Lab 2.1. K-nearest neighbors (KNN)

NỘI DUNG 1

1.1. KNN

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing *K*, the user can select the number of nearby observations to use in the algorithm.

Here, we will show you how to implement the KNN algorithm for classification and show how different values of *K* affect the results.

1.2. How does it work?

K is the number of nearest neighbors to use. For classification, a majority vote is used to determined which class a new observation should fall into. Larger values of *K* are often more robust to outliers and produce more stable decision boundaries than very small values (*K=3* would be better than *K=1*, which might produce undesirable results.

1.3. Example

Start by visualizing some data points:

```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]

plt.scatter(x, y, c=classes)
plt.show()
```

Q1. Explaining about the above code

Q2. Results?

1.4. Now we fit the KNN algorithm with K=1 and use it to classify a new data point:

```
from sklearn.neighbors import KNeighborsClassifier

data = list(zip(x, y))
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(data, classes)
```

```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Q3. Explaining about the above code

Q4. Results?

1.5. Now we do the same thing, but with a higher K value which changes the prediction:

```
knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(data, classes)

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Q5. Explaining about the above code

Q6. Results?

Note. For more information you can refer to the link:
https://www.w3schools.com/python/python_ml_knn.asp

NỘI DUNG 2.