KNN

→ dated 3-6-25

from sklearn.neighbors import
KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 3)

knn.fit (x_train, y_train)

3) Experiment with diff. k values

Use cross_val_score to evaluate scores (stability). for different 'k' values.

Visualize decision boundaries

x_2d = x.iloc [:, 1:3]

→ take the first 2 features for 2D plotting

x_min, x_max = x_2d.iloc [:,0].min()-1,
x_2d.iloc [:,0].max()+1

y_min, y_max = x_2d.iloc[:,1].min()-1,
x_2d.iloc [:,1].max()+1

} getting min and max value from each column. We use ±1 to ex

xx, yy = np.meshgrid(np.arange(x.min, x.max, 0.02), np.arange(y.min, y.max, 0.02))

Eg:

If x = [1, 2, 3]
y = [4, 5]

generates value from x.min to x.max with step as 0.02
(i.e) if x.min = 0 x.max = 1
=> [0 0.02 0.04 0.06 ... 1]

xx, yy = np.meshgrid(x, y)

=> xx = [[1, 2, 3]
[1, 2, 3]]

yy = [[4, 4, 4]
[5, 5, 5]]

→ 2D Array

mesh_points = pd.DataFrame(np.c_[xx.ravel(), yy.ravel()], columns = X_2d.columns)

makes xx and yy as 1D array

Output

[1, 2, 3, 1, 2, 3] → xx
[4, 4, 4, 5, 5, 5] → yy

And np.c_ this (1,4) (2,4) (3,4) ...

And pd.DataFrame creates dataset with X_2d.columns names as Column names.

Final Output is like see in Notebook

decision boundary plot nicely surrounds all data points

I = labels = knn.predict(mesh_points)

It is nothing but

Sepal width	Petal length
1.00	0.00
1.02	0.00

Like this and it do our normal knn thing (i.e) finding Euclidian distance for each row of this mesh-points with ~~trained dataset of knn~~. the original dataset

Note:

In KNN model, it doesn't learn any pattern and then predict like normal models.

It simply stores the full dataset (trained dataset) during modelling or training.

And later if we want to predict it with unseen data (test set), it simply computes Euclidian distance → value of any unseen set

Eg: $X = (x_1, x_2, \ldots x_k)$ $Y = (y_1, y_2, \ldots y_k)$ → each row value of train set

$$\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\ldots(x_k-y_k)}$$

After this we obtain list of values (for each row), now we need to consider k nearest values (distance) and then take majority voting for the outcome (output)

Eg: Predict : (2.5, 3)

If

Point =>	x (Sepalwidth)	y (Petallength)	Label
	1	2	A
	2	3	A
	3	3	B
	6	5	B

in our example

Compute distances to all points

(1,2) and (2.5,3) => $\sqrt{(2.5-1)^2+(3-2)^2} \approx 1.8$

Ily for all other (2,3) => 0.5,
(3,3) => ≈0.5
(6,5) => ≈4.0

if k=3

(1,2), (2,3) (3,3) is nearest, And labels for that are A, A and B

majority is A

So the answer is A.

~~Z_labels~~

class_to_int = {label:idx for idx, label in enumerate(knn.classes_)}

this creates a dictionary that has each classe and with value as integers like 0, 1, 2.. like that.

↓ we have 3 for this dataset

Z = np.array([class_to_int[label] for label in Z_labels])

converts our Z_label is string output to integers as class_to_int.

Z = Z.reshape(xx.shape)

we have to convert Z into 2D array because only if Z is 2d arr. then only we can plot Decision Boundary

For Eg:

We plot Decision Boundary By computing x[i][j] and y[i][j] → So we need Z[i][j]

↓ 2D array

plt.contourf(xx, yy, Z, alpha=0.4) → diagram's transparency

① Defines x and y axis values

x-axis ⇒ xx.min to x.max

y-axis ⇒ yy.min to y.max

We have to do this, because whenever,

⇒ Z[i][j] = 0, it fills that patch of the plot with color for class 0.

⇒ Z[i][j] = 1, it fills with color for class 1.

⋮

And so on

② Contourf looks at 2D array Z and draws regions where values are the same.

③ It automatically finds boundaries when Z changes (ie) class 0 ends and class 1 begins

This is how we get our Decision boundary.