Bab 6
K-Nearest Neighbour

A. KOMPETENSI DASAR

- Memahami konsep *KNN*.
- Memahami eksperimen klasifikasi menggunakan KNN
- Memahami membuat model KNN

B. ALOKASI WAKTU

4 js (4x50 menit)

C. PETUNJUK

- Awali setiap aktivitas dengan do'a, semoga berkah dan mendapat kemudahan.
- Pahami Tujuan, dasar teori, dan latihan-latihan praktikum dengan baik dan benar.
- Kerjakan tugas-tugas dengan baik, sabar, dan jujur.
- Tanyakan kepada teman lalu asisten/dosen apabila ada hal-hal yang kurang jelas.

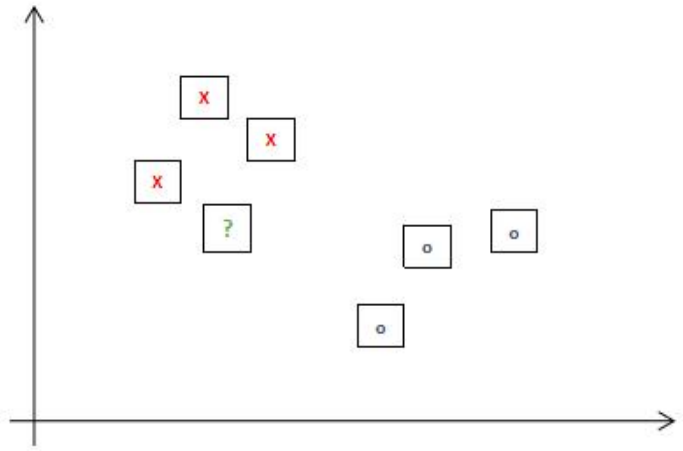
D. DASAR TEORI K-NN

K-Nearest Neighbor atau KNN atau K-NN adalah algoritma supervised machine learning paling sederhana dibandingkan yang lainnya. KNN dapat digunakan untuk klasifikasi maupun regresi, namun paling umum untuk memodelkan klasifikasi. Beberapa alasan KNN dipilih untuk membangun sebuah model prediksi:

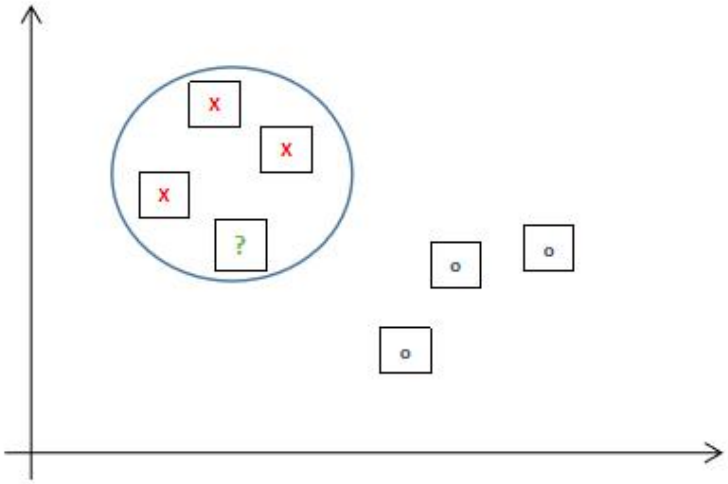
1. Interpretasi output yang sederhana
2. Waktu kalkulasi relative cepat
3. Kemampuan prediksi yang kuat

Untuk memahami bagaimana KNN bekerja, mari kita ambil sebuah kasus dua kelas lingkaran (O) dan silang (X). Asumsikan terdapat 6 sampel dengan label masing-masing dan satu sampel (?) yang ingin diketahui kelas nya (lihat Gambar 1). Pilihan untuk sampel (?) ini adalah kelas O atau X.

Misal, pada suatu iterasi pemodelan KNN dengan K=3, maka sample (?) akan masuk dalam kelas X, sesuai dengan ilustrasi pada Gambar 2. Hal tersebut dikarenakan sampel (?) memiliki jarak terdekat dengan 3 sampel dari kelas X, sedangakan tidak ada satu pun sampel O yang dekat dengan sampel ?. Pemilihan nilai K pada algoritma ini sangat menentukan bagaimana model memprediksi suatu kasus. Oleh karena itu bagaimana mencari nilai K yang tepat untuk prediksi menggunakan algoritma KNN ini?

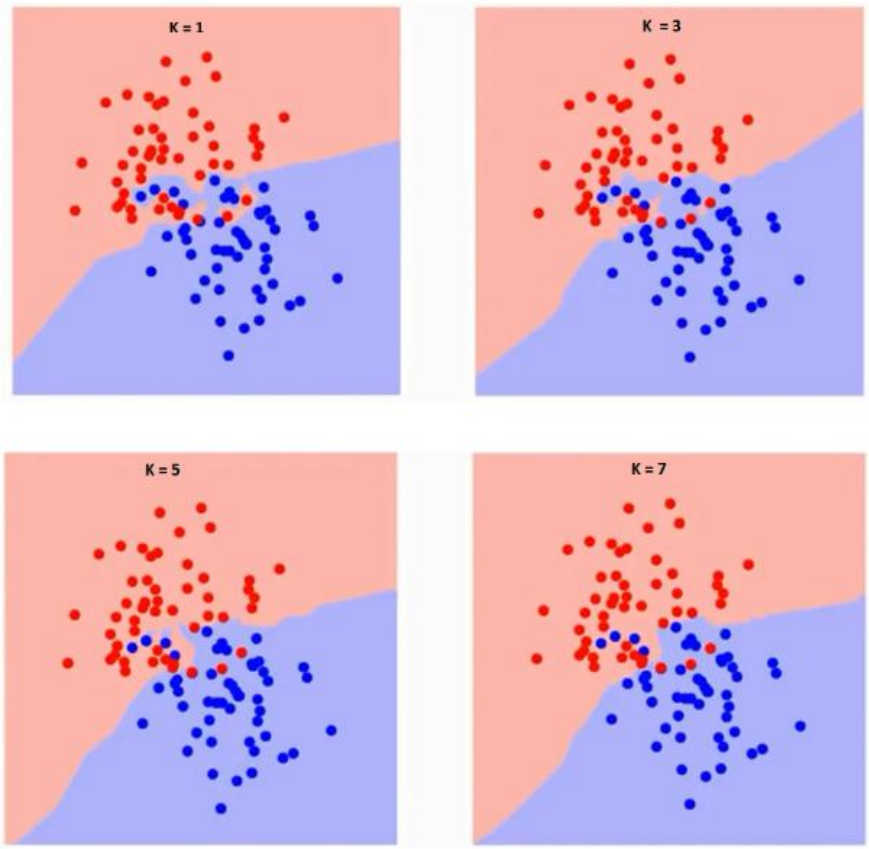


Gambar 1. Contoh Kasus Dua Kelas



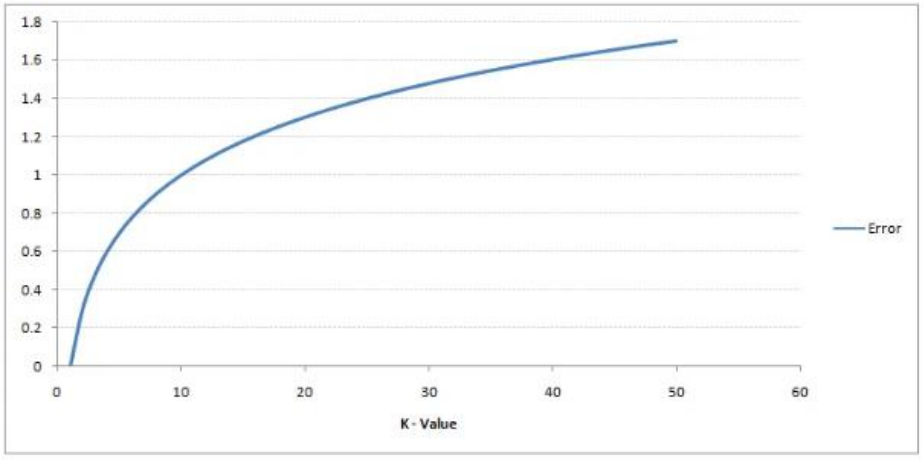
Gambar 2. KNN dengan K=3

Pertama, mari kita coba memahami apa sebenarnya pengaruh K dalam algoritma. Jika kita melihat contoh kasus tersebut, mengingat bahwa 6 sampel dengan label yang konstan, maka dengan suatu nilai K dapat membentuk batasan masing-masing kelas. Batas-batas ini akan memisahkan kelas X dari kelas O. Dengan cara yang sama, mari kita coba untuk melihat efek dari nilai "K" pada batas kelas. Berikut ini adalah batasan yang berbeda yang memisahkan dua kelas dengan nilai K. yang berbeda.



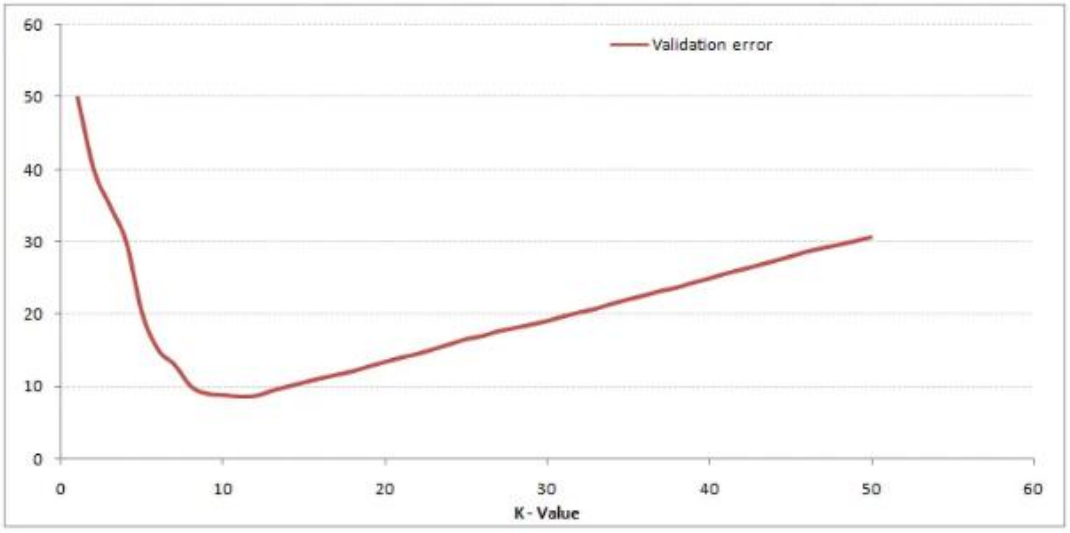
Gambar 3. Pembentukan garis batas kelas pad nilai K yang bervariasi

Jika Anda memperhatikan dengan seksama, Anda dapat melihat bahwa garis batas kelas menjadi lebih halus dengan meningkatnya nilai K. Dengan K bertambah hingga tak terhingga menyebabkan semua sampel masuk kelas biru atau merah tergantung pada total mayoritas. Tingkat kesalahan ketika training dan tingkat kesalahan ketika validasi adalah dua parameter yang perlu kita akses pada nilai-K yang berbeda. Berikut ini adalah kurva untuk tingkat kesalahan pelatihan dengan nilai K yang bervariasi:



Gambar 4. Grafik Error pada nilai K yang bervariasi

Seperti yang Anda lihat, tingkat kesalahan pada K = 1 selalu nol untuk sampel pelatihan. Ini karena titik terdekat ke titik data pelatihan adalah titik itu sendiri. Oleh karena itu prediksi akan selalu akurat dengan K = 1. Jika kurva kesalahan validasi akan serupa, pilihan K kami adalah 1. Berikut ini adalah kurva kesalahan validasi dengan nilai K yang bervariasi:



Gambar 5. Grafik Error pada validasi dengan K bervariasi

Dengan grafik tersebut maka semakin jelas bahwa pada K = 1, model prediksi mengalami *overfitting*. Hal ini dapat dilihat dengan tingkat kesalahan yang sangat tinggi di awal dan berangsur turun hinga mencapai minima pada K=10. Setelah titik minima, error validasi berangsur naik kembali. Untuk mendapatkan nilai K yang optimal, Anda dapat memisahkan pelatihan dan validasi dari set data awal. Denga plot kurva kesalahan validasi tersebut membantu untuk mendapatkan nilai optimal K.

Algoritma KNN secara umum adalah sebagai berikut:

1. Muat data
2. Menginisialisasi nilai k
3. Untuk mendapatkan kelas yang diprediksi, lakukan iterasi dari 1 ke jumlah total sampel *training set*
   a) Hitung jarak antara data tes dengan setiap sample pada training set. Di sini kita akan menggunakan jarak Euclidean sebagai metrik jarak. Metrik lain yang dapat digunakan adalah Manhattan, Chebyshev, cosinus, dll.
   b) Urutkan (*ascending*) pasangan sampel dengan data latih berdasarkan jarak yang telah diukur.
   c) Dapatkan sejumlah K sampel data latih pada urutan teratas (terdekat dengan sampel tes).
   d) Lakukan agregasi dengan menghitung kelas yang paling sering muncul hasil dari tahap c.
   e) Kelas hasil prediksi adalah kelas yang paling banyak.

E. LATIHAN (jawaban dapat ditulis pada halaman baru)

1. Persiapkan data berupa array 2 dimensi sebagai berikut: `np.array([[7,3], [1,10], [5,5], [2,1], [3,3], [2,7], [17,14], [6,8], [7,5], [3,6]])` sebagai variable X. Kemudian buatkan array 1 dimensi: `np.array([1, 1, 1, 2, 2, 2, 1, 2, 1])`, sebagai label atau variabel y.

2. Implementasikan KNN classifier:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

k_neighbors = 2

# persiapkan data input dan label nya
feature = np.array([[7,3],  [1,10],  [5,5],  [2,1],  [3,3],  [2,7],
[17,14], [6,8],  [7,5], [3,6]])
X = feature
y = np.array([1, 1, 1, 2, 2, 2, 1, 2, 1])

h = .02  # step size in the mesh
# pemetaan warna untuk plot
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

for weights in ['uniform', 'distance']:
    # training KNN dengan 2 parameter: k dan weights.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot decision boundary dengan warna yang berbeda
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # hasil klasifikasi pada peta warna
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
    # Plot data training
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("2-Class classification (k = %i, weights = '%s')"
              % (k_neighbors, weights))

plt.show()
```

3. Lakukan eksplorasi parameter KNeighborsClassifier yang terdiri dari:
   a) **n_neighbors : int, optional (default = 5)**
   Number of neighbors to use by default for kneighbors queries.
   b) **weights : str or callable, optional (default = 'uniform')**
   weight function used in prediction. Possible values:
      i. 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
      ii. 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
      iii. [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
   c) **algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, optional**
   Algorithm used to compute the nearest neighbors:
      i. 'ball_tree' will use BallTree
      ii. 'kd_tree' will use KDTree
      iii. 'brute' will use a brute-force search.
      iv. 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method.
   d) **leaf_size : int, optional (default = 30)**
   Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
   e) **p : integer, optional (default = 2)**
   Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
   f) **metric : string or callable, default 'minkowski'**
   the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of the DistanceMetric class for a list of available metrics.
   g) **metric_params : dict, optional (default = None)**
   Additional keyword arguments for the metric function.
   h) **n_jobs : int or None, optional (default=None)**
   The number of parallel jobs to run for neighbors search. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details. Doesn't affect fit method.

F. TUGAS

1. Buatlah atau carilah dataset sederhana dengan jumlah baris antara 100-200.
2. Lakukan eksperimen (dengan 5-fold cross validation) dengan nilai **K yang bervariasi** dan **parameter metric berupa Euclidean**. Tambahkan tiap eksperimen yang anda lakukan dengan mengukur performa dari tiap model yang dihasilkan berupa: **akurasi, presisi dan recall**.
3. Buatlah kesimpulan tentang Model KNN berdasarkan hasil eksperimen anda.