Predictive Model kNN

Dr Liew How Hui

Jan 2021

Week 2

- Thursday 28 Jan 2021 is Thaipusam
- Friday morning Tutorial class cancelled.
- Assignment announcement on my website.
- Discussing the assignment.

Theory

The *k-nearest neighbours* (kNN) algorithm is a non-parametric method used for classification and regression.

The assumption of kNN is “similar inputs have similar outputs”. Based on this assumption, a test input $\boldsymbol{x}$ should be assigned the most common label amongst its k most similar training inputs.

Theory (cont)

Given a positive integer k and an input $\boldsymbol{x}$, the kNN algorithm first identifies the k points in the training data $(\boldsymbol{x}_i, y_i)$ that are "closest" to $\boldsymbol{x}$, represented by $N(\boldsymbol{x})$.

- For kNN classifier, the prediction is

$$h(\boldsymbol{x}) = \text{mode}(\{y'' : (\mathbf{x}'', y'') \in N(\boldsymbol{x})\}),$$

$$\mathbb{P}(Y = j | \boldsymbol{X} = \boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in N(\boldsymbol{x})} I(y_i = j).$$

- For kNN regressor, the prediction is

$$h(\boldsymbol{x}) = \frac{1}{k} \sum_{(\boldsymbol{x}'', y'') \in N(\boldsymbol{x})} y''.$$

Distance

The kNN algorithm fundamentally relies on a “distance” metric.

What is a “distance”?

A *distance* $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a function which satisfies the following conditions: For any $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, $\boldsymbol{x}_k$,

(i) $d(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$ and $d(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$ iff $\boldsymbol{x}_i = \boldsymbol{x}_j$;
(ii) $d(\boldsymbol{x}_i, \boldsymbol{x}_j) = d(\boldsymbol{x}_j, \boldsymbol{x}_i)$; and
(iii) $d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq d(\boldsymbol{x}_i, \boldsymbol{x}_k) + d(\boldsymbol{x}_k, \boldsymbol{x}_j)$ (triangle inequality).

Distance (cont)

The most common choice is the *Minkowski distance*:

$$d(\boldsymbol{x},\boldsymbol{z}) = \|\boldsymbol{x}-\boldsymbol{z}\|_r = \left(\sum_{i=1}^{p} |x_i - z_i|^r\right)^{\frac{1}{r}}, \quad \boldsymbol{x},\ \boldsymbol{z} \in \mathbb{R}^p. \quad (1)$$

Note that $\|\cdot\|^r$ is called the ℓ^r norm.

When $r = 1$, we have the *Manhattan distance*:

$\|\boldsymbol{x}-\boldsymbol{z}\|_1 = |x_1 - z_1| + |x_2 - z_2| + \cdots + |x_p - z_p|.$

When $r = 2$, we have the *Euclidean distance*:

$\|\boldsymbol{x}-\boldsymbol{z}\|_2 = \sqrt{(x_1 - z_1)^2 + (x_2 - z_2)^2 + \cdots + (x_p - z_p)^2}.$

Distance (cont)

When $r = \infty$, we have the *Chebyshev distance*:

$$||\boldsymbol{x} - \boldsymbol{z}||_\infty = \max\{|x_1 - z_1|, |x_2 - z_2|, \cdots, |x_p - z_p|\}.$$

The Euclidean distance is more sensitive to outliers than the Manhattan distance. When outliers are rare, the Euclidean distance performs very well and is generally preferred. When the outliers are significant, the Manhattan distance is more stable.

Question

Which of the following distance metric can not be used in k-NN?

A) Gower

B) Minkowski

C) Tanimoto

D) Jaccard

E) Mahalanobis

F) All can be used

Theory: Bayes Optimal Classifier

If we **know** $\mathbb{P}(y|\boldsymbol{x})$ (which is almost never the case), then the “optimal” prediction is

$$y^* = h_{\text{opt}}(\boldsymbol{x}) = \operatorname*{argmax}_y \mathbb{P}(y|\boldsymbol{x})$$

For this type of classifier, the **error rate** is

$$\epsilon_{BayesOpt} = 1 - \mathbb{P}(y^*|\boldsymbol{x})$$

Why is the Bayes optimal classifier interesting, if it cannot be used in practice?

It provides a **lower bound of the error rate**.

Theory (cont)

What is the **upper bound on the error**?

The constant classifier.

What about the kNN Classifier?

Theorem

As $n \to \infty$, the 1-NN error is no more than twice the error of the Bayes Optimal classifier:

$$\epsilon_{\text{BayesOpt}} \leq \epsilon_{NN} \leq 2\epsilon_{\text{BayesOpt}}$$

Similar guarantees hold for $k > 1$.

Implementation of kNN Classifier

A naive Python + Numpy implementation is listed below.

```
# https://github.com/joelgrus/data-science-from-scratch
from scipy.spatial.distance import euclidean
import logging
logging.basicConfig(level=logging.DEBUG)

def knn_classify(k: int, X, y, x_new, distance=euclidean) -> str:
    import numpy as np, collections
    dist_list = [distance(x, x_new) for x in X]
    logging.debug('distances={}'.format([round(d,4) for d in dist_list]))
    order = np.argsort(dist_list)
    k_nearest_labels = y[order[:k]]
    logging.debug('knn={}'.format(k_nearest_labels))
    vote_counts = collections.Counter(k_nearest_labels)
    winner, winner_count = vote_counts.most_common(1)[0]
    return winner, winner_count/k

if __name__=='__main__':
    from sklearn import datasets
    iris_df = datasets.load_iris()
    X = iris_df['data']; y = iris_df['target']
    knn_classify(1,X,y,[6,3,5,2]); knn_classify(1,X,y,[5,3,1,0])
```

Example of kNN Classifier

A sport school would like to group their new enrolled students into 2 groups, as according to the existing students' weight and height. The weight and height of 7 existing students with group are shown in the table below.

Student	Weight (kg)	Height (cm)	Group
A	29	118	A
B	53	137	B
C	38	127	B
D	49	135	B
E	28	111	A
F	24	111	A
G	30	121	A

Example (cont)

(a) Perform and use $k = 3$-NN method (with Euclidean distance) to predict which group the following students will be grouped into, based on **cut-off of 0.7** on group A.

Student	Weight (kg)	Height (cm)
H	35	120
I	47	131
J	22	115
K	38	119
L	31	136

Example (cont)

(b) The actual groups of the students are {A, B, A, B, B} for students {H, I, J, K, L} respectively. Construct a confusion matrix and calculate the accuracy measurements for a cut-off of 0.5 and a cut-off of 0.7.

(c) Write a Python script to produce the above calculations using the simple implementation by lecturer above and also use sklearn’s implementation.

(d) Is R script easier to write???

kNN Regressor and Example

The *kNN regressor* applies “mean” instead of “mode” to “predict” the output.

R: `FNN::knn.reg`

Python: `sklearn.neighbors.KNeighborsRegressor`

Example 2.4.1 (Exam SRM Study Manual, p225, Q15.10)

A continuous variable Y is modelled as a function of X using kNN with $k = 3$. With the following data:

X	5	8	15	22	30
Y	4	1	10	16	30

Calculate the fitted value of Y at $X = 12$. Try write a script using R (or Python) to perform the calculation for you.

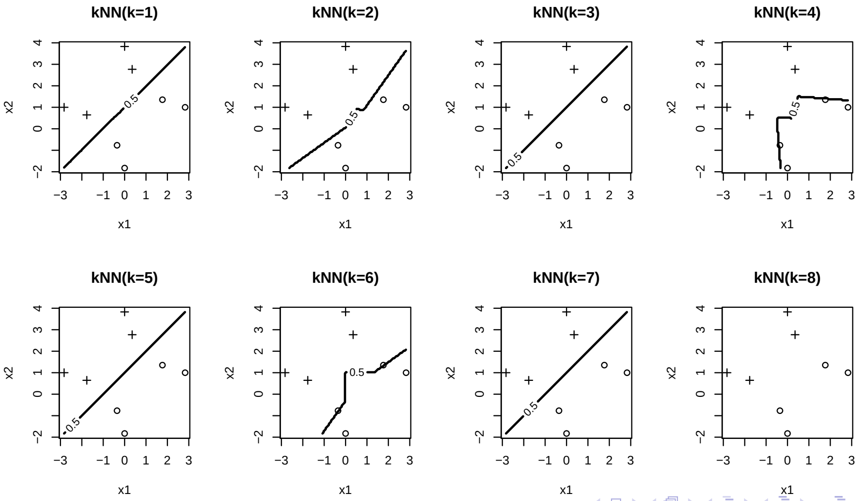
Classifier Boundary

What is “classifier boundary”? It is the boundary between different classes. We can think of it as the “boundaries” between “countries”, for example, Malaysia has boundaries with Singapore, Thailand and Indonesia.

Most data are not 2D, for example, the iris data. But most analysis will try to reduce to 2D to “visualise” the boundary. We can find a comprehensive analysis of the iris data using R with beautiful diagrams on `https://www.datacamp.com/community/tutorials/machine-learning-in-r`. As for the drawing the kNN boundary between classes, a “contour” plot is required and is illustrated by `https://stats.stackexchange.com/questions/21572/how-to-plot-decision-boundary-of-a-k-nearest-neighbor-c`

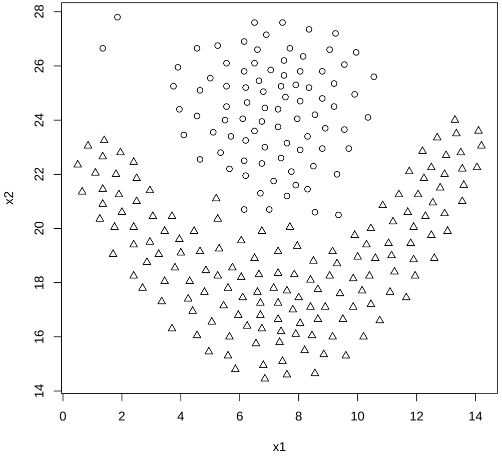
Classifier Boundary (cont)

First, consider the boundary of the ‘data1’ found in your Practical 1 which is symmetrical. Therefore, we “feel” that it is a “line”.

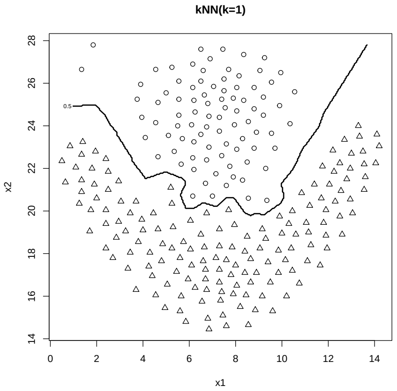


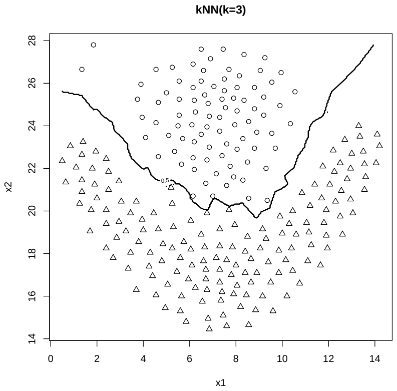
Classifier Boundary (cont)

Now, we investigate a “nonlinear boundary” data.

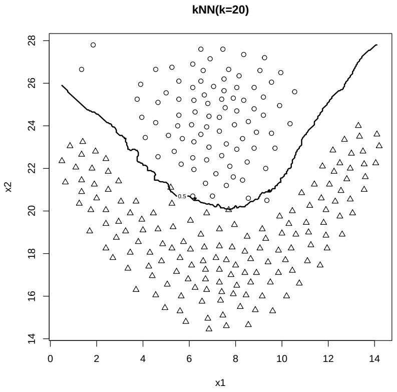


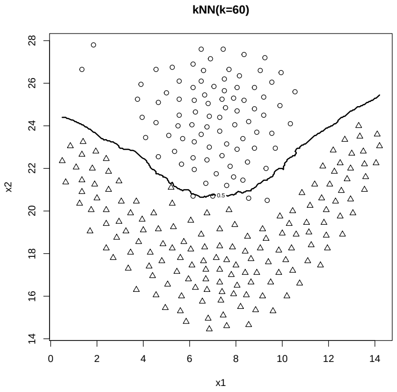
Classifier Boundary (cont)



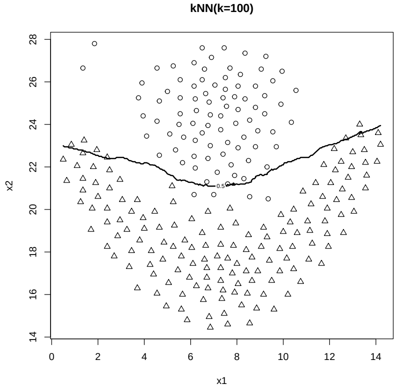


Classifier Boundary (cont)





Classifier Boundary (cont)



Conclusion: The choice of k depends upon the data.
In general,

- k small, decision boundary is over "flexible" ⇒ kNN classifier is low bias, high variance.
- k large, decision boundary is less flexible, "less noise", "smoother", more "linear" ⇒ kNN classifier is low-variance, high bias.

Feature Scaling / Standardisation

In the above example, if we change student weight to *g* and height to metre, what would happen?

Student	Weight (g)	Height (m)	Group
A	29,000	1.18	A
B	53,000	1.37	B
C	38,000	1.27	B
D	49,000	1.35	B
E	28,000	1.11	A
F	24,000	1.11	A
G	30,000	1.21	A

Some numbers are too large, while others are too small!

Feature Scaling (cont)

Bad for "predictive model" training.

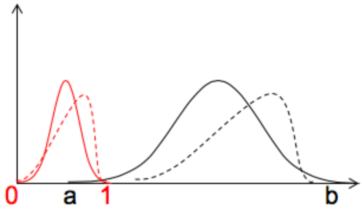
Feature scaling / Standardisation: Put all variables into the "similar" range, the variables are equally weight.

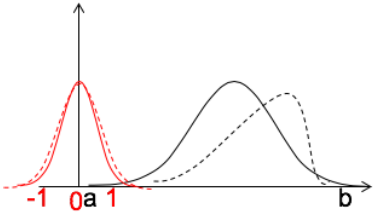
Two famous methods:

- Min-Max Normalisation (Rescaling): $M(X_{ij}) = \frac{X_{ij} - X_{\text{min,j}}}{X_{\text{max,j}} - X_{\text{min,j}}}$
- Standard Scaler / Standardisation: $M(X_{ij}) = \frac{X_{ij} - \overline{X}_j}{s_{X,j}}$

Feature Scaling (cont)

Diagrams to illustrate the effect of min-max scaling and standardisation are





The former transforms to the range [0, 1] while the later transforms to a "standard normal distributed" range.

Feature Scaling Example

Given the training data and testing data in Example of kNN Classifier, apply the standardisation to perform and use $k = 3$-NN method (with Euclidean distance) to predict which group the following students will be grouped into, based on **cut-off of 0.7** on group A.

WARNING

In Final Exam / Assessment, if the question did not mention “feature scaling” you DON’T NEED TO SCALE the features.

In real-world situation, you want perform feature scaling a lot of the times using computer.

Feature Scaling Example (cont)

Student	Weight (kg)	Height	Norm_Wgt	Norm_Hgt	Group
A	29	118	-0.6113	-0.4587	A
B	53	137	1.5284	1.3355	B
C	38	127	0.1910	0.3912	B
D	49	135	1.1717	1.1467	B
E	28	111	-0.7005	-1.1197	A
F	24	111	-1.0571	-1.1197	A
G	30	121	-0.5222	-0.1754	A
Mean	35.8571	122.8571			
Std. dev	11.2165	10.5898			

Feature Scaling Example (cont)

$k = 3$, cut-off = 0.7:

		H	I	J	K	L
	Group	Distance	Distance	Distance	Distance	Distance
A	A	0.5673	2.0205	0.6854	0.8079	1.7091
B	B	2.2699	0.7792	3.4575	2.1628	1.9637
C	B	0.7131	0.8869	1.8218	0.7554	1.0544
D	B	1.8879	0.4177	3.0596	1.8013	1.6076
E	A	1.0544	2.5370	0.6548	1.1686	2.3759
F	A	1.2977	2.7878	0.4177	1.4590	2.4419
G	A	0.4557	1.7857	0.9109	0.7378	1.4193
$P(Y = A)$		0.6667	0.0000	1.0000	0.6667	0.3333
Cut-off = 0.5	$\hat{y}$	A	B	A	A	B
Cut-off = 0.7	$\hat{y}$	B	B	A	B	B

Weighted kNN

class::knn $\rightarrow$ kNN classifier with Euclidean distance

FNN::knn.reg $\rightarrow$ kNN regressor with Euclidean distance

kknn::kknn $\rightarrow$ Weighted kNN (wkNN)

1. Let $N(\boldsymbol{x}, k+1)$ be the $k+1$ nearest neighbours to $\boldsymbol{x}$ according to a distance function $d(\boldsymbol{x}, \boldsymbol{x}_i)$.
2. The $(k+1)$th neighbour is used for "standardisation" of the k smallest distances via $D_i = \frac{d(\boldsymbol{x},\boldsymbol{x}_i)}{d(\boldsymbol{x},\boldsymbol{x}_{k+1})}$.
3. A weighted majority of the k nearest neighbour

$$\hat{y} = \max_j \left\{ \sum_{i=1}^{k} K(D_i) I(y_i = j) \right\}.$$

Weighted kNN (cont)

K is called the *kernel (function)* if it satisfies

1. $K(x) \geq 0$ for all $x \in \mathbb{R}$;
2. $K(x)$ is maximum when $x = 0$;
3. $K(x)$ descents monotonously when $x \to \pm\infty$.

A *rectangular kernel*, $K(x) = \frac{1}{2} I(|x| \leq 1)$; A *triangular kernel*, $K(x) = (1 - |x|) \cdot I(|x| \leq 1)$.

wkNN = kNN when kernel=rectangular (R's kknn) or weights=uniform (Python)