Applied Machine Learning

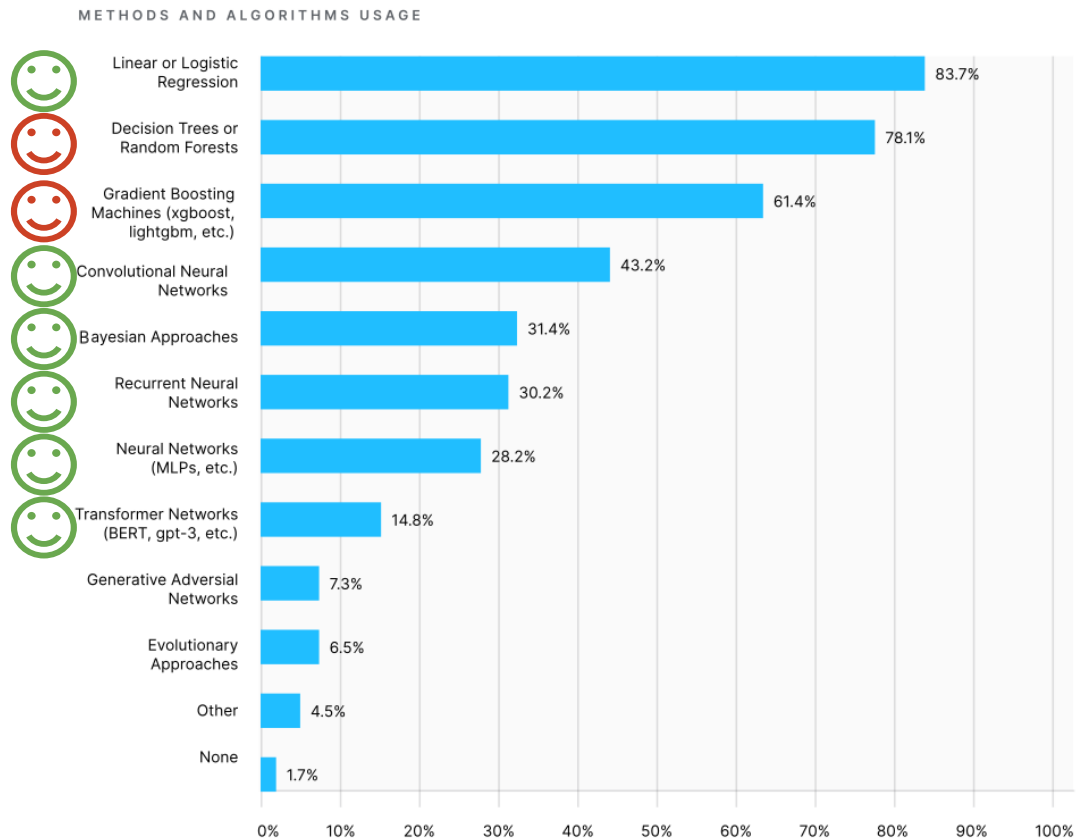
Nearest Neighbours

Isabeau Prémont-Schwarz

Motivation

What we have left to cover for this course:

- Nearest neighbours
- Classification and regression trees
- Linear support vector machines
- Bagging & boosting
- Unsupervised learning
- Dimensionality reduction



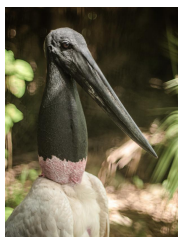
from 2020 Kaggle's survey on the state of Machine Learning and Data Science, you can read the full version here

Objectives

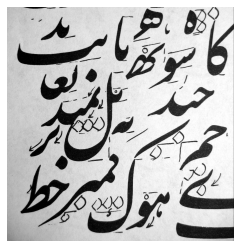
- variations of k-nearest neighbors for
  - classification
  - regression
- computational complexity
- some pros and cons of K-NN
- what is a hyper-parameter?

Classifying by Similarity

We guess type of unseen instances based on their similarity to our past experience
Let's give this a try:



is this a kind of
(a) stork
(b) pigeon
(c) penguin



is this calligraphy from
(a) east Asia
(b) Africa
(c) middle east

Accretropin**:** is it
(a) an east European actor
(b) drug
(c) gum brand

The Price Is Right
1972 · Game Show · 48 seasons

example of nearest neighbor regression
pricing based on similar items
(e.g., used in the housing market)

Nearest neighbour classifier

training: do nothing and only record the data (a **lazy learner**, also a **non-parametric** model)

inference: predict the label by finding the most similar example in training set

$$\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$$

pairs of input vector and corresponding target or label

$\mathcal{D}$: training set

x : *D*-dimensional vector

y : a categorical or nominal variable

N : number of training instances

n : index of training instance ($n \in \{1 \dots N\}$)

indexes can be placed up or down based on the notation in use, or droped all together. When up, not to be confused with a power

	<tumorsize, texture, perimeter>	,	<cancer>	
$x^{(1)}$	<18.2, 27.6, 117.5>	,	< No >	$y^{(1)}$
$x^{(2)}$	<17.9, 10.3, 122.8>	,	< No >	$y^{(2)}$
$x^{(3)}$	<20.2, 14.3, 111.2>	,	< Yes >	$y^{(3)}$
⋮	⋮			⋮
$x^{(N)}$	<15.5, 15.2, 135.5>	,	< No >	$y^{(N)}$

Nearest neighbour classifier

training: do nothing and only record the data (a **lazy learner**, also a **non-parametric** model)

inference: predict the label by finding the most similar example in training set

we need a measure of distance/similarity

e.g., Euclidean distance

$$||x - x'||_2 = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$$

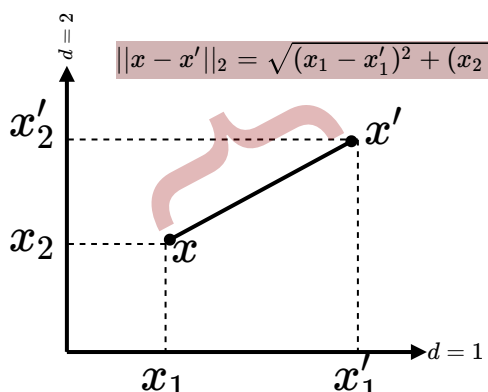
indexes the features in an instance

$$x^* = \underset{x^{(i)} \in train.set}{\arg\min} \; distance(x^{(i)}, x)$$

$$\hat{y} = y^*$$

assume each instance (represented by a vector) is a point in a D-dimensional space, the Euclidean distance is the length of a line segment between any two points

e.g. in 2D we have:



	<tumorsize,	texture,	perimeter>	,	
$x^{(1)}$	<cancer>				
$x^{(2)}$	<18.2,	27.6,	117.5>	,	< No >
$x^{(3)}$	<17.9,	10.3,	122.8>	,	< No >
$x^{(4)}$	<20.2,	14.3,	111.2>	,	< Yes >
	<15.5,	15.2,	135.5>	,	< No >

<16.5,	10.1,	121.2>

Nearest neighbour classifier

training: do nothing and only record the data (a **lazy learner**, also a **non-parametric** model)

inference: predict the label by finding the most similar example in training set

need a measure of distance/similarity (e.g., a metric)

examples

for real-valued feature-vectors

Euclidean distance $\mathsf{D}_{\text{Euclidean}}(x, x') = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$

Manhattan distance $\mathsf{D}_{\text{Manhattan}}(x, x') = \sum_{d=1}^{D}|x_d - x'_d|$

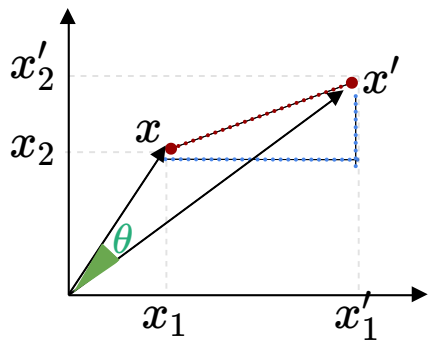
Minkowski distance $\mathsf{D}_{\text{Minkowski}}(x, x') = \left(\sum_{d=1}^{D}|x_d - x'_d|^p\right)^{\frac{1}{p}}$

Cosine similarity $\mathsf{D}_{\text{Cosine}}(x, x') = x^\top x' / ||x||||x'||$

for discrete feature-vectors (e.g. smoker?)

Hamming distance $\mathsf{D}_{\text{Hamming}}(x, x') = \sum_{d=1}^{D}\mathbb{I}(x_d \neq x'_d)$

... and there are metrics for strings, distributions etc.



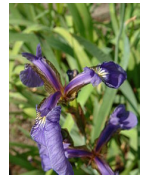
$||x|| = \sqrt{\sum_{d=1}^{D} x_d^2}$

$x^\top$ denotes transpose of x

$x^\top x'$ denotes the matrix product of $x^\top$ and x' which is equal to the dot product of x and x'

$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}$

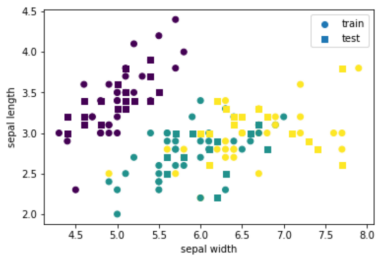
indicator function



Iris dataset

N = 150 instances of flowers
D=4 features {the length and the width of the sepals and petals}
C=3 classes {setosa, versicolor, virginica} : 50 samples of each

one of the most famous datasets in statistics



for better visualization, we use only two features

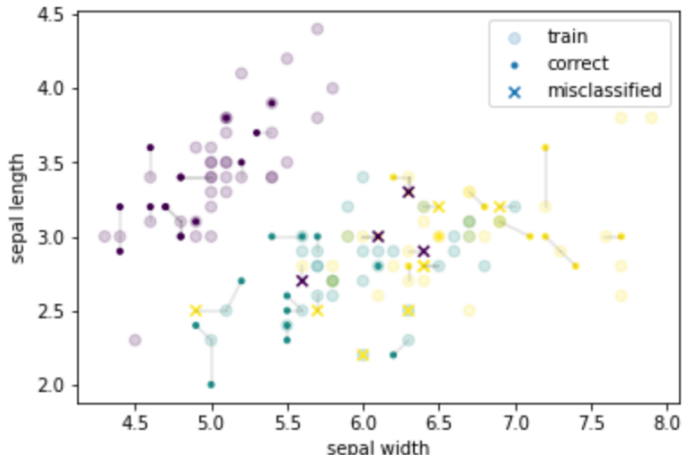
input $x^{(n)} \in \mathbb{R}^2$

label $y^{(n)} \in \{1, 2, 3\}$

$n \in \{1, \ldots, N\}$

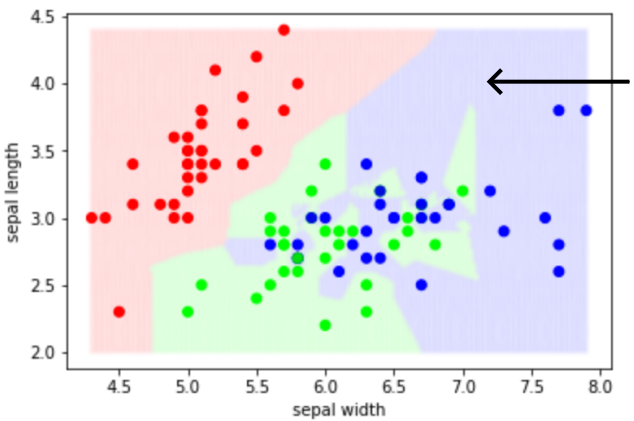
indexes the training instance
sometime we drop (n)

using Euclidean distance nearest neighbor classifier gets 68% accuracy (correct/total) in classifying the test instances



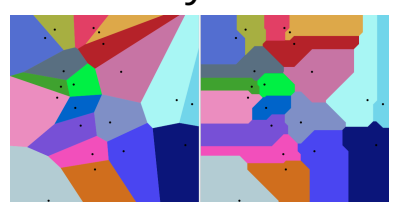
Decision boundary

a classifier defines a decision boundary in the input space



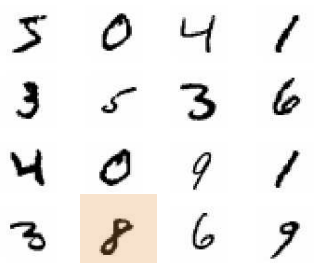
all points in this region will have the same class

the **Voronoi diagram** visualizes the decision boundary of nearest neighbor classifier: each color shows all points closer to the corresponding training instance than to any other instance



images from wiki

Higher dimensions: digits dataset



input $x^{(n)} \in \{0, \dots, 255\}$ 28×28 size of the input image in pixels

8-bit grayscale, see wiki

label $y^{(n)} \in \{0, \dots, 9\}$

$n \in \{1, \dots, N\}$ indexes the training instance

sometime we drop (n)

Classic example of handwritten digit recognition **MNIST**

[60K train, 10K test, 28x28, centered]

see wiki, this, and a fun watch

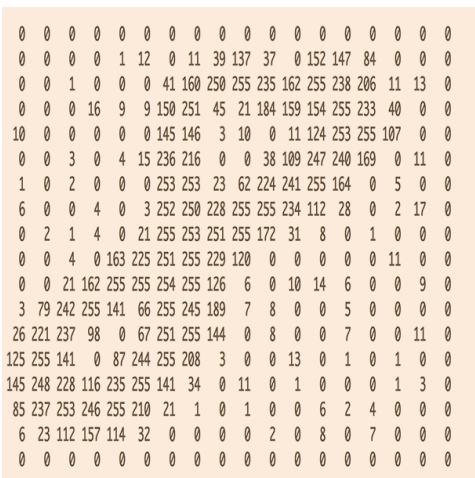
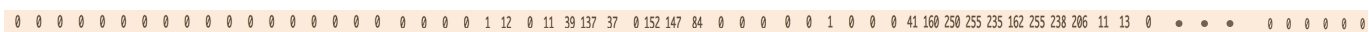


image from here

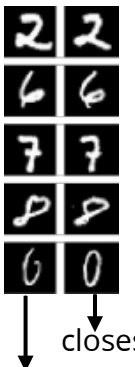
vectorization:

$x \to \text{vec}(x) \in \mathbb{R}^{784}$ input dimension **D**

assume intensities are real numbers



K - Nearest Neighbor (K-NN) classifier



closest instances

new test instances

training: do nothing

test: find the nearest image in the training set

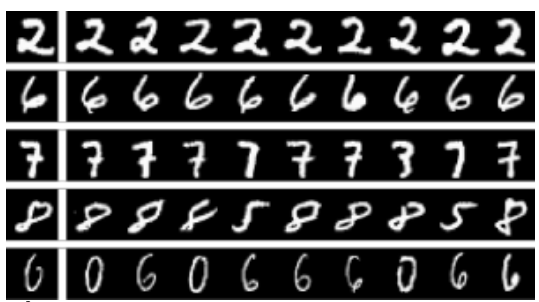
we are using Euclidean distance in a 784-dimensional space to find the closest neighbour

can we make the predictions more robust?

consider **K**-nearest neighbors and label by the majority

we can even estimate the **probability** of each class

$$p(y^{new} = c \mid x_{new}) = \frac{1}{K} \sum_{x^{(k)} \in \mathrm{KNN}(x^{new})} \mathbb{I}(y^{(k)} = c)$$



9 closest instances in the train set per new test instance

new test instances

$p(y = 6|$  $) = \frac{6}{9}$

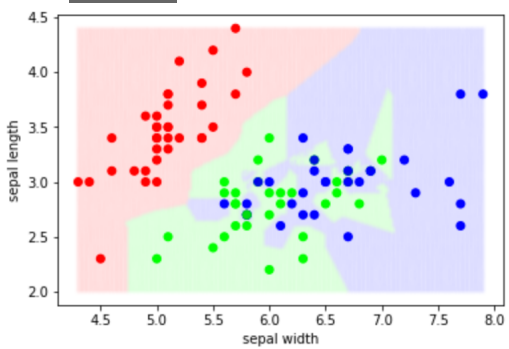
$p(y = 0|$ $) = ?$

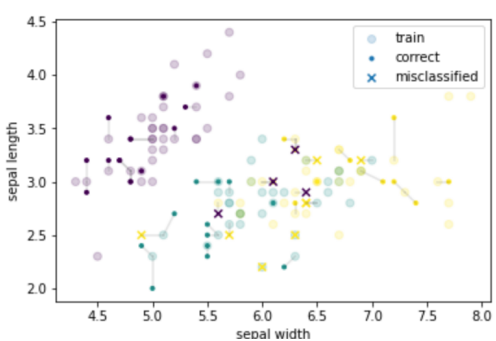
Choice of K

K is a **hyper-parameter** of our model

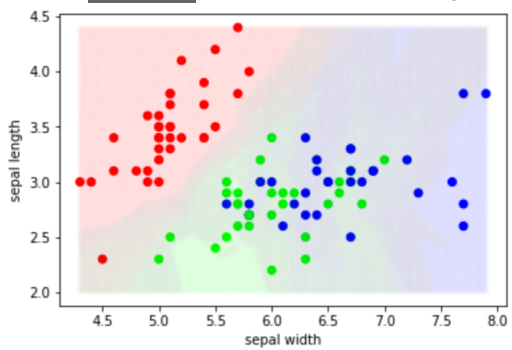
in contrast to parameters, the hyper-parameters are not learned during the usual training procedure

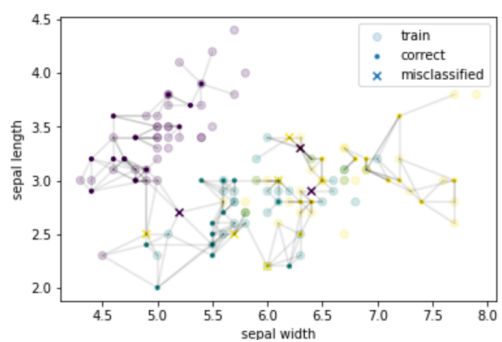
$K = 1$ 76% accuracy



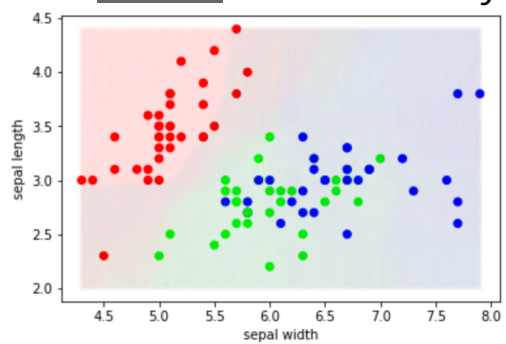


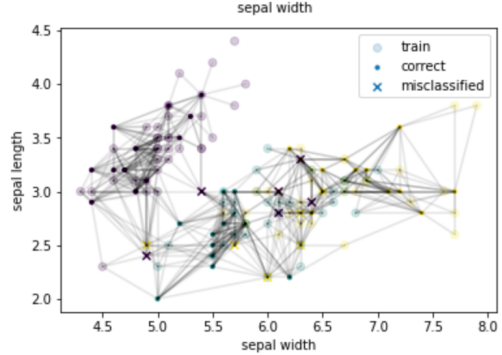
$K = 5$ 84% accuracy





$K = 15$ 78% accuracy



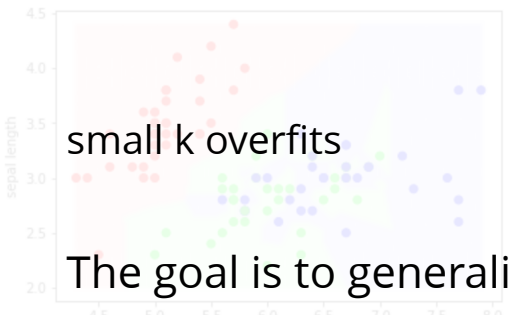


Choice of K

K is a **hyper-parameter** of our model

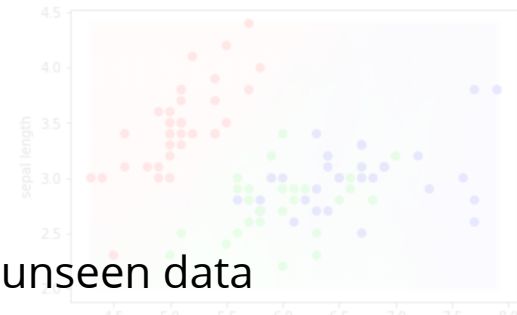
in contrast to parameters, the hyper-parameters are not learned during the usual training procedure

$K = 1$ 76% accuracy

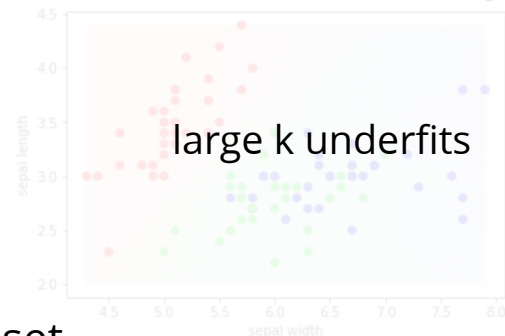


small k overfits

$K = 5$ 84% accuracy



$K = 15$ 78% accuracy



large k underfits

The goal is to generalize to unseen data

We can approximate generalization error based on the test set

We tune the hyperparameters on a validation set instead of test set, test set is only used at the end for final evaluation

Computational complexity

$$\sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$$

the **computational complexity** for a single test query: $\mathcal{O}(ND + NK)$

for each point in the training set calculate the distance in $\mathcal{O}(D)$ for a total of $\mathcal{O}(ND)$

find the K points with smallest of distances in $\mathcal{O}(NK)$

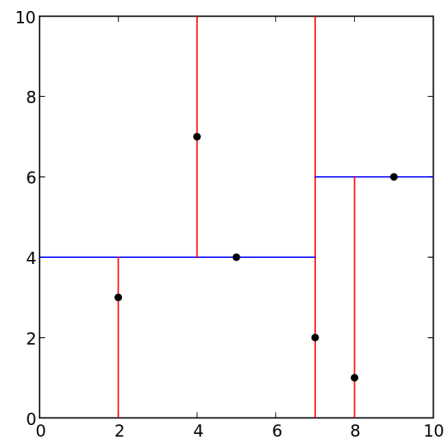
bonus

in practice efficient implementations using KD-tree (and ball-tree) exist

partition the space based on a tree structure

for a query point only search the relevant part of the space

[see here for more information]



from wiki

Scaling and importance of features

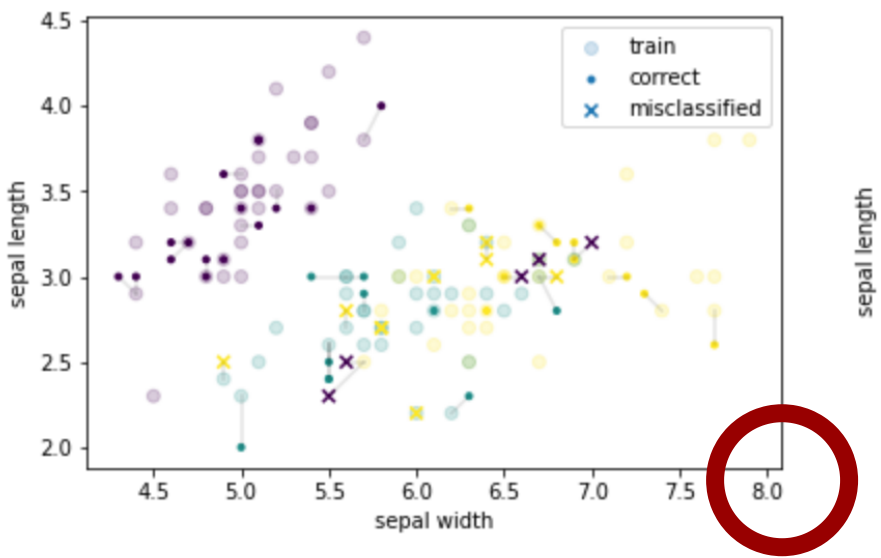
$$\sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$$

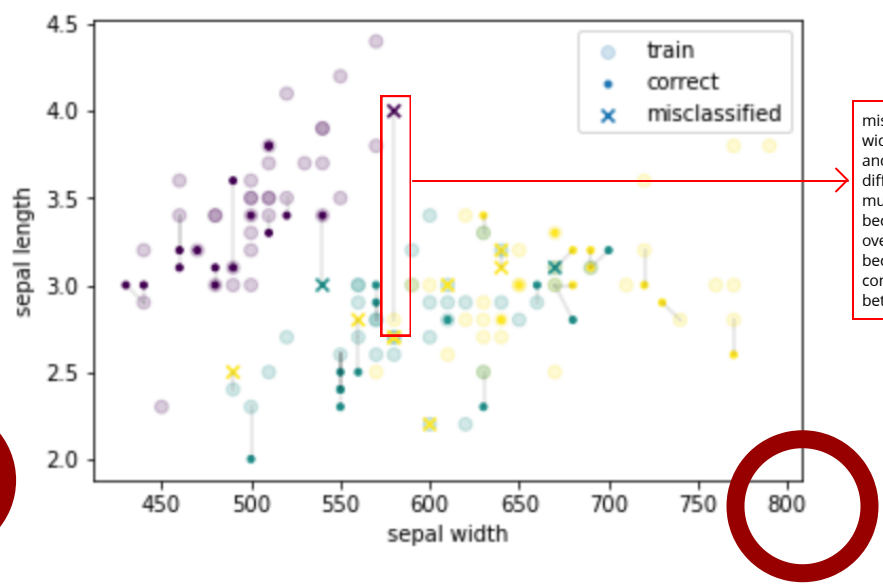
scaling of features affects distances and nearest neighbours

example

feature sepal width is scaled **x100**

closeness in this dimension becomes more important in finding the nearest neighbor





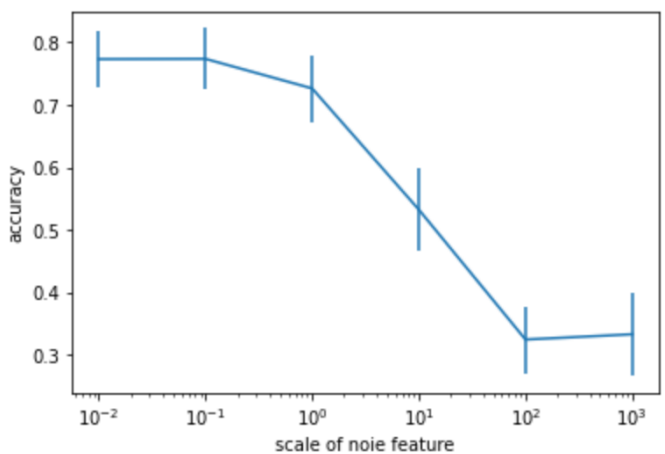
misclassified since the sepal width is almost the same and the sepal length difference is ignored due to much mower scale, i.e. it becomes negligible in the overall distance, and width becomes the main factor contributing to distance between any two points

Scaling and importance of features

we want **important features** to maximally affect the classification:
they should have **larger scale**

noisy and irrelevant features should have a small scale

K-NN is not adaptive to feature scaling and it is sensitive to noisy features ☹



example

add a feature that is random noise to previous example

plot the effect of the scale of noise feature on accuracy

K-NN regression

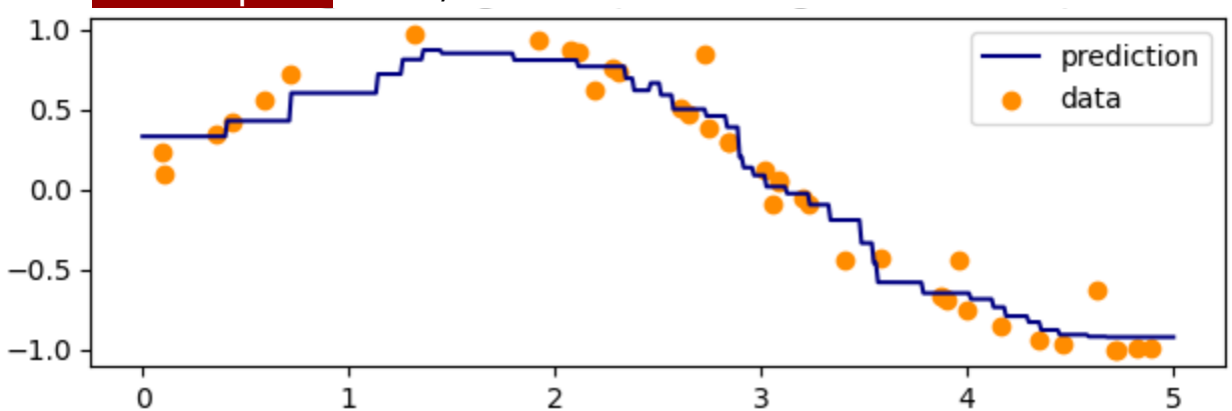
so far our task was **classification**

- use *majority vote* of neighbors for prediction at test time

the change for **regression** is minimal

- use the *mean (or median)* of K nearest neighbors' targets

example D=1, K=5

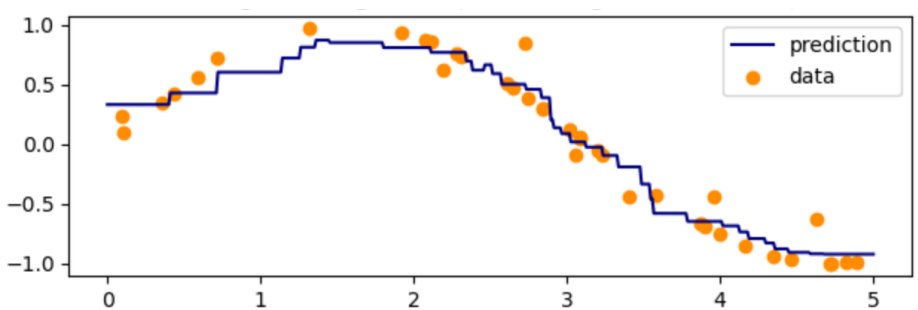


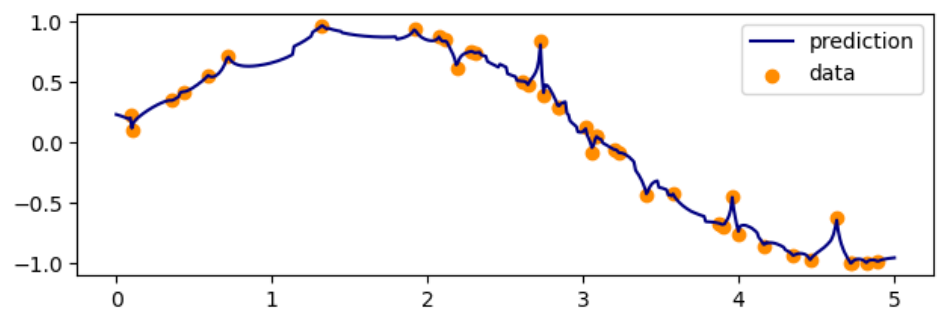
example from scikit-learn.org, see here

Some variations

in **weighted K-NN** the neighbors are weighted inversely proportional to their distance

- for classification the votes are weighted
- for regression calculate the weighted average





example from scikit-learn.org

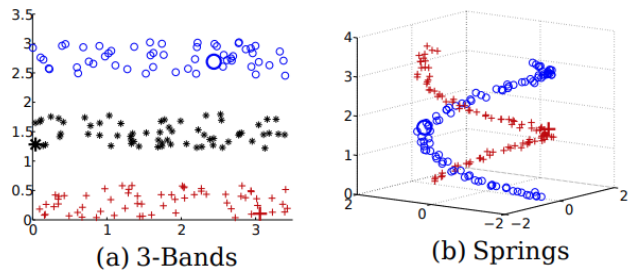
in **fixed radius nearest neighbors** all neighbors in a fixed radius are considered

in dense neighbourhoods we get more neighbors

K-NN for unsuprevised and semi-supervised learning

Semi-supervided setting:

Propagate labels to nearby points



see the learning from labeled and unlabeled data with label propagation

Unsupervised setting:

Partition the k-NN graphs to cluster the data

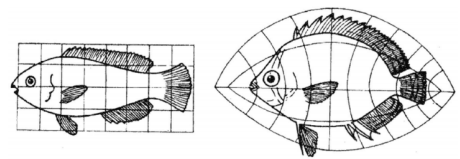
connects each point to its k-nearest neighbor in the [training] data

read more on KNN here, and on clustering with Knn here

Summary

K-NN performs classification/regression by finding similar instances in training set

- need a notion of distance, performance improves a lot with a better similarity measure e.g. see here
- how many neighbors to consider (fixed K, or fixed radius)
- how to weight the neighbors



K-NN is a ***non-parametric*** method and a ***lazy*** **learner**

- non-parametric: our model has no parameters (in fact the training data points are model parameters)
- Lazy, because we don't do anything during the training
  - test-time complexity grows with the size of the data, as well as space complexity (store all data)
  - good performance when we have lots of data, see here

K-NN is sensitive to feature scaling and noise