





# Applied Machine Learning

Nearest Neighbours

Reihaneh Rabbany



# Admin

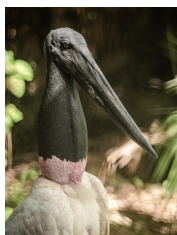
- First tutorial is out
  - Tutorials are pre-recorded
  - Go to the TA's office hour after you watched the tutorial for questions
- Remember to do the two quizzes
  - second one is going to be released today  
  - make sure to try them before the add/drop as these test the prerequisites needed for this course
- Join a study group  
- Any questions?

# Objectives

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

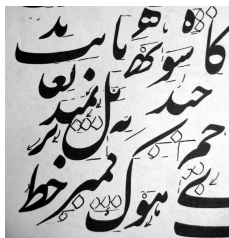
# Classifying by Similarity

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



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

The Price Is Right  
1972 · Game Show · 48 seasons



# 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

$\mathcal{D}$  : training set

$\boldsymbol{x}$  :  $D$ -dimensional vector

$\boldsymbol{y}$  : 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 dropped all together.  
When up, not to be confused with a power

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

pairs of input vector  
and corresponding  
target or label

	<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 >	$y^{(3)}$
$\vdots$				$\vdots$
$x^{(N)}$	<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

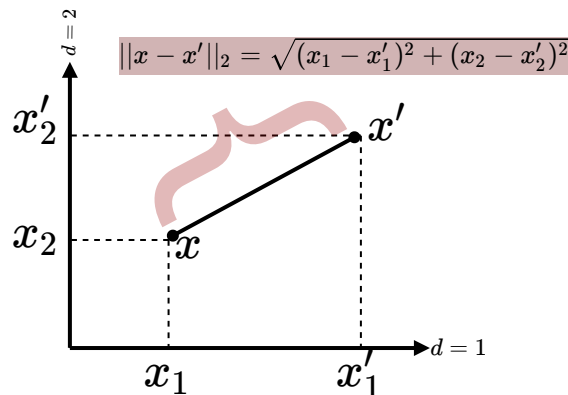
e.g., Euclidean distance

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

indexes the features in an instance

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:



$$x^* = \arg \min_{x^{(i)} \in \text{train.set}} \text{distance}(x^{(i)}, x)$$
$$\hat{y} = y^*$$

$x^{(1)}$	<tumorsize, texture, perimeter>	,	
	<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  $D_{\text{Euclidean}}(x, x') = \sqrt{\sum_{d=1}^D (x_d - x'_d)^2}$

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

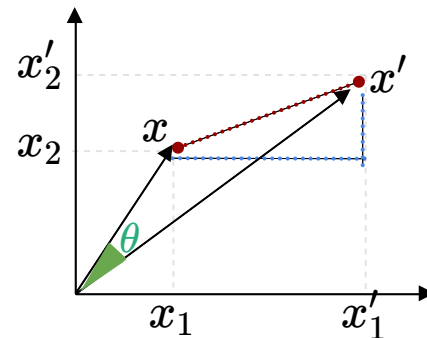
Minkowski distance  $D_{\text{Minkowski}}(x, x') = \left(\sum_{d=1}^D |x_d - x'_d|^p\right)^{\frac{1}{p}}$

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

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

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

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



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

$x^T$  denotes transpose of  $x$

$x^T x'$  denotes the matrix product of  $x^T$  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

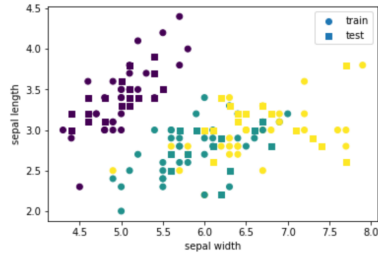
one of the most famous datasets in statistics



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



for better visualization, we use only two features

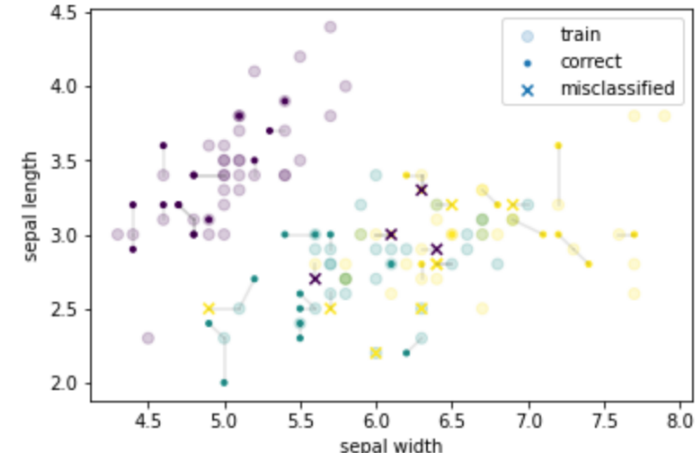
input  $x^{(n)} \in \mathbb{R}^2$

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

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

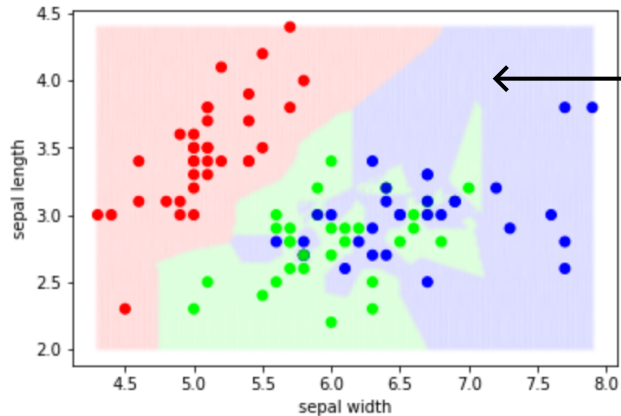
indexes the training instance  
sometime we drop (n)

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



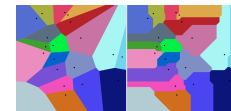
# Decision boundary

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



Euclidean v.s. Manhattan distance

# Higher dimensions: digits dataset

5 0 4 1  
 3 5 3 6  
 4 0 9 1  
 3 8 6 9

input  $x^{(n)} \in \{0, \dots, 255\}$   $28 \times 28$  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 ( $n$ )

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 12 0 0 11 39 137 37 0 152 147 84 0 0 0
0 0 1 0 0 0 41 160 250 255 235 162 255 238 206 11 13 0
0 0 0 16 9 9 150 251 45 21 184 159 154 255 233 40 0 0
10 0 0 0 0 0 145 146 3 10 0 11 124 253 255 107 0 0
0 0 3 0 4 15 236 216 0 0 38 109 247 240 169 0 11 0
1 0 2 0 0 0 253 253 23 62 224 241 255 164 0 5 0 0
6 0 0 4 0 0 3 252 250 228 255 255 234 112 28 0 2 17 0
0 2 1 4 0 0 21 255 253 251 255 172 31 8 0 1 0 0 0
0 0 4 0 163 225 251 255 229 120 0 0 0 0 0 11 0 0
0 0 21 162 255 255 254 255 126 6 0 10 14 6 0 0 9 0
3 79 242 255 141 66 255 245 189 7 8 0 0 5 0 0 0 0
26 221 237 98 0 67 251 255 144 0 8 0 0 7 0 0 11 0
125 255 141 0 87 244 255 208 3 0 0 13 0 1 0 1 0 0
145 248 228 116 235 255 141 34 0 11 0 1 0 0 0 1 3 0
85 237 253 246 255 210 21 1 0 1 0 0 6 2 4 0 0 0
6 23 112 157 114 32 0 0 0 0 0 2 0 8 0 7 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    
```

image from [here](#)

Classic example of  
 handwritten digit  
 recognition **MNIST**  
 [ 60K train, 10K test,  
 28x28, centered ]  
 see [wiki](#), [this](#), and a fun [watch](#)

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9
    
```

## vectorization:

$x \rightarrow \text{vec}(x) \in \mathbb{R}^{784}$  input dimension  $D$   
 assume intensities are real numbers

0 1 12 0 11 39 137 37 0 152 147 84 0 0 0 0 0 0 1 0 0 0 41 160 250 255 235 162 255 238 206 11 13 0

# K - Nearest Neighbor (K-NN) classifier

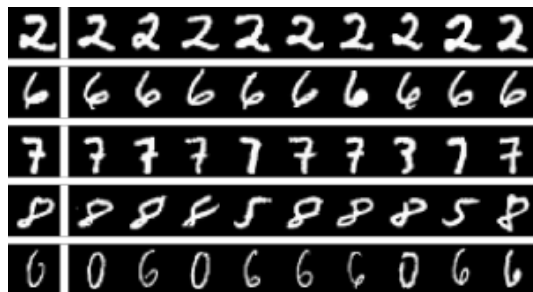


**training:** do nothing

**test:** find the nearest image in the training set

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

↓ closest instances  
new test instances



9 closest instances in the train set per new test instance

↓ new test instances

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

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

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

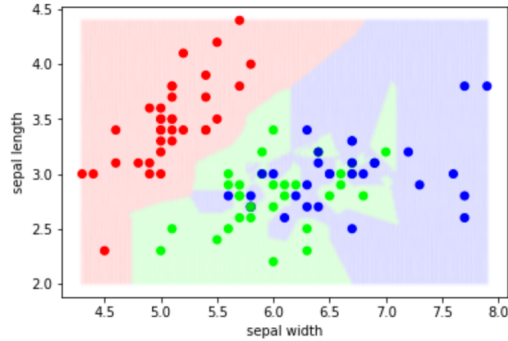
$$p(y = 0 | \text{6}) = ?$$

# Choice of K

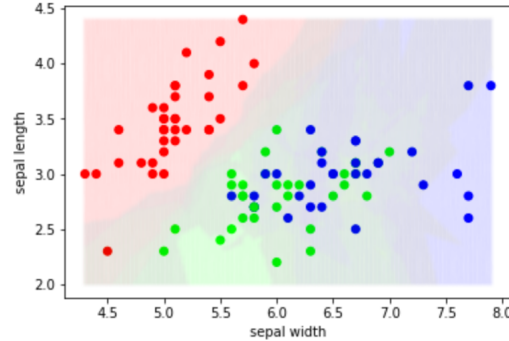
K is a **hyper-parameter** of our model

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

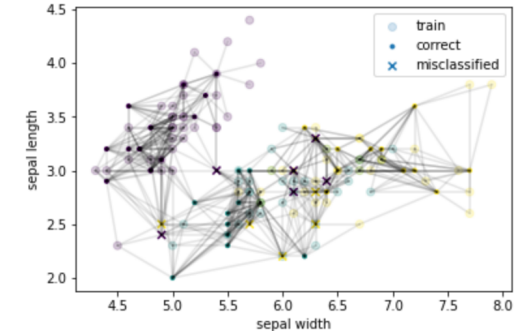
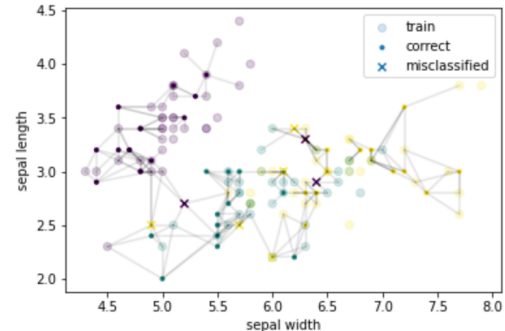
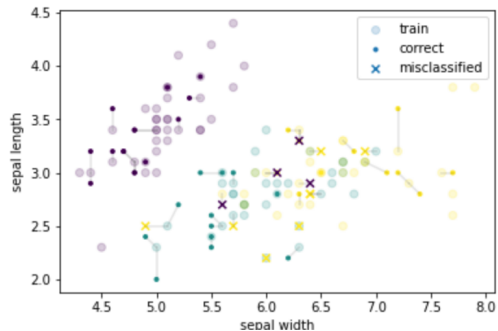
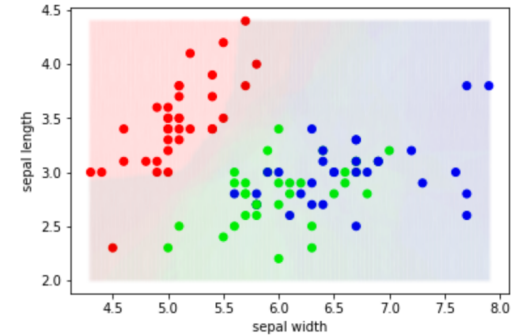
$K = 1$  76% accuracy



✓  $K = 5$  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

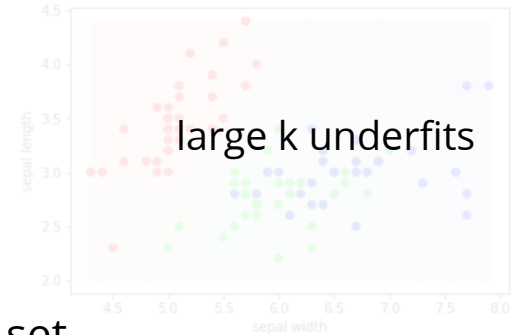
$K = 1$  76% accuracy



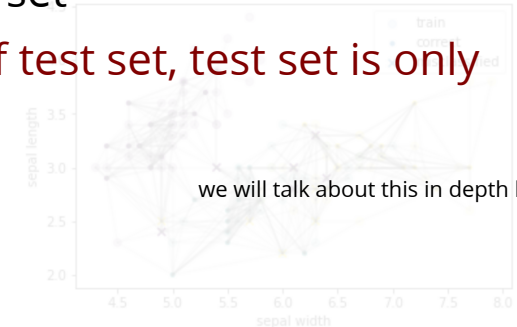
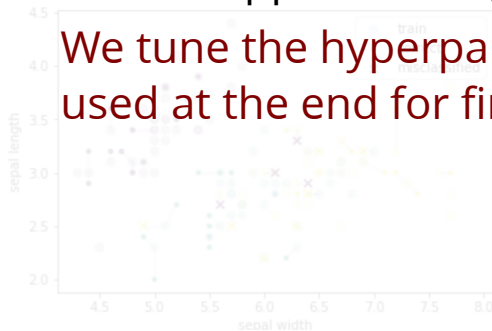
$K = 5$  84% accuracy



$K = 15$  78% accuracy



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  $\mathcal{O}(ND)$

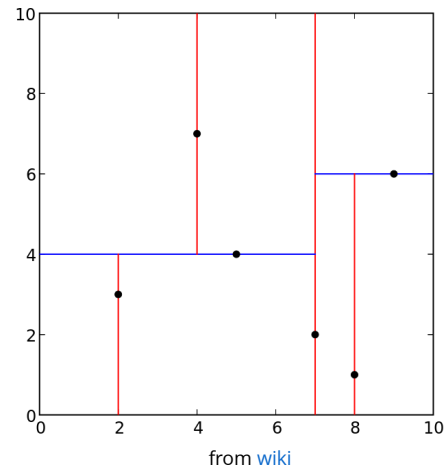
find the K points with smallest of distances in  $\mathcal{O}(NK)$

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



# Scaling and importance of features

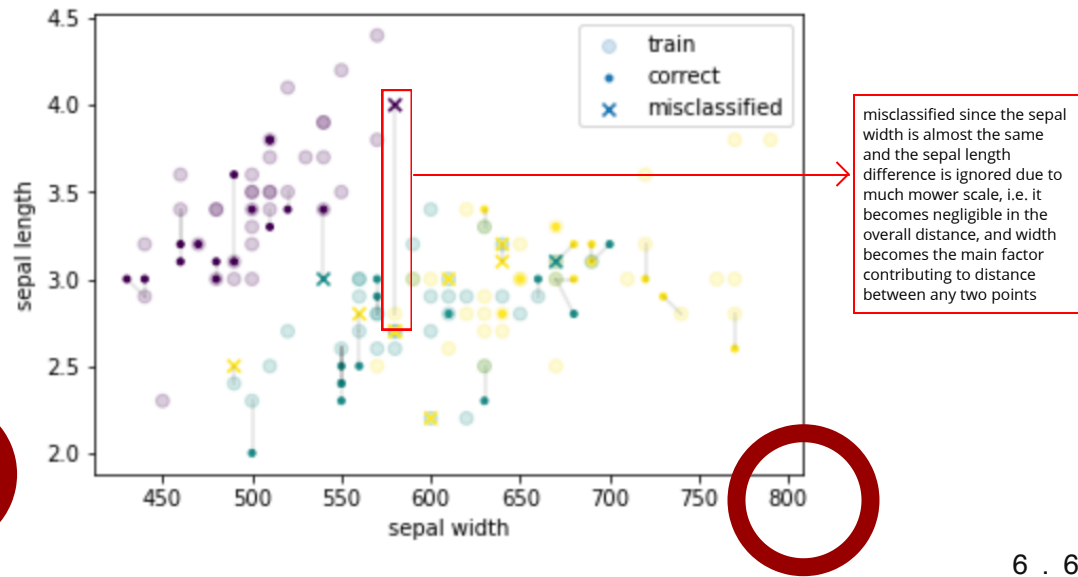
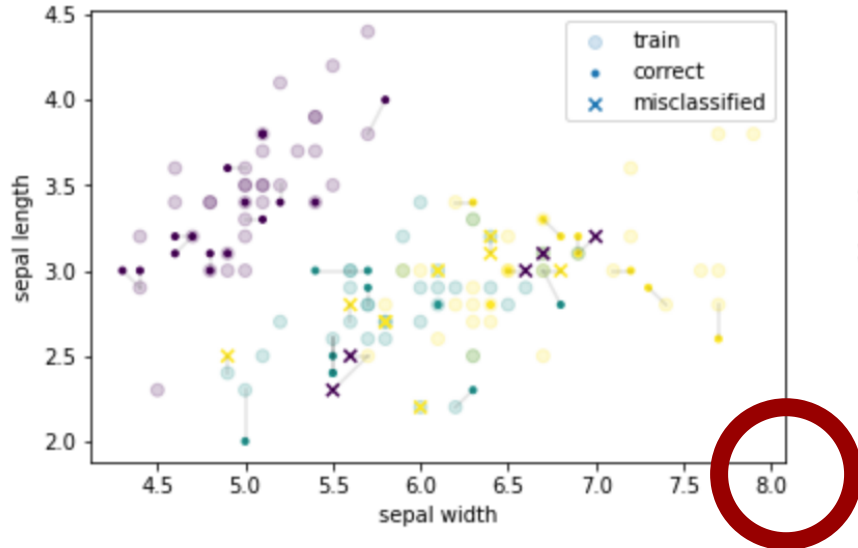
$$\sqrt{\sum_{d=1}^D (x_d - x'_d)^2}$$

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

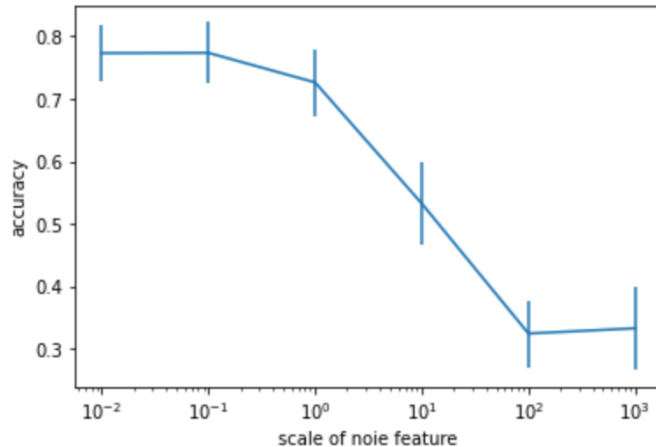


# Scaling and importance of features

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

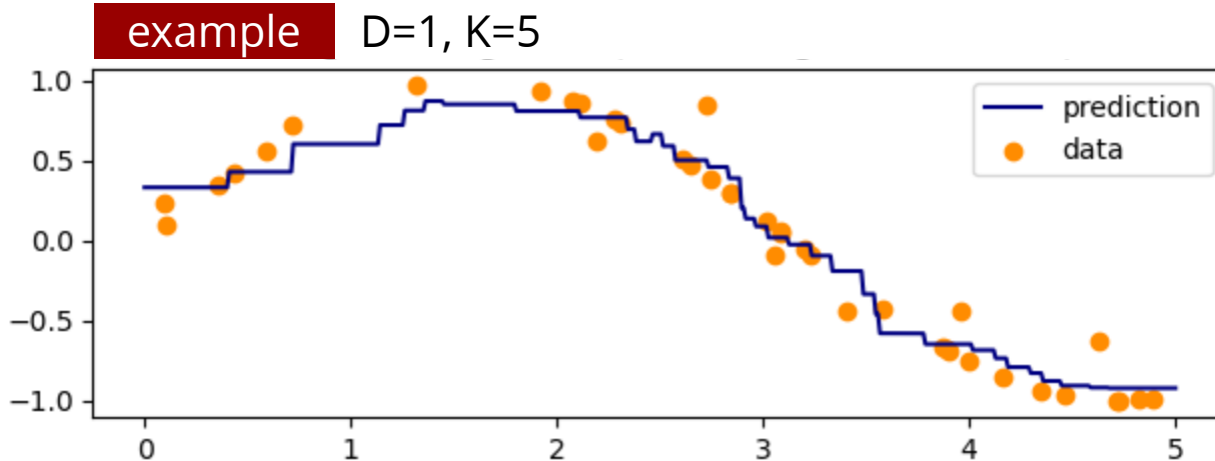
# K-NN regression

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

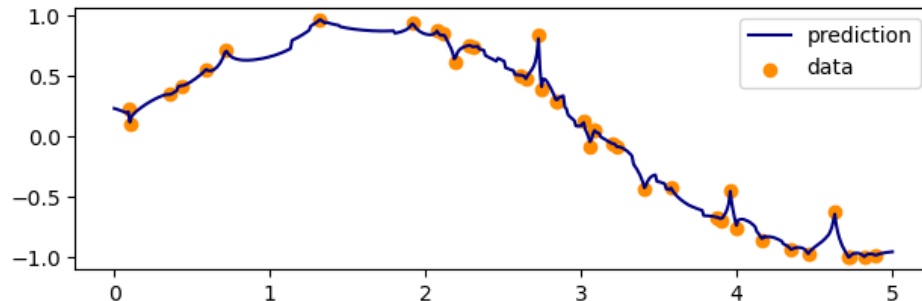
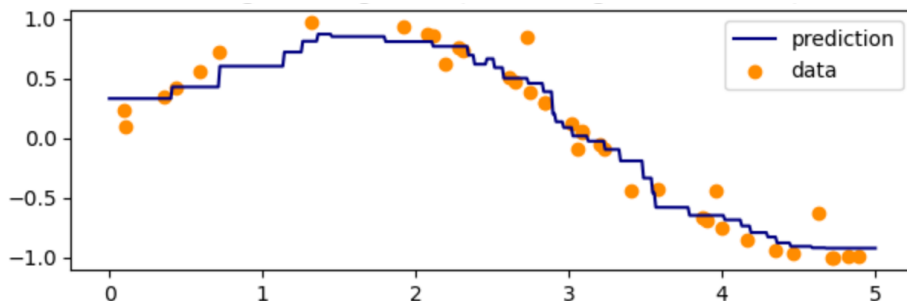


example from scikit-learn.org, see [here](#)

# 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



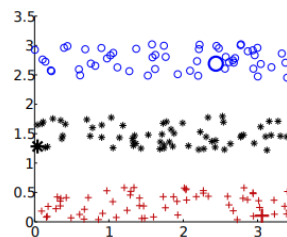
*example from scikit-learn.org*

in **fixed radius nearest neighbors** all neighbors in a fixed radius are considered  
in dense neighbourhoods we get more neighbors

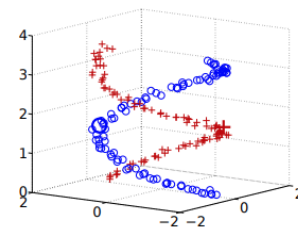
# K-NN for unsupervised and semi-supervised learning

## Semi-supervised setting:

Propagate labels to nearby points



(a) 3-Bands



(b) Springs

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

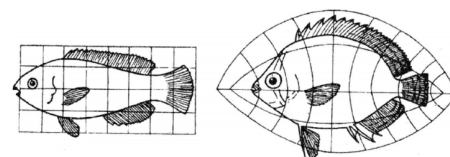
read more on KNN [here](#), and on clustering with Knn [here](#)



# 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

TA Lecture tomorrow for code review and QA

