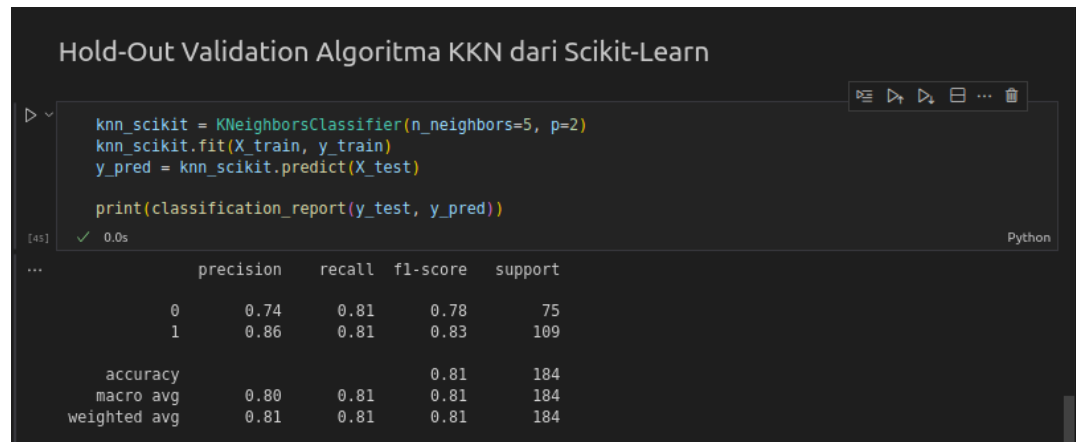
[44] ✓ 0.4s Python

	precision	recall	f1-score	support
0	0.74	0.81	0.78	75
1	0.86	0.81	0.83	109
accuracy			0.81	184
macro avg	0.80	0.81	0.81	184
weighted avg	0.81	0.81	0.81	184

Gambar 7. Hold-Out Validation KNN Metrik Euclidean K=5

Sumber: Penulis

- **KNN from Scikit-Learn**

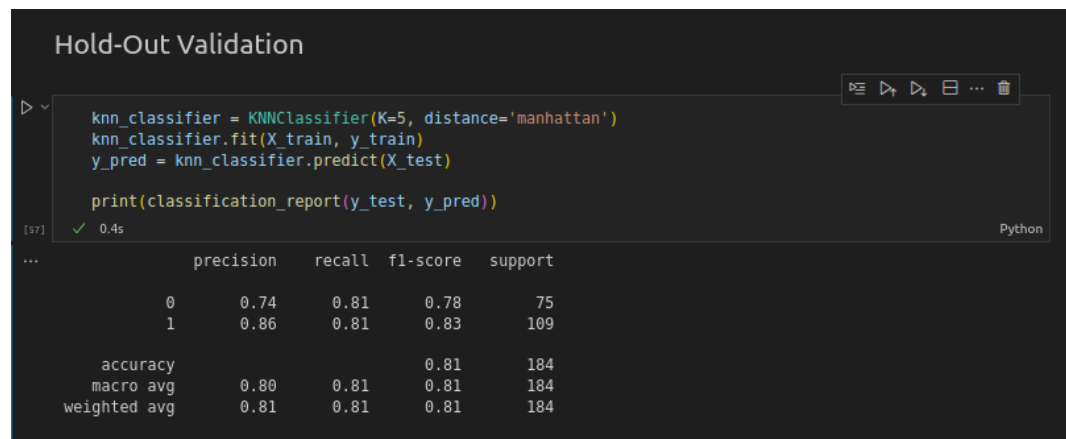


Gambar 8. Hold-Out Validation KNN dari Sklearn Metrik Euclidean K=5

Sumber: Penulis

- **Evaluasi model dengan Hold-Out Validation untuk K = 5 dan metrik jarak manhattan:**

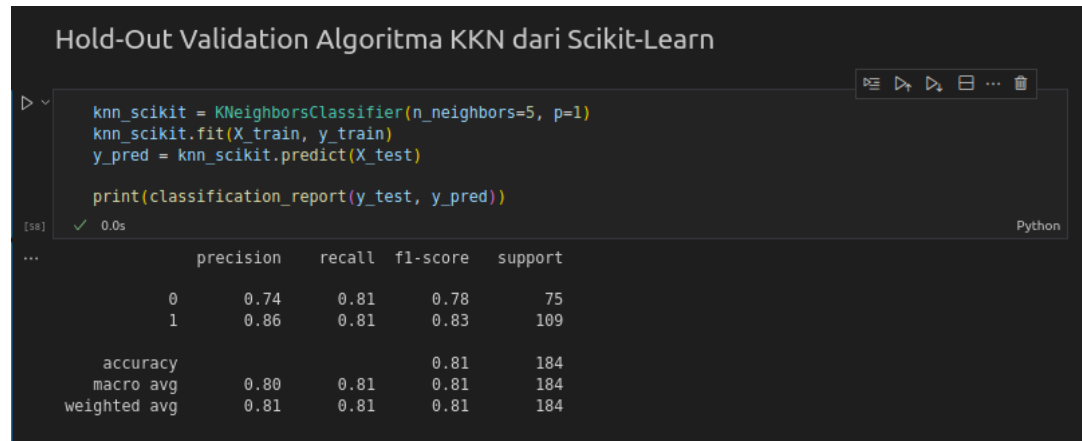
- **KNN from Scratch**



Gambar 9. Hold-Out Validation KNN Metrik Manhattan K=5

Sumber: Penulis

- **KNN from Scikit-Learn**



Gambar 10. Hold-Out Validation KNN dari Sklearn Metrik Manhattan K=5

Sumber: Penulis

Terlihat bahwa hasil evaluasi untuk masing-masing model juga sama untuk evaluasi dengan Hold-Out Validation. Salah satu perbedaannya juga sama yaitu terletak di waktu eksekusi dengan alasan yang sama seperti perbedaan waktu evaluasi pada K-Fold Cross-Validation.

Sementara itu, untuk masing-masing metode perhitungan jarak juga didapatkan hasil yang sama. Hasil sama ini dapat terjadi akibat pembagian data pada hold-out validation. Terdapat kemungkinan besar bahwa outlier-outlier yang kebetulan masuk ke data testing di beberapa k-fold tidak masuk ke data testing pada hold-out validation, sehingga keunggulan manhattan distance dalam menangani outlier tidak terlihat.

### Ruang Improvement:

Terdapat beberapa hal yang bisa dilakukan untuk meningkatkan performa model KNN from Scratch ini, yaitu:

- Optimasi Kode KNN: Pada implementasi method predict, hanya sebagian kode menerapkan vectorized operation, sehingga masih terdapat for loop yang membuat waktu eksekusi cukup lambat. Hal ini dapat diatasi dengan menggunakan vectorized operation secara keseluruhan.
- Eksplorasi Hyperparameter Tambahan: Dapat diuji parameter weights di mana tetangga yang lebih dekat akan memiliki pengaruh yang lebih besar daripada tetangga yang lebih jauh.



- Implementasi Algoritma Pencarian K-Nearest Neighbors yang lebih efisien: KNN from scratch menggunakan metode brute-force yang sudah cukup dioptimasi dengan vectorized operation. Akan tetapi, masih terdapat algoritma KD-Tree atau Ball-Tree yang lebih advance dan digunakan oleh KNN dari Scikit-Learn.
- Penanganan Outliers: Dapat melakukan konsultasi atau research lebih dalam tentang domain-knowledge, dalam hal ini bidang kedokteran yang berkaitan dengan penyakit jantung.

## Referensi:

**[1]** *Implementation of K-Nearest Neighbors from Scratch using Python*, GeeksforGeeks. [Daring]. Tersedia:

<https://www.geeksforgeeks.org/machine-learning/implementation-of-k-nearest-neighbors-from-scratch-using-python/>. [Diakses: 1 September 2025].

**[2]** *sklearn.neighbors.KNeighborsClassifier*, scikit-learn API Reference, 2025. [Daring]. Tersedia:

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>. [Diakses: 1 September 2025].