Nearest Neighbour Algorithms

Christos Dimitrakakis

September 23, 2025

Outline

Supervised learning

- Given labelled training examples $(x_1, y_1), \dots (x_T, y_T)$ where
- $x_t \in X$ are features
- $y_t \in Y$ are labels..

Feature space $\mathcal{X}$

- Usually $\mathcal{X} = \mathbb{R}^n$: the n-dimensional Euclidean space
- How do we use your class data?

Classification

- $Y = \{1, \dots, m\}$ are discrete labels

Regression

- $Y = \mathbb{R}^m$ are continuous values

The kNN algorithm idea

- Assume an unknown example is similar to its neighbours
- Smoothness allows us to make predictions

Discriminatory analysis-nonparametric discrimination: consistency properties, Evelyn Fix and Joseph L. Hodges Jr, 1951.

Figure: Evelyn Fix

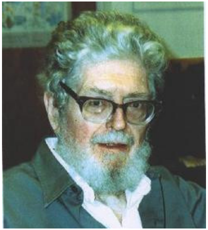
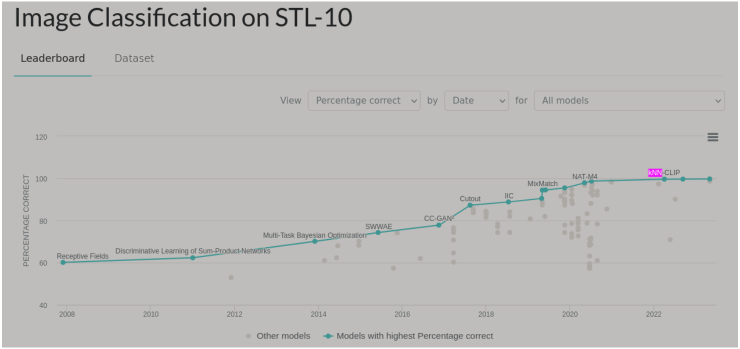


Figure: Joseph Hodges

Performance of KNN on image classification



- Really simple!
- Can outperform really complex models!

The Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d

The Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d
- $t^* = \arg\min_t d(x_t, x)$ / How do we implement this?

The Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d
- $t^* = \arg\min_t d(x_t, x)$ / How do we implement this?
- Return $\hat{y}_t = y_{t^*}$

The Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^T$, test point x, distance d
- $t^* = \arg\min_t d(x_t, x)$ / How do we implement this?
- Return $\hat{y}_t = y_{t^*}$

The Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d
- $t^* = \arg\min_t d(x_t, x)$ / How do we implement this?
- Return $\hat{y}_t = y_{t^*}$

Classification

$\hat{y}_t \in [m] \equiv \{1, \ldots, m\}$

The Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d
- $t^* = \arg\min_t d(x_t, x)$ / How do we implement this?
- Return $\hat{y}_t = y_{t^*}$

Classification

$\hat{y}_t \in [m] \equiv \{1, \ldots, m\}$

Regression

$\hat{y}_t \in \mathbb{R}^m$

The k-Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d, neighbours k

The k-Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d, neighbours k
- Calculate $h_t = d(x_t, x)$ for all t.

The k-Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d, neighbours k
- Calculate $h_t = d(x_t, x)$ for all t.
- Get sorted indices $s = \texttt{argsort}(h)$ so that $d(x_{s_i}, x) \leq d(x_{s_{i+1}}, x)$ for all i.

The k-Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d, neighbours k
- Calculate $h_t = d(x_t, x)$ for all t.
- Get sorted indices $s = \texttt{argsort}(h)$ so that $d(x_{s_i}, x) \leq d(x_{s_{i+1}}, x)$ for all i.
- Return $\sum_{i=1}^{k} y_{s_i} / k$.

The k-Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^{T}$, test point x, distance d, neighbours k
- Calculate $h_t = d(x_t, x)$ for all t.
- Get sorted indices $s = \texttt{argsort}(h)$ so that $d(x_{s_i}, x) \leq d(x_{s_{i+1}}, x)$ for all i.
- Return $\sum_{i=1}^{k} y_{s_i}/k$.

The k-Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^T$, test point x, distance d, neighbours k
- Calculate $h_t = d(x_t, x)$ for all t.
- Get sorted indices $s = \mathtt{argsort}(h)$ so that $d(x_{s_i}, x) \leq d(x_{s_{i+1}}, x)$ for all i.
- Return $\sum_{i=1}^k y_{s_i}/k$.

Classification

- We use a one-hot encoding $(0, \ldots, 0, 1, 0, \ldots, 0)$, with $y_t \in \{0, 1\}^m$.
- The class of the t-th example is $j \Leftrightarrow y_{t,j} = 1$.
- Equivalently, return p with

$$p_i = \sum_{t=1}^{k} \mathbb{I}\left\{y_{s_t} = i\right\}/k$$

The k-Nearest Neighbour algorithm

Pseudocode

- Input: Data $(x_t, y_t)_{t=1}^T$, test point x, distance d, neighbours k
- Calculate $h_t = d(x_t, x)$ for all t.
- Get sorted indices $s = \texttt{argsort}(h)$ so that $d(x_{s_i}, x) \leq d(x_{s_{i+1}}, x)$ for all i.
- Return $\sum_{i=1}^k y_{s_i}/k$.

Classification

- We use a one-hot encoding $(0, \dots, 0, 1, 0, \dots, 0)$, with $y_t \in \{0, 1\}^m$.
- The class of the t-th example is $j \Leftrightarrow y_{t,j} = 1$.
- Equivalently, return p with

$$p_i = \sum_{t=1}^k \mathbb{I}\left\{y_{s_t} = i\right\}/k$$

Regression

- $y_t \in \mathbb{R}^m$, so we need do nothing

The number of neighbours

$k = 1$

- How does it perform on the training data?
- How might it perform on unseen data?

$k = T$

- How does it perform on the training data?
- How might it perform on unseen data?

Distance function

For data in $\mathbb{R}^n$, p-norm

$$d(x, y) = \|x - y\|_p$$

Scaled norms

When features having varying scales:

$$d(x, y) = \|Sx - Sy\|_p$$

Or pre-scale the data

Complex data

- Manifold distances
- Graph distance

Distances

A distance $d(\cdot, \cdot)$:

- Identity $d(x, x) = 0$.
- Positivity $d(x, y) > 0$ if $x \neq y$.
- Symmetry $d(y, x) = d(x, y)$.
- Triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$.

For data in $\mathbb{R}^n$, \$p\$-norm

$$d(x, y) = \|x - y\|_p$$

Norms;

A norm $\|\cdot\|$

- Zero element $\|0\| = 0$.
- Homogeneity $\|cx\| = c\|x\|$ for any scalar a.
- Triangle inequality $\|x+y\| \leq \|x\| + \|y\|$.

\$p\$-norm

$$\|z\|_p = \left(\sum_i z_i^p\right)^{1/p}$$

Neighbourhood calculation

If we have T datapoints

Sort and top K.

- Requires $O(T \ln T)$ time

Use the Cover-Tree or KD-Tree algorithm

- Requires $O(cK \ln T)$ time.
- c depends on the data distribution.

Making a decision

kNN as a model

- Given features x, we get a vector p of class probabilities:

$$p_i = P(y = i|x),$$

where $P(y = i|x)$ is the probability that y is i, given x.

Making a decision

kNN as a model

- Given features x, we get a vector p of class probabilities:

$$p_i = P(y = i|x),$$

where $P(y = i|x)$ is the probability that y is i, given x.

Decisions to maximise accuracy

At time t:

- We observe features x_t

Making a decision

kNN as a model

- Given features x, we get a vector p of class probabilities:

$$p_i = P(y = i|x),$$

where $P(y = i|x)$ is the probability that y is i, given x.

Decisions to maximise accuracy

At time t:

- We observe features x_t
- We predict label $a_t = \arg\max_i P(y_t = i|x_t)$

Making a decision

kNN as a model

- Given features x, we get a vector p of class probabilities:

$$p_i = P(y = i|x),$$

where $P(y = i|x)$ is the probability that y is i, given x.

Decisions to maximise accuracy

At time t:

- We observe features x_t
- We predict label $a_t = \arg\max_i P(y_t = i|x_t)$
- We observe the actual label y_t.

Making a decision

kNN as a model

- Given features x, we get a vector p of class probabilities:

$$p_i = P(y = i|x),$$

where $P(y = i|x)$ is the probability that y is i, given x.

Decisions to maximise accuracy

At time t:

- We observe features x_t
- We predict label $a_t = \arg\max_i P(y_t = i|x_t)$
- We observe the actual label y_t.
- We win if $y_t = a_t$ and lose otherwise

Making a decision

kNN as a model

- Given features x, we get a vector p of class probabilities:

$$p_i = P(y = i|x),$$

where $P(y = i|x)$ is the probability that y is i, given x.

Decisions to maximise accuracy

At time t:

- We observe features x_t
- We predict label $a_t = \arg\max_i P(y_t = i|x_t)$
- We observe the actual label y_t.
- We win if $y_t = a_t$ and lose otherwise

Making a decision

kNN as a model

- Given features x, we get a vector p of class probabilities:

$$p_i = P(y = i|x),$$

where $P(y = i|x)$ is the probability that y is i, given x.

Decisions to maximise accuracy

At time t:

- We observe features x_t
- We predict label $a_t = \arg\max_i P(y_t = i|x_t)$
- We observe the actual label y_t.
- We win if $y_t = a_t$ and lose otherwise

The model versus the prediction

- The model P tells us the probability of different classes.
- When we decide what our prediction should be, we can use the model.
- We will use π to denote the decision rule or policy.

Decisions versus predictions

- We frequently need to make a decision, instead of just a prediction.
- Our utility function $U(y, a)$ represents our preferences.
- The space of actions A is *not identical* to the set of labels Y.

Decisions versus predictions

- We frequently need to make a decision, instead of just a prediction.
- Our utility function $U(y, a)$ represents our preferences.
- The space of actions A is *not identical* to the set of labels Y.

Minimise spam annoyance

What utility function would you use for the spam detection problem?

Utility	Pass	Flag	Trash
Normal			
Spam			
Virus			

Decisions versus predictions

- We frequently need to make a decision, instead of just a prediction.
- Our utility function $U(y, a)$ represents our preferences.
- The space of actions A is *not identical* to the set of labels Y.

Minimise spam annoyance

What utility function would you use for the spam detection problem?

Utility	Pass	Flag	Trash
Normal			
Spam			
Virus			

Classification decision to maximise expected utility

- Expected utility of a single decision

$$\mathbb{E}[U|a, x] = \sum_y P(y|x, a) U(y, a) = \sum_y P(y|x) U(y, a)$$

- The decision maximising expected utility

$$a^* = \arg\max_a \mathbb{E}[U|a, x]$$

KNN activity

- Implement nearest neighbours
- Introduction to scikitlearn nearest neighbours

Homework: Measure performance

In this exercise, you will measure utility on a test set, and select actions that potentially maximise utility in expectation:

Measure utility

Create a function called *utilityScore(y, actions, U)*.
This takes as input the actual labels y_t, and actions a_t (e.g. predicted labels) of a classifier. It then returns the average utility:
$\sum_{t=1}^{T} U(a_t, y_t)/T$.

Calculate utility scores

Calculate the $utility_{score}$ of a basic kNN classifier for various values of k.

Return highest-utility actions

Create a function *predictUtil(clf, X, U)* that takes a classifier *clf*, a dataset of features X and a utility function U as input. It calls $clf.predict\_{proba}()$ and returns a list of actions, one for each row of X.

Verification

Verify that using $\text{predict}_{\text{util}}()$ givs you a higher utilityScore() than simply using predict()