Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

MSE | MASTER OF SCIENCE IN ENGINEERING

Machine Learning

F-MachLe

2. K-NN - an intuitive classification and regression system Curse of dimensionality

Jean Hennebert
Andres Perez Uribe

Plan - Supervised Learning for Classification tasks

2.1 Recap and examples

2.2 Intuitive classification with K-NN

2.3 Use case - K-NN for an image classification task

2.4 Regressor K-NN

2.4 The curse of dimensionality

Practical Work 2

2.1 Recap and examples

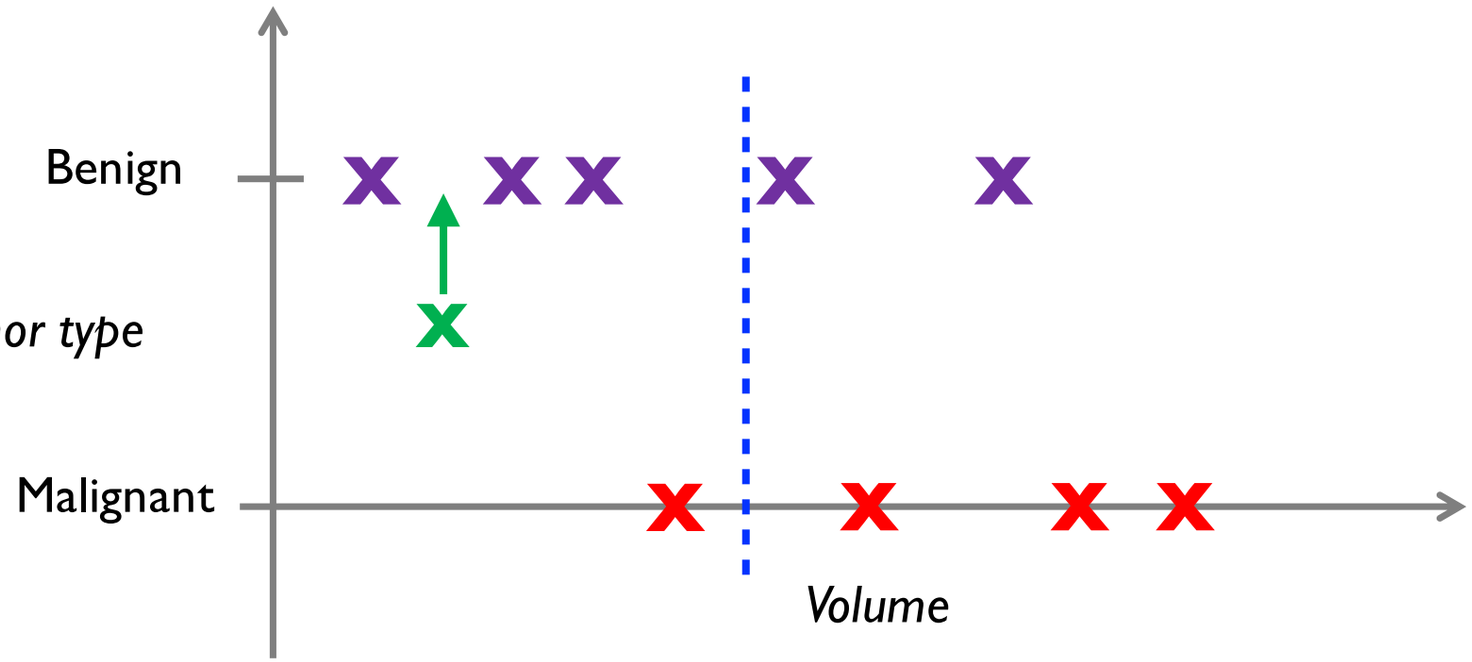
Classification task

Supervised learning - classification tasks

A **classification task** maps inputs **x** to a finite set of discrete outputs **y**. The outputs are the class labels corresponding to the different categories we want to predict.

- Usually classes are **mutually exclusive**, i.e. only one label is output of the system.
  - However, some systems are said **multi-label** when a given input x belongs to more than one class.
- 2-class systems are sometimes called **detection** or **verification** systems, where the objective is to answer yes/no questions
  - Biometric example: *Is this the face of Sheldon?* *Is the identity verified?*
  - Warship example: *Is a torpedo going to hit us?* *Is a torpedo detected?*

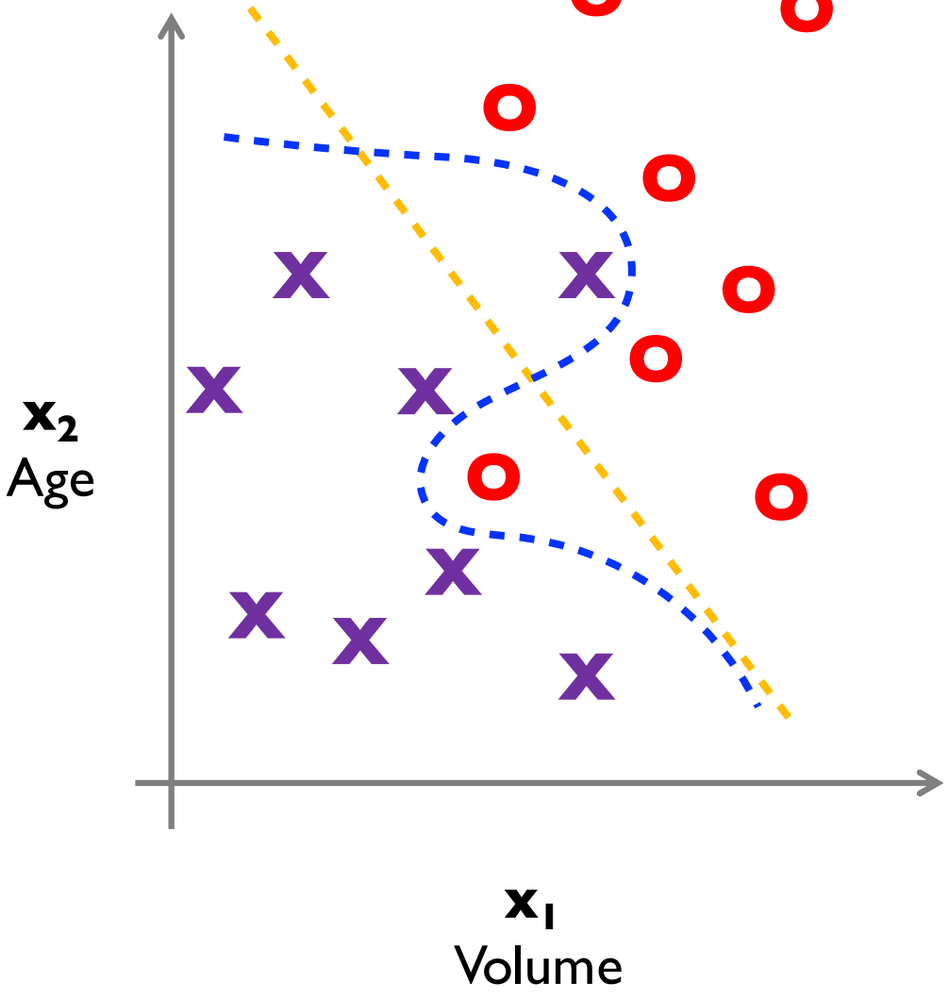
Supervised learning example 2 - tumour classification



The output of the system is discrete: this is called a **classification**

- We want to **categorise** a tumour into two classes: B or M
- We have a set of examples with the *right answers*
- We build a mathematical model explaining the mapping: *class = h(volume)*
  - In this case a simple threshold
- We can use the model to **classify** a new tumour

Supervised learning example 2 - tumour classification



- A typical way to increase the performance is to take into account more **features**
  - the input is becoming a vector with 2 components in this example
- The mapping function is *class = h(x1,x2)*
  - in yellow a linear model
  - in blue a non-linear model
- What other features could we bring in?
  - smoking quantity, family history, …

Tip: adding relevant features may increase performance. However this usually requires to have more data. See curse of dimensionality https://en.wikipedia.org/wiki/Curse_of_dimensionality

Some notations

$\mathbf{x}$	input variable of the system, also called **input feature**
y	output variable of the system, also called **target output**
$\widehat{y}$	**gotten output**, as computed by the mapping function
$(\mathbf{x}, y)$	a **training sample** (pair of feature and corresponding target)
$(\mathbf{x}_n, y_n)$	the training sample at index n in the training set ("in row" n)
N	the total number of training samples in the training set

The **training set** can then be noted

$$\{(\mathbf{x}_n, y_n); n = 1, \ldots, N\}$$

D	the dimension of the input space (x is a vector if $D > 1$)
$x_{n,d}$	the d'th coefficient of n'th training vector $\mathbf{x}_n$ (scalar value)

We can also define the **design matrix** X as the N-by-D matrix that contains the training examples' input values in its rows :

2.2 Intuitive classification with k-nn

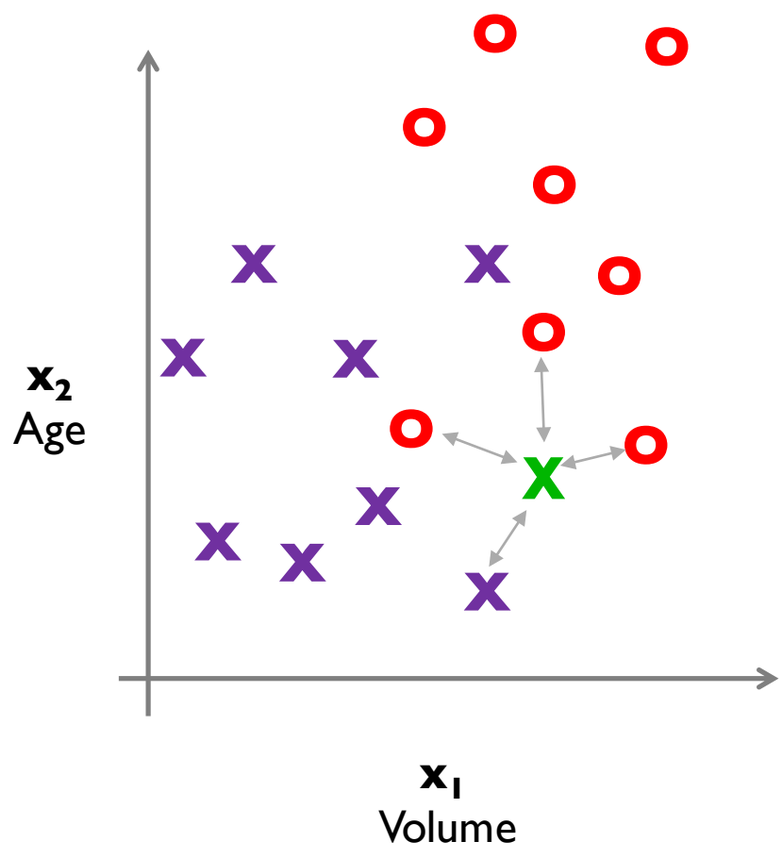
By Balu Ertl https://upload.wikimedia.org/wikipedia/commons/5/54/Euclidean_Voronoi_diagram.svg

k-nearest neighbour algorithm

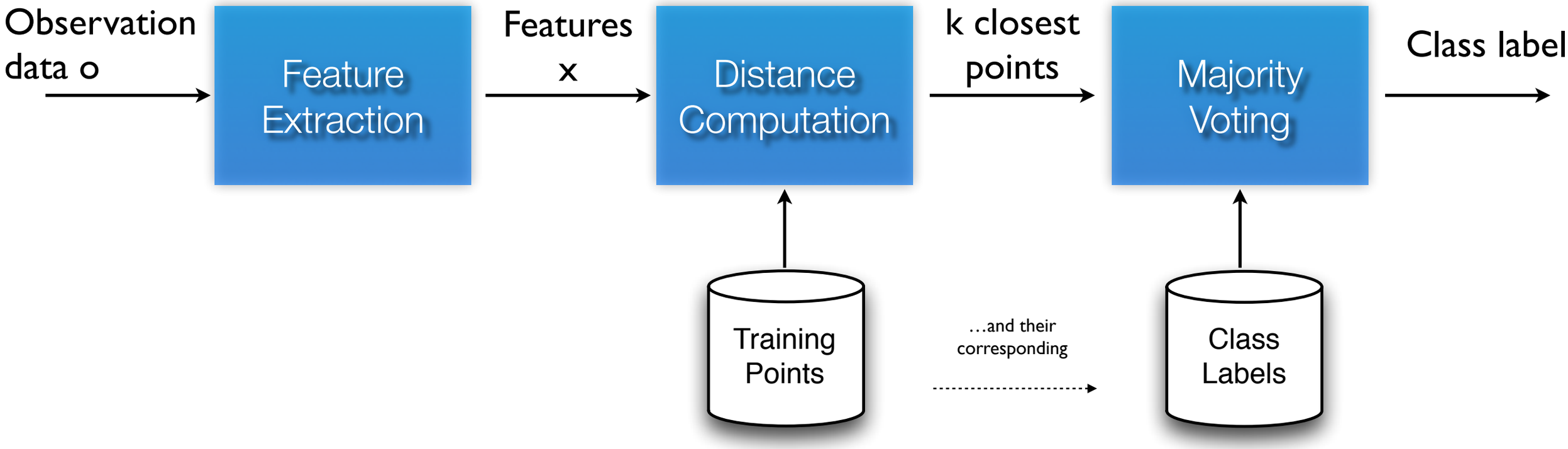
The **k-nearest neighbour** or **k-nn** algorithm is a method that classifies unlabelled examples based on their *similarity* with examples in the training set.

- The method requires
  - a parameter value *k*, the number of neighbour we consider to take the decision
  - a set of labeled examples
  - a distance metric
  - a voting strategy

With k=4 in this example, the new sample X is labelled O.



Intuitive classification: the k-nn approach



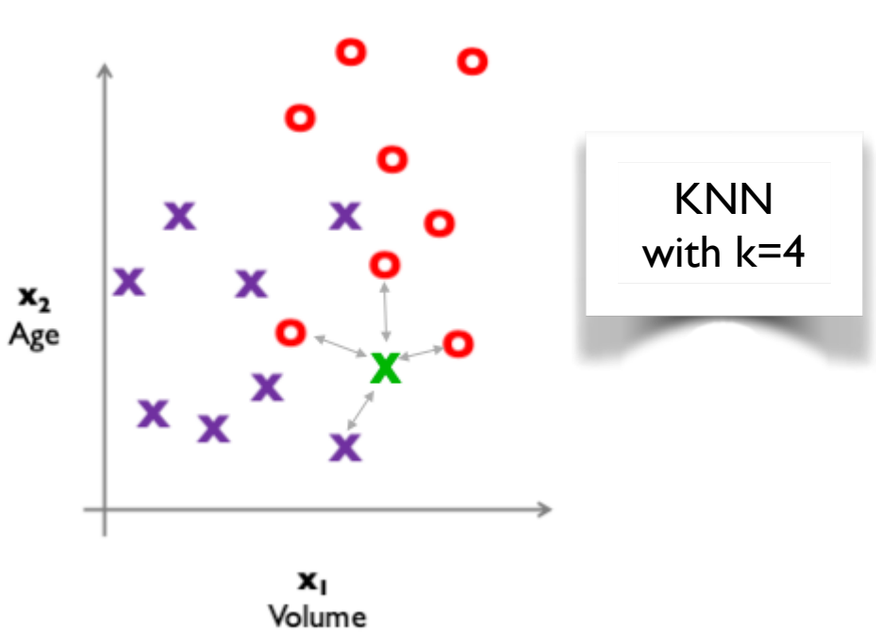
Critics *about k-nn*

1. *What kind of distance to use?*
2. *What is the best value for k?*
3. *Are we guaranteed to minimise the classification errors?*
4. *What if we have unbalanced training sets?* *(for example 10x more data for class a than class b)*
    - *We probably give priority to the class with lots of points*
    - *Is it bad or good?*

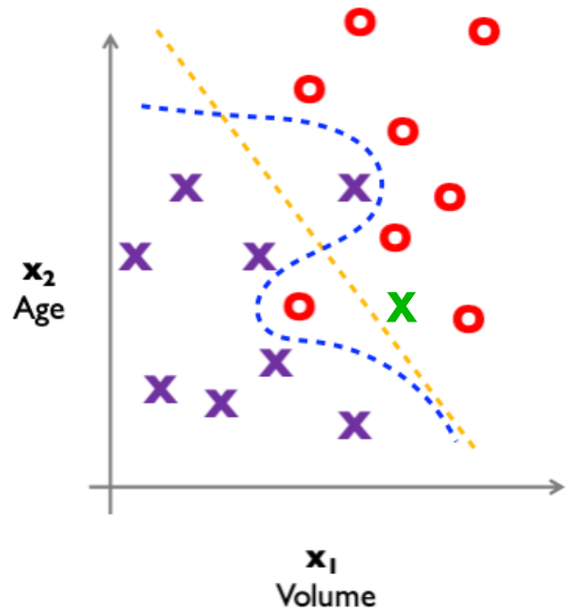
Instance-based vs model-based learning

Instance-based learning: the system learns the examples by heart and generalises by similarity

Model-based learning: the system learns a mapping function $\hat{y} = h(\mathbf{x})$ defined by an equation and a set $\{\theta\}$ of parameters



KNN
with k=4

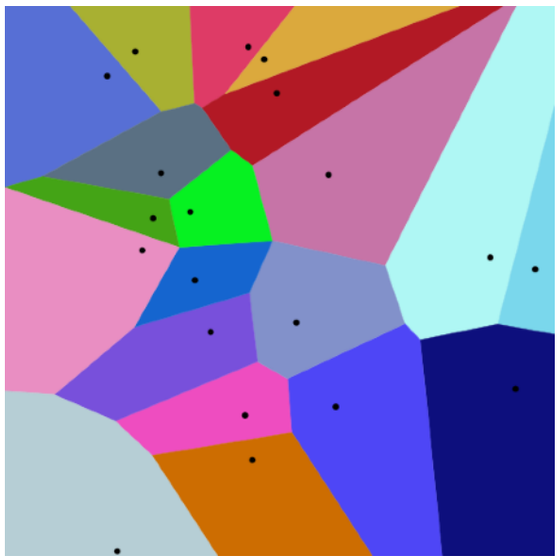


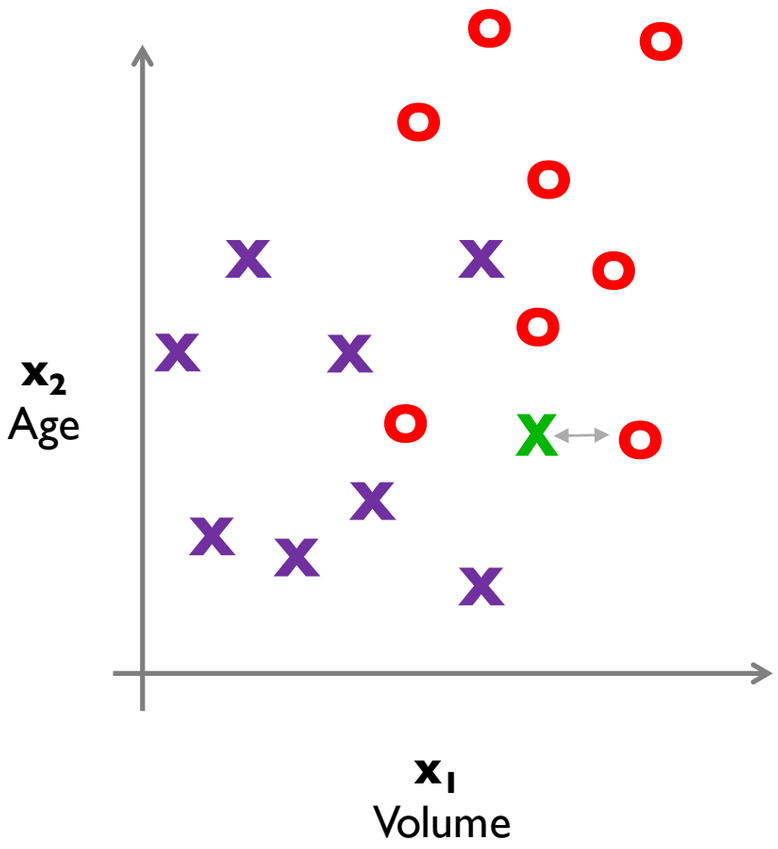
Logistic regression with a linear model (yellow) and a non-linear model (blue)

- The KNN is an instance-based learning approach

1-nn

- k-nn with k=1
- Predict the same value/class as the nearest instance in the training set
- We can visualise the partition regions of points “belonging” to a training point. This is called a **Voronoi diagram**





Assuming N examples in the training set, what are the temporal (cpu time) and spatial (memory) complexity of K-NN?

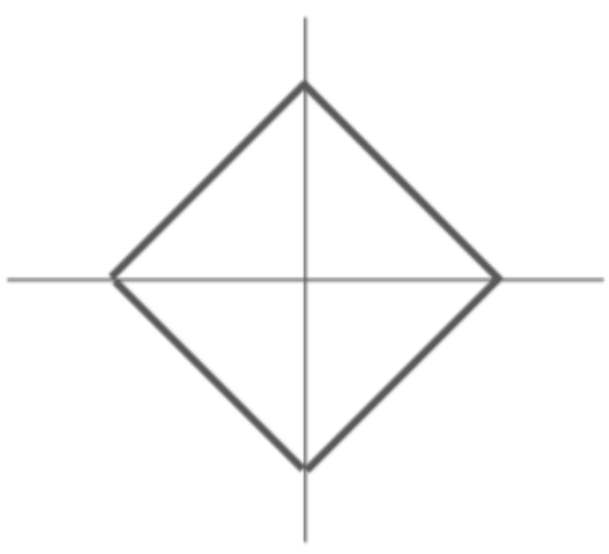
Activity

- At training time?
- For 1 prediction?
- Are these complexities good or bad?

Distance metrics to use?

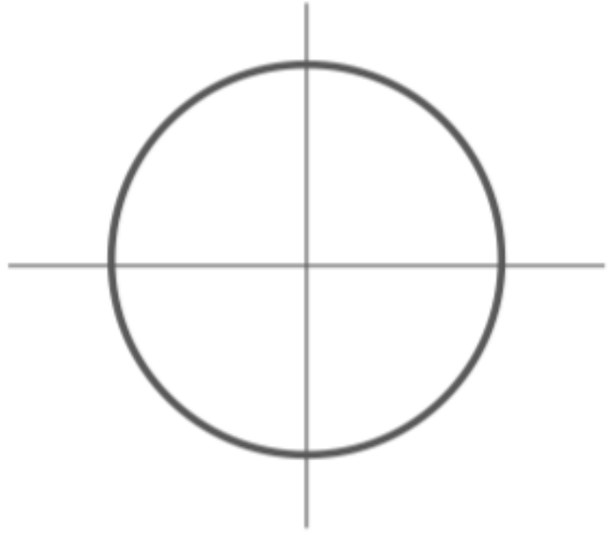
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

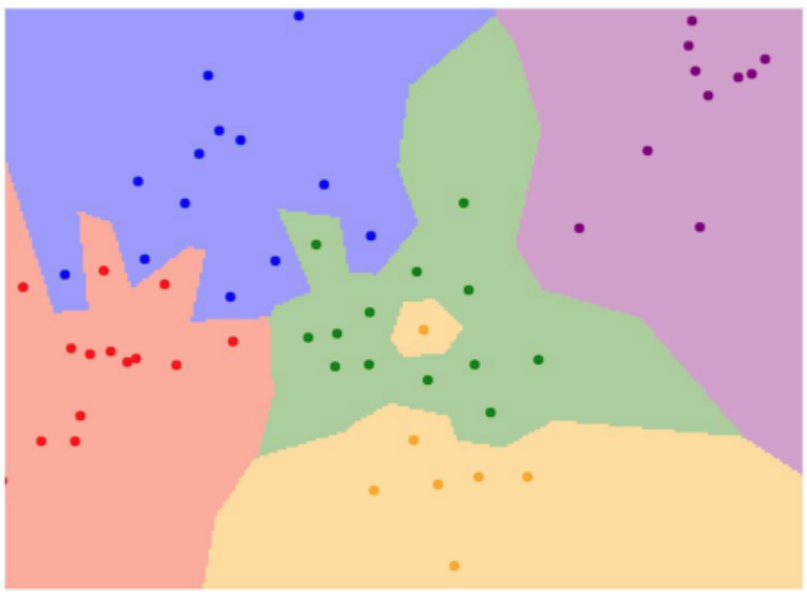
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



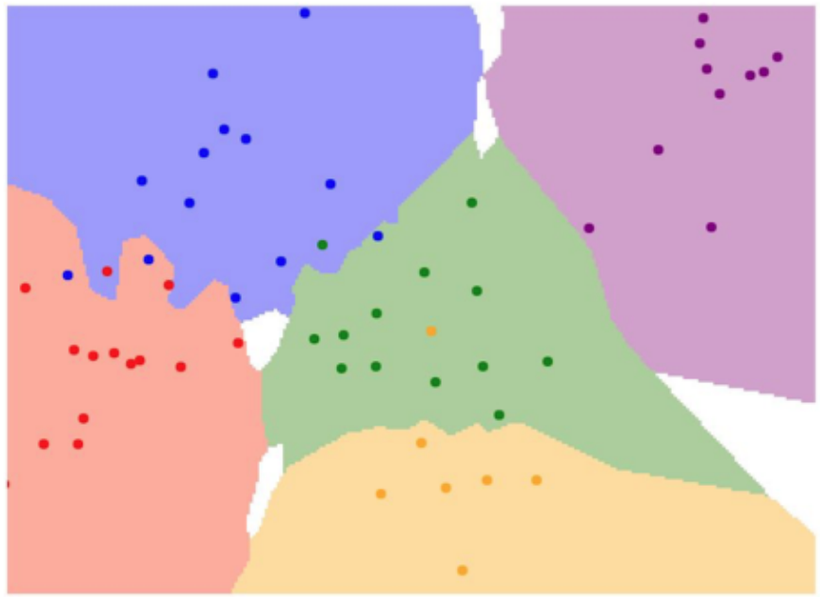
From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

1-nn vs. k-nn

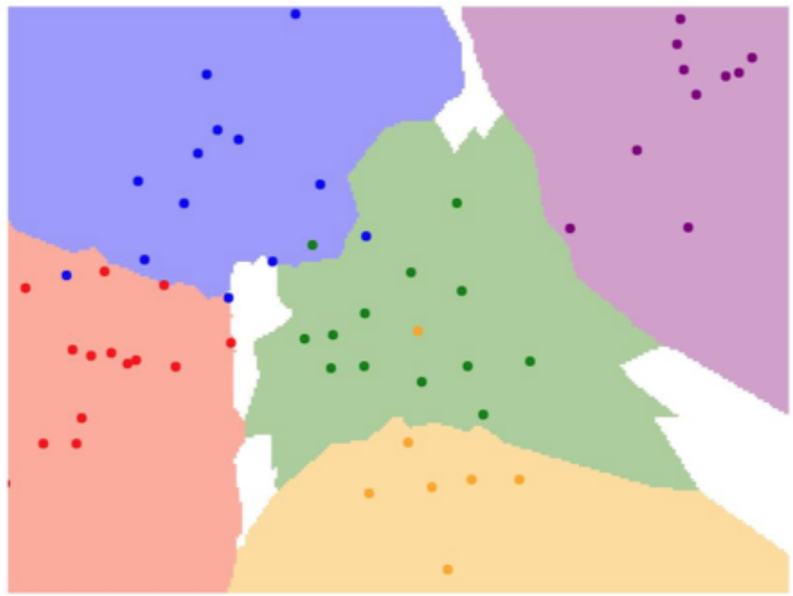
- Larger k brings some advantages
  - More confidence, especially with noisy data
  - Provides “probabilistic” information
    - The ratio of examples for each class gives information about the confidence or ambiguity of the decision
- However, too large values of k may be detrimental
  - It destroys the locality of the estimation since farther examples are taken into account
  - It increases the computational burden



K = 1



K = 3

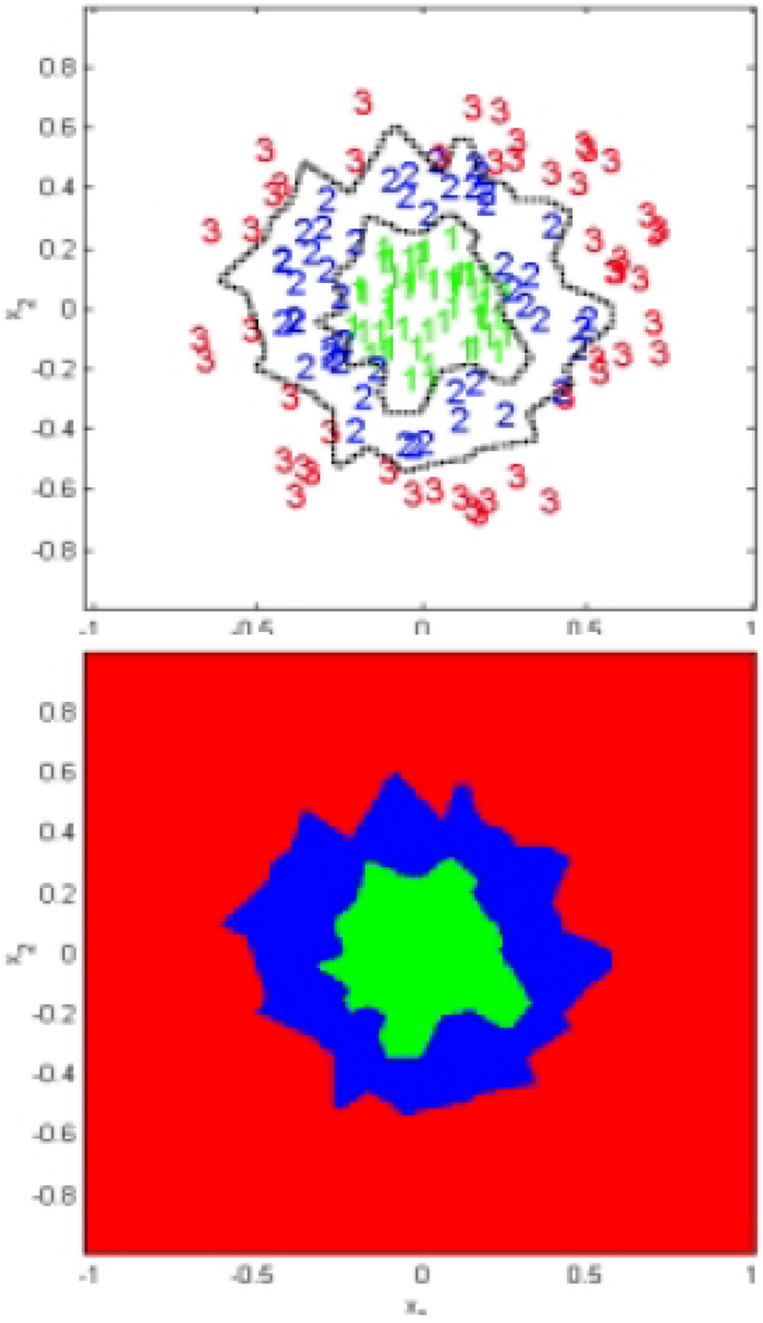


K = 5

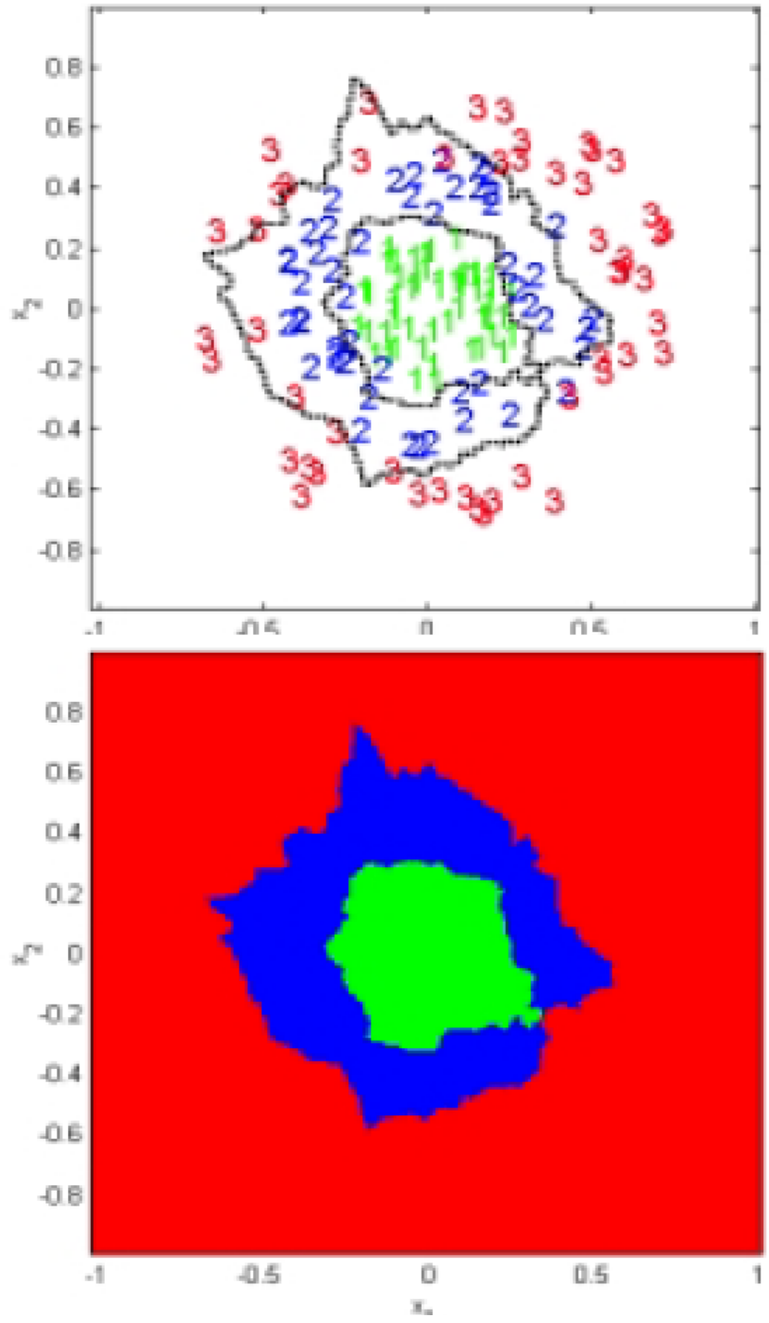
From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Check out the live demo under http://vision.stanford.edu/teaching/cs231n-demos/knn/

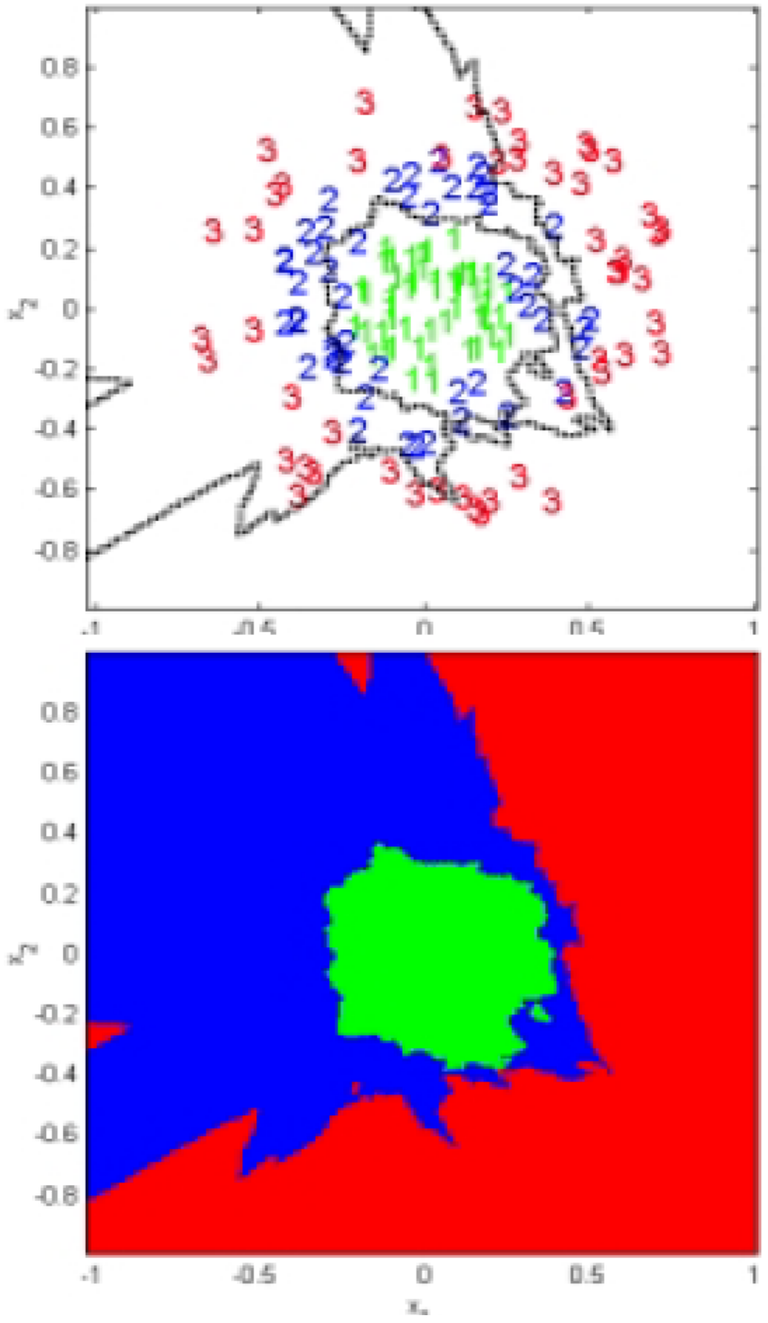
k=1



k=5



k=20



Integer values (and colours) indicate class labels.

k-nn hyperparameters to tune

- Hyper-parameters to be tuned
  - **normalisation type**: none, min-max rescaling, zero-norm[1]
  - **distance metric**: Euclidian, Manhattan, Mahalanobis, …
  - **k**: from 1 to …
  - **strategies in case of no majority**: min distance class, go to k+1, …
- k-nn is heavy in terms of
  - **memory**: the full training set needs to be stored and accessible
  - **cpu**: the distance is computed against the whole training set O(N)
    - some optimisation techniques can be used such as k-d tree

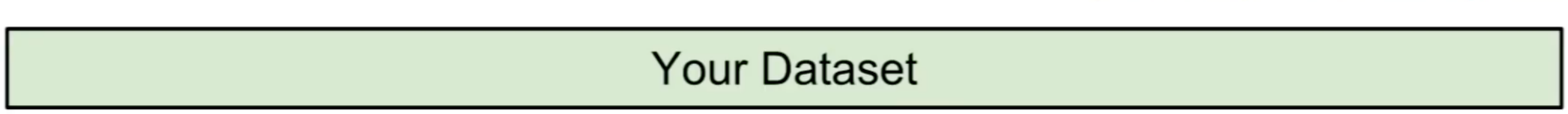
(1) See chapter on data preparation in few weeks to a full range of potential normalisations

k-nn tuning - hyper parameters tuning

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

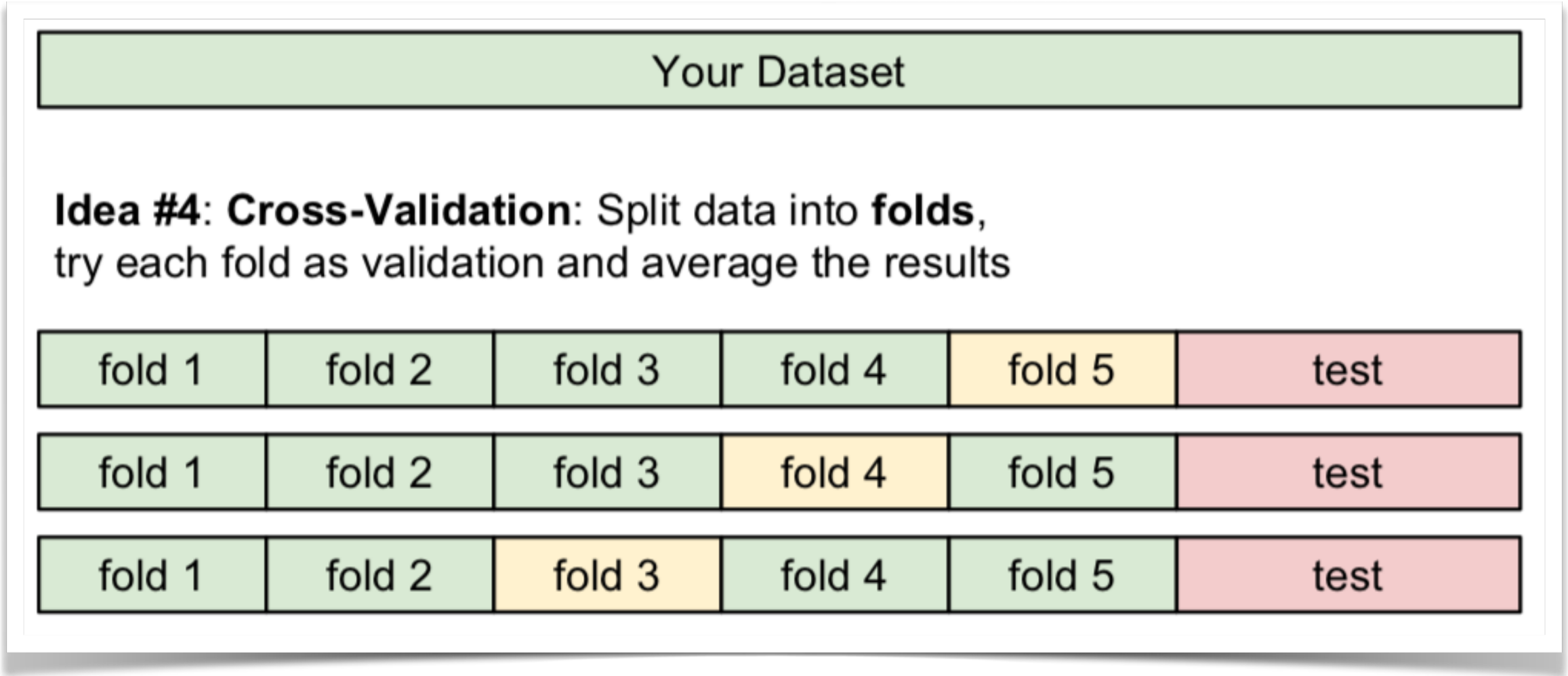
BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

k-nn tuning - hyper parameters tuning



ML Engineering tip: never do idea 1, do at least idea 2, even better if you do idea 3.
In case of small data sets, do idea 4 and observe the variability of your performances.

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

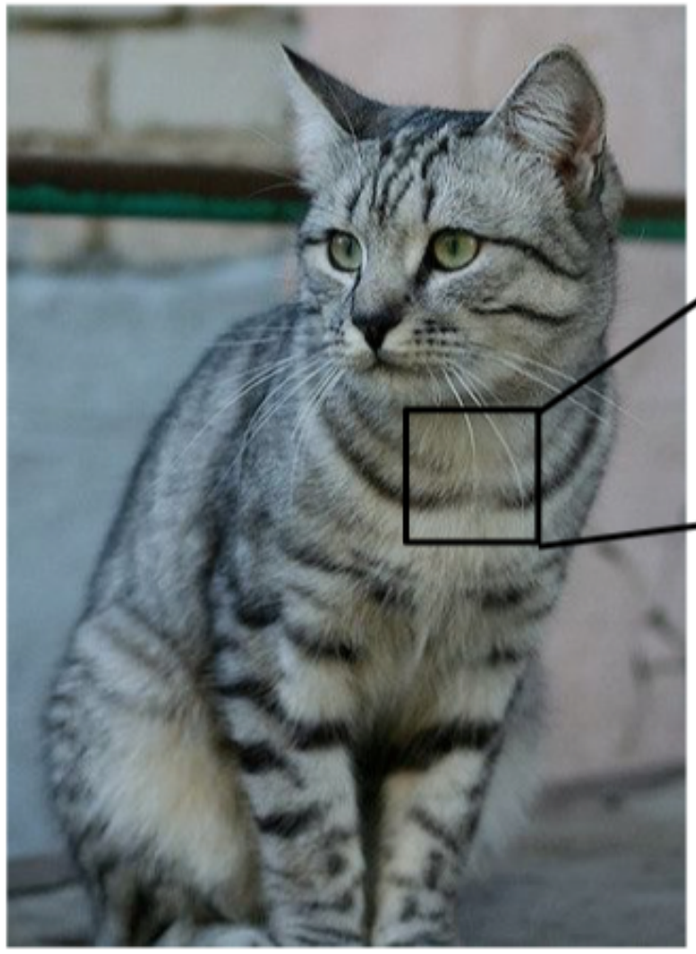
2.3 Use case: k-nn for image classification

By Balu Ertl https://upload.wikimedia.org/wikipedia/commons/5/54/Euclidean_Voronoi_diagram.svg

Some preliminary questions

- How to compute image distances?
- What metric to use?
- Is using KNN a good idea for image classification?
  - If no, why?
  - If yes, in which conditions?

Image representation



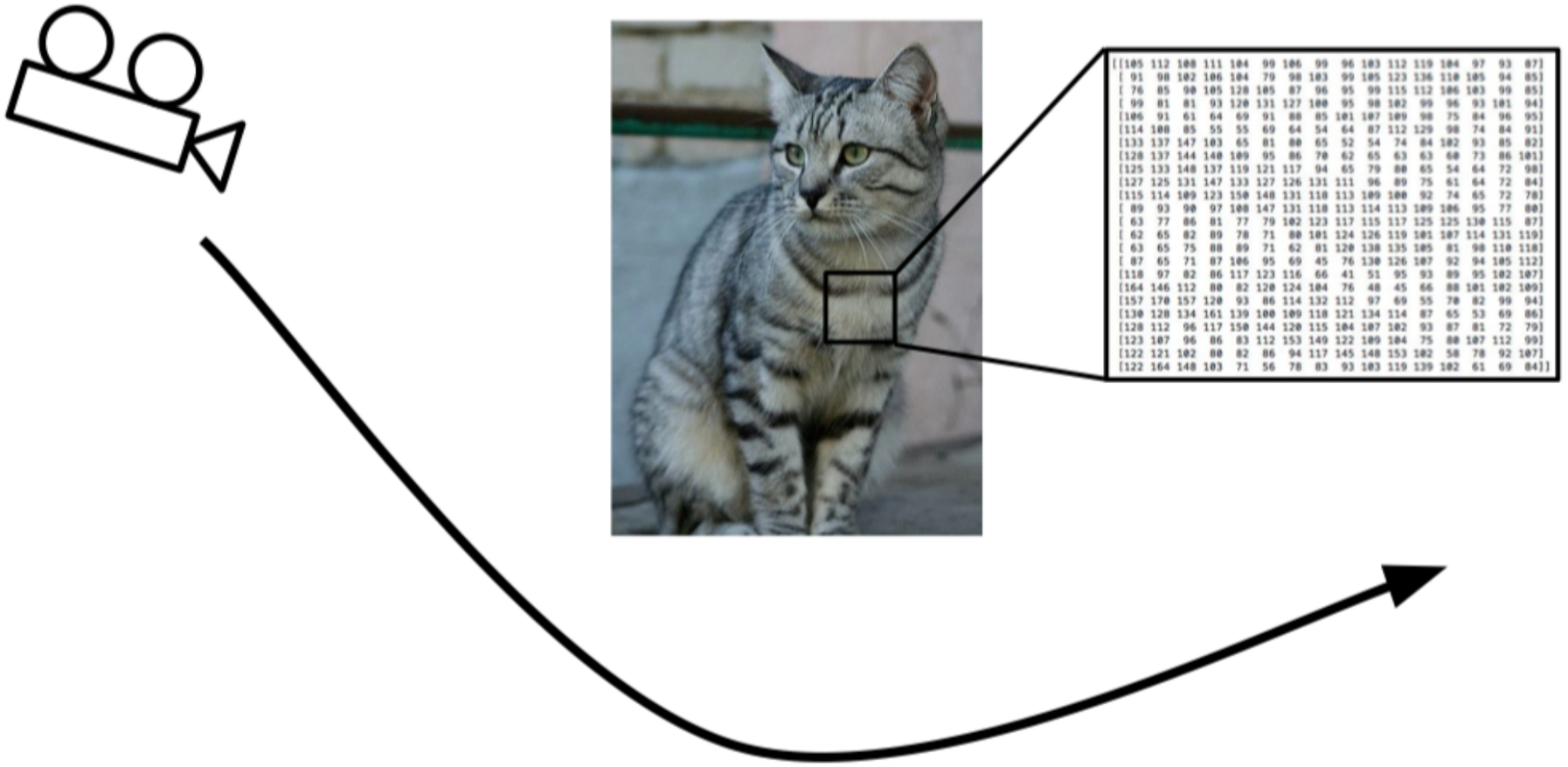

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: viewpoint



From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

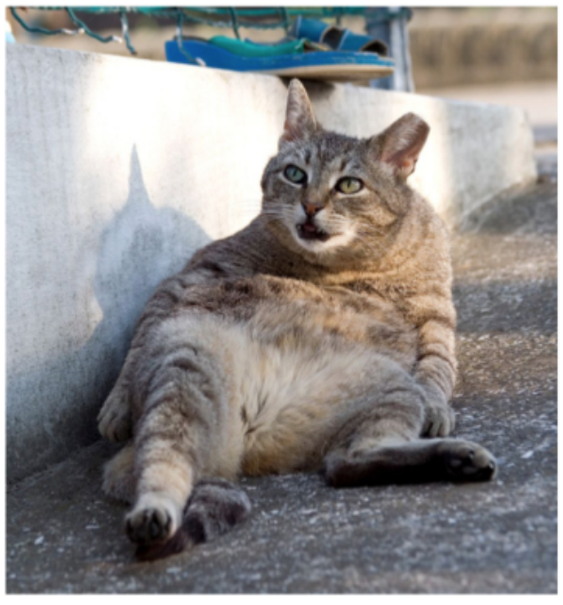
Challenges: illumination / deformations

This image is CC0 1.0 public domain

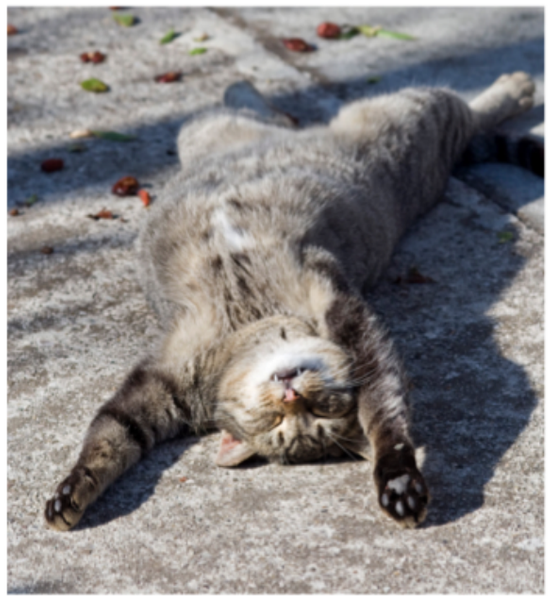
This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

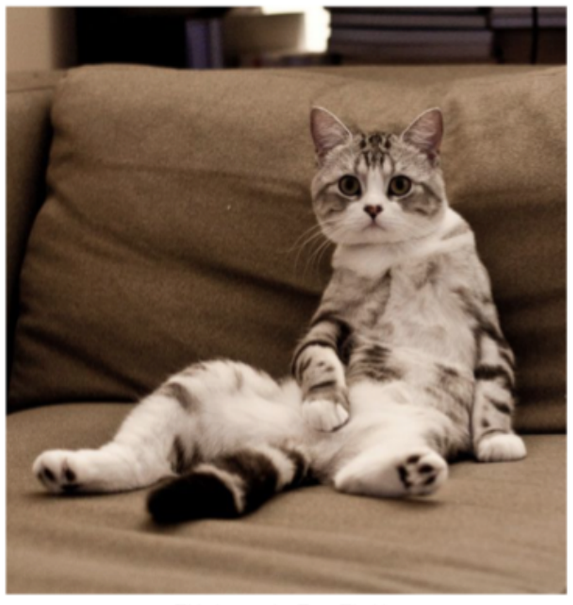


This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by Umberto Salvagnin is licensed under CC-BY 2.0

This image by sare bear is licensed under CC-BY 2.0



This image by Tom Thai is licensed under CC-BY 2.0

Challenges: occlusions / background

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Challenges: intra-class variabilities

This image is CC0 1.0 public domain

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Machine learning approach with KNN

```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Let's assume we use an L1 distance metric

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

\-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

add → 456

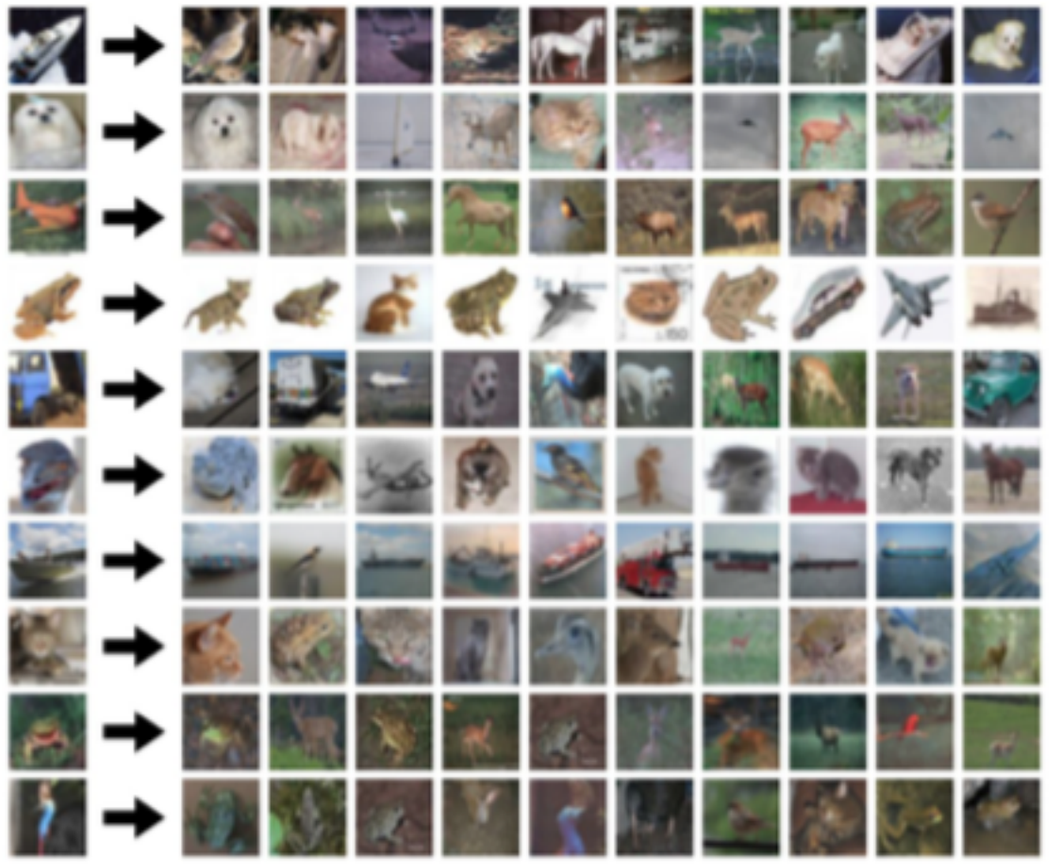
L2 (Euclidian) could also be appropriate depending to the type of images.

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Example on CIFAR10

10 classes
50,000 training images
10,000 testing images

Test images and nearest neighbors



From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Memorize training data

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

For each test image:
Find closest train image
Predict label of nearest image

argmin returns the index of the minimum value in distances

Broadcasting is used here. See Ex 1 of PW-02

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

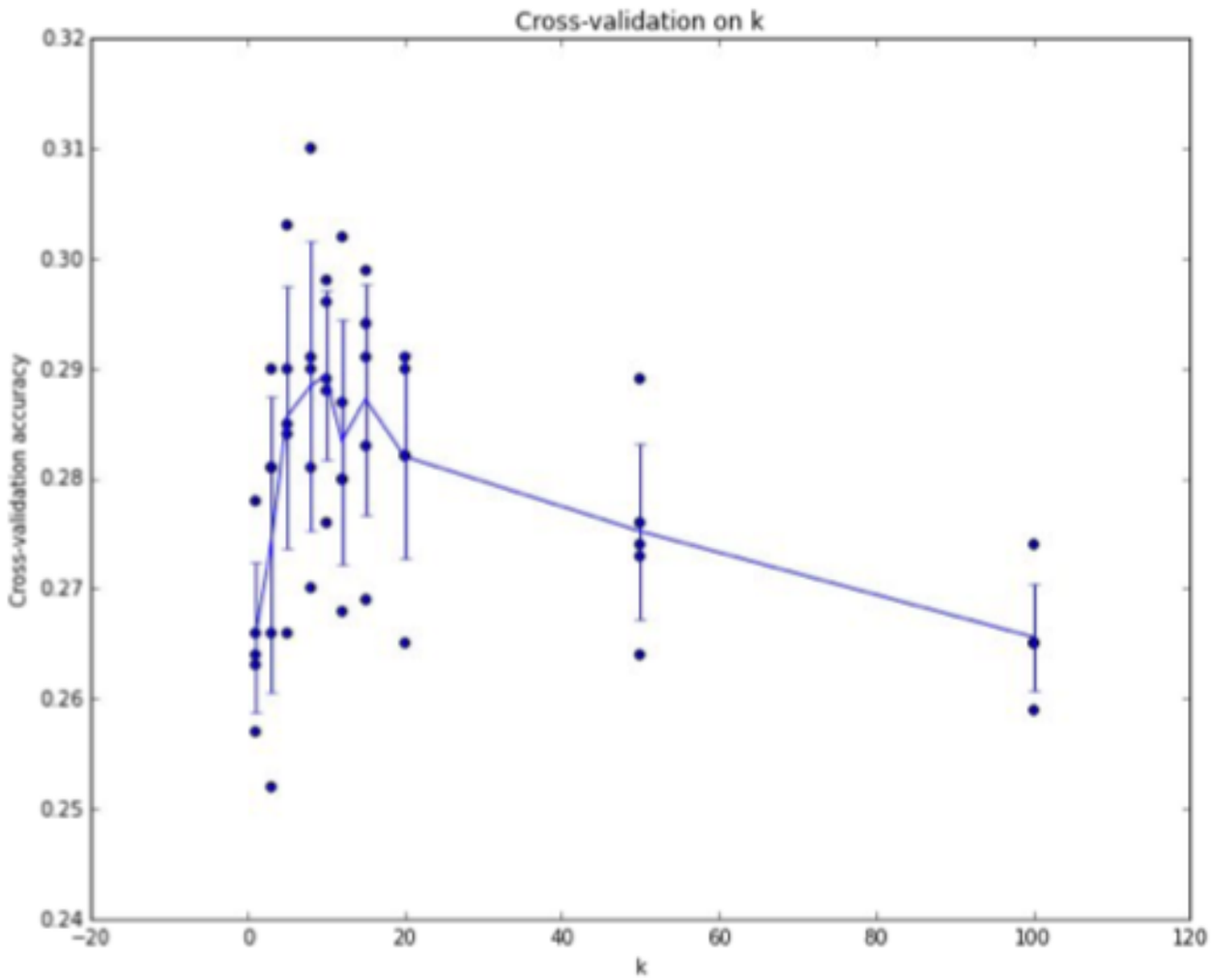
Classification results

Nearest neighbours

About 30% accuracy on 10 classes

From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Setting Hyperparameters



Example of
5-fold cross-validation
for the value of **k.**

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

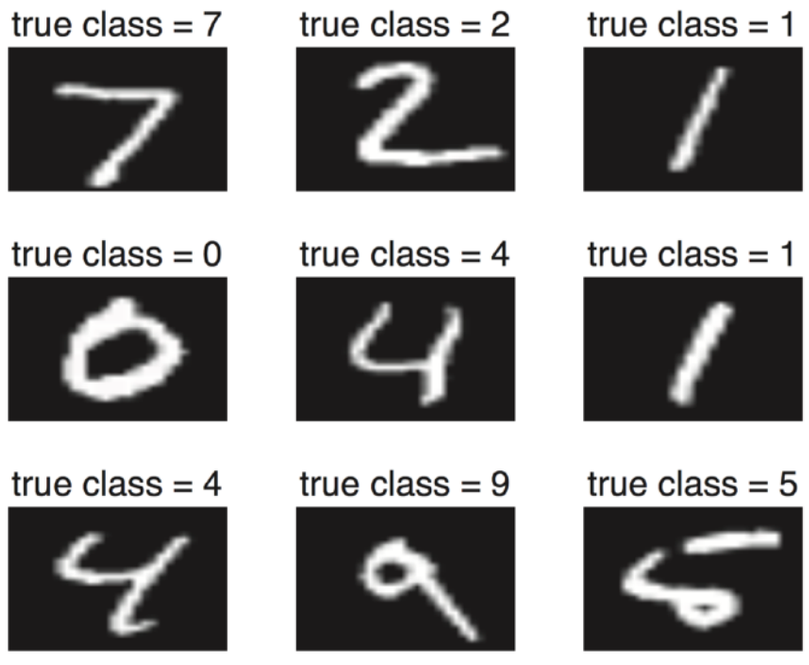
(Seems that k ~= 7 works best for this data)

About 30% accuracy on 10 classes. In practice KNN is a bad idea on images with high variabilities.

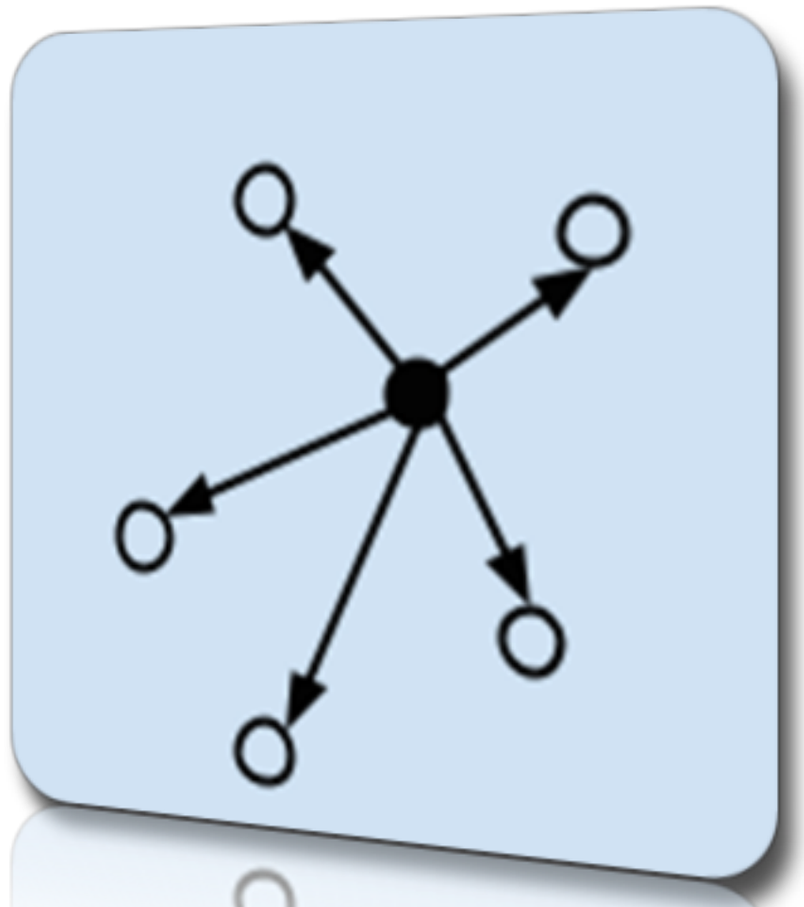
KNN on MNIST handwritten digits images

Activity

- KNN shows decent performances on this dataset, above 90% accuracy.
  - Why? State several reasons.



2.4 Regressor KNN



What is it?

How to use it?

KNN Regressor

The **k-nearest regressor** algorithm is a method that predicts a target value $\hat{y}$ based on the $\{y_k\}_K$ values of the K closest examples in the training set.

- We could opt for a simple average of the y_k values with

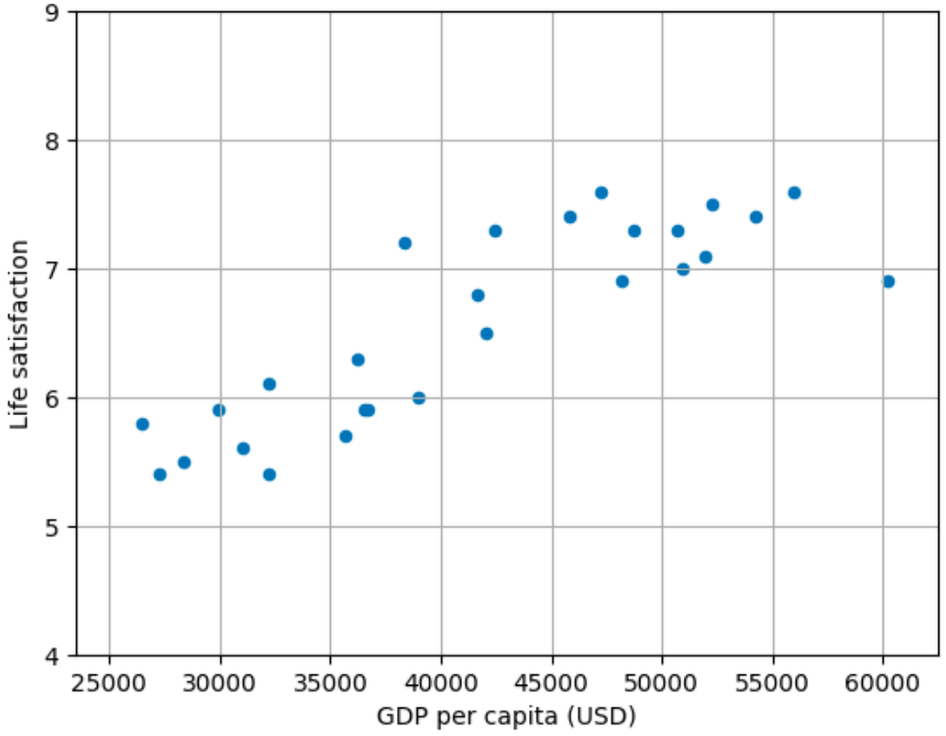
  $\hat{y} = \frac{1}{K}\sum_{k=1}^{K} y_k$, where each y_k are corresponding to the k-nearest neighbour.

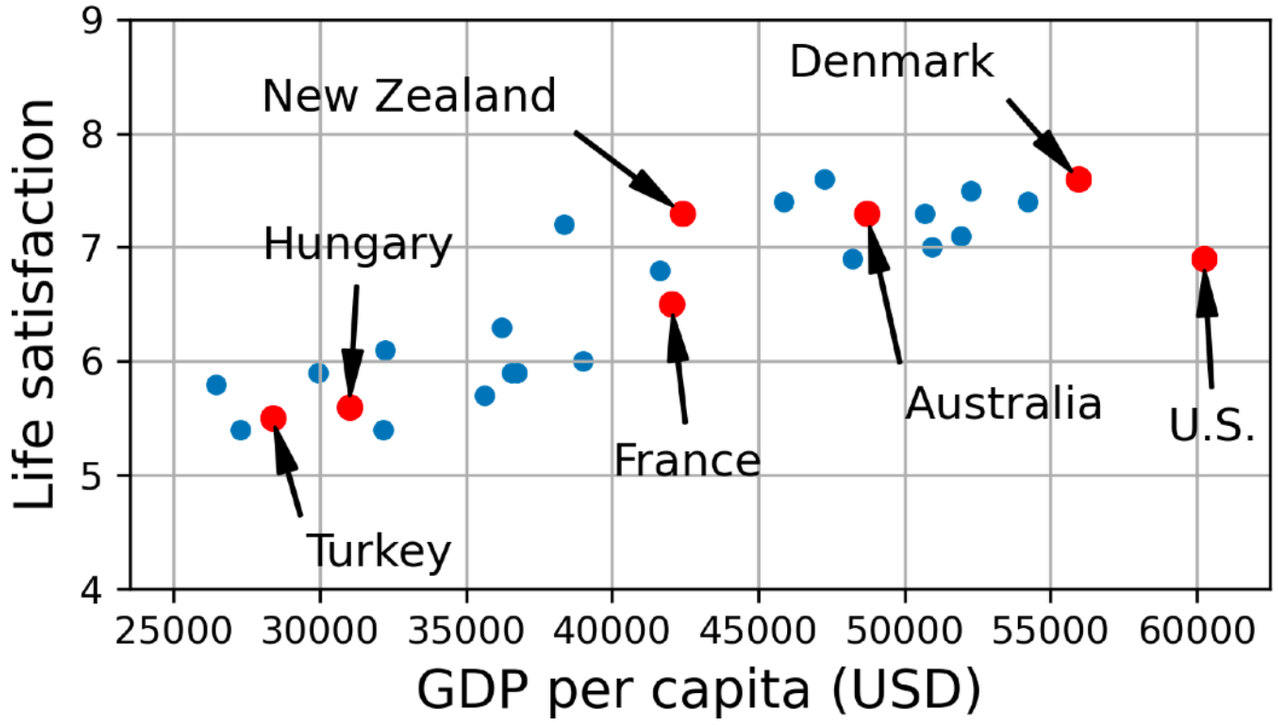
- Other strategies apply, e.g. using a weighted average where the weights are decreasing either with the rank :

  $\hat{y} = \sum_{k=1}^{K} w_k y_k$, with e.g. $\{w_1 < w_2 < \ldots < w_K\}$ and $\sum_{k=1}^{K} w_k = 1.0$

  or with the inverse of the distance to the nearest point $w_k \propto \frac{1}{d(x, x_k)}$

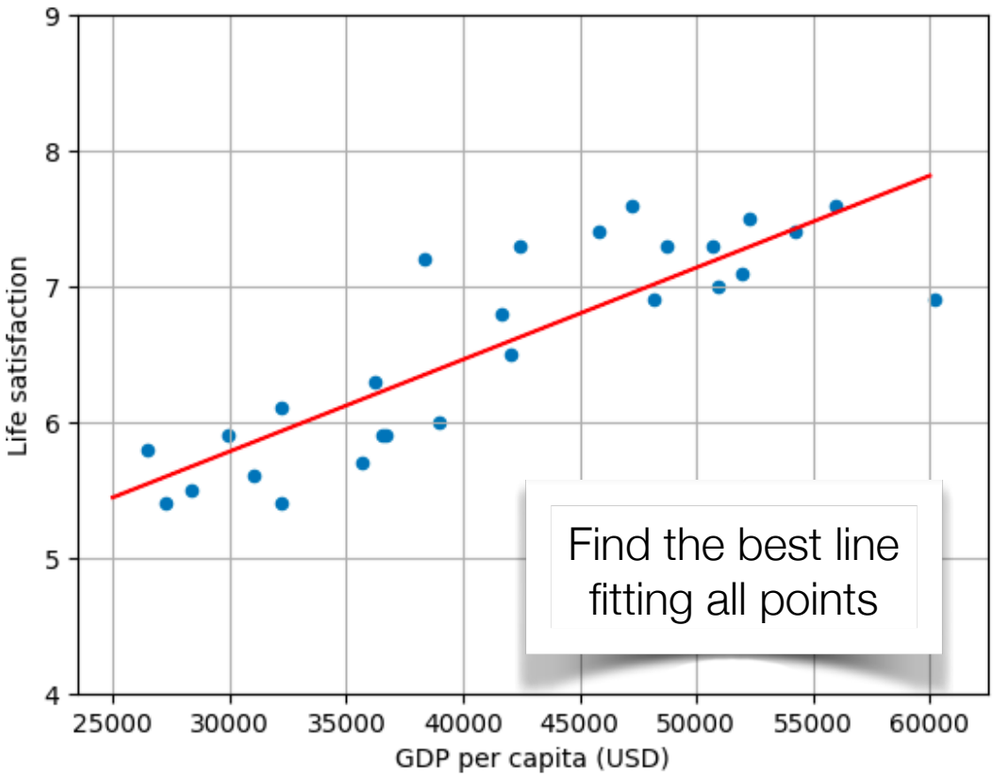
Example: more $$$ = happiness ?

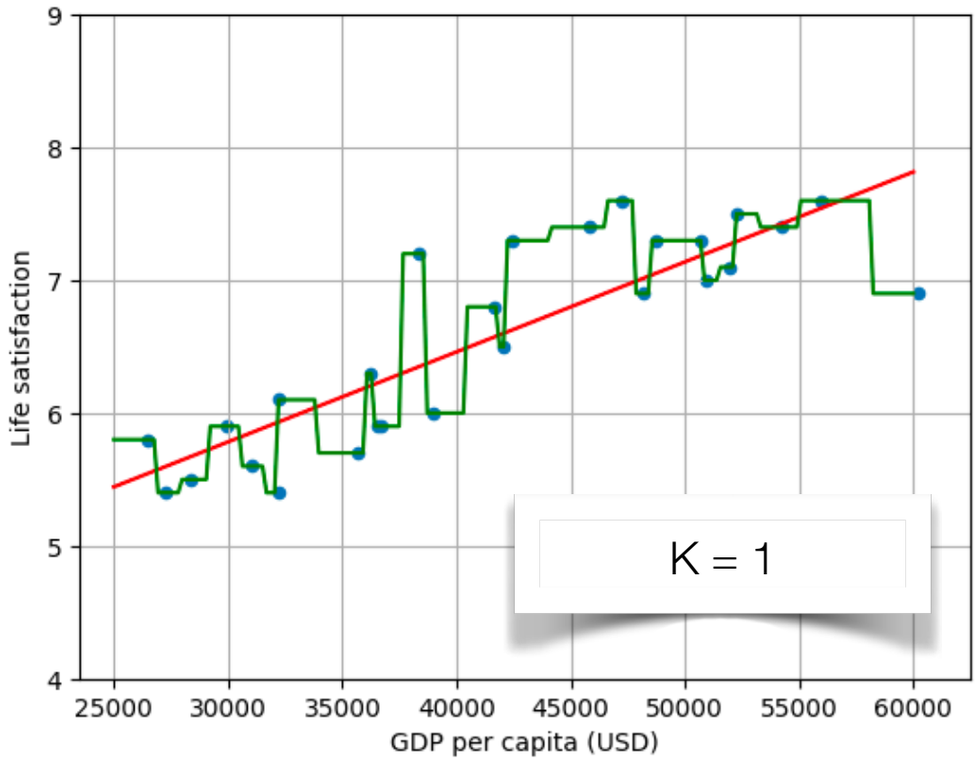




- Source: https://ourworldindata.org/grapher/gdp-vs-happiness
- In x: the GDP = gross domestic product, a measure of the goods and services produced in a given country

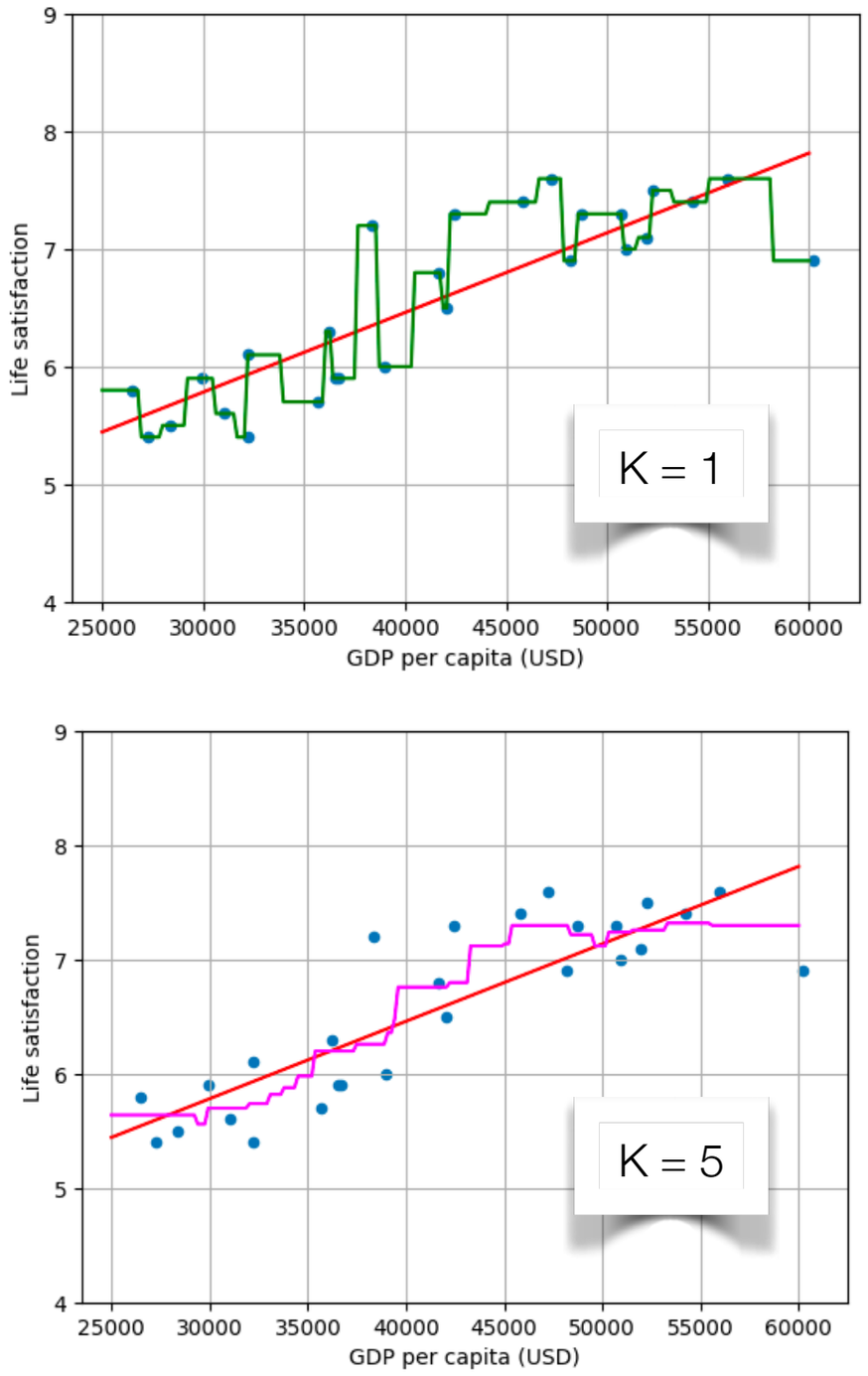
Linear regression vs K-NN regression

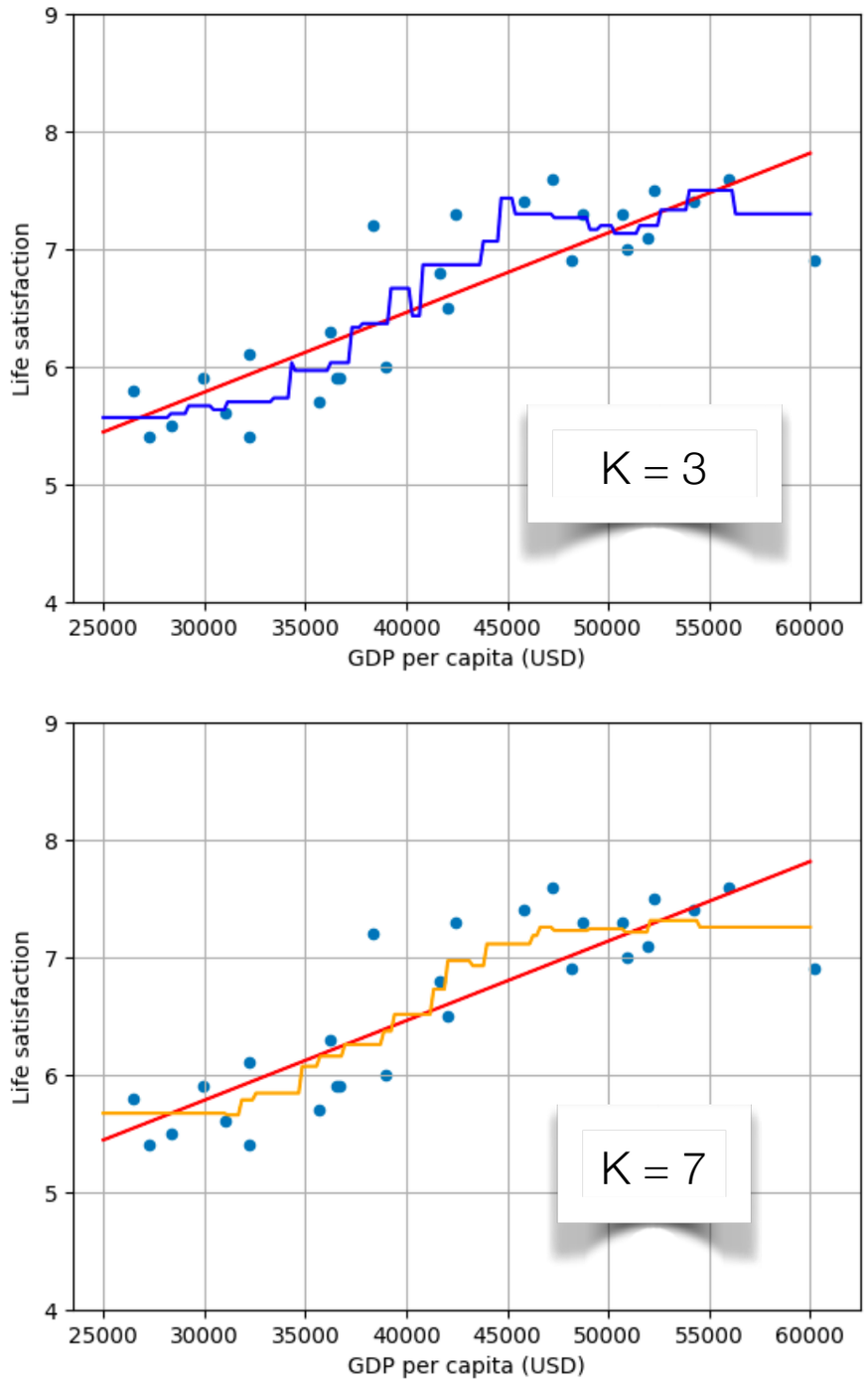




- On the left, we used the linear regression algorithm that finds the equation $\hat{y} = \theta_0 + \theta_1 x$ by minimising the least square errors between the line and the input points.
  - We will re-visit linear regression soon!

Increasing K





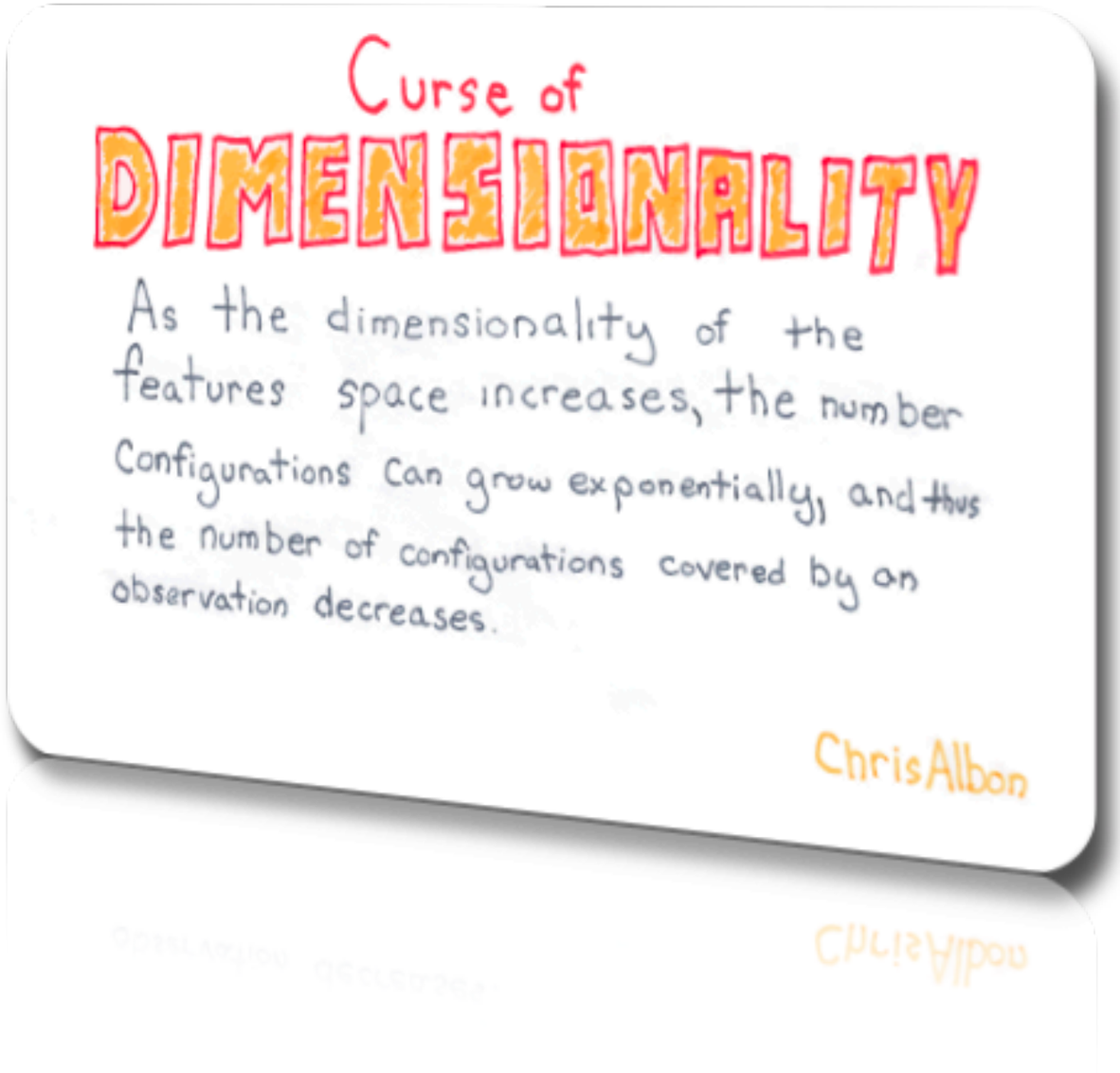
- Higher K brings smoothness

2.5 The curse of dimensionality

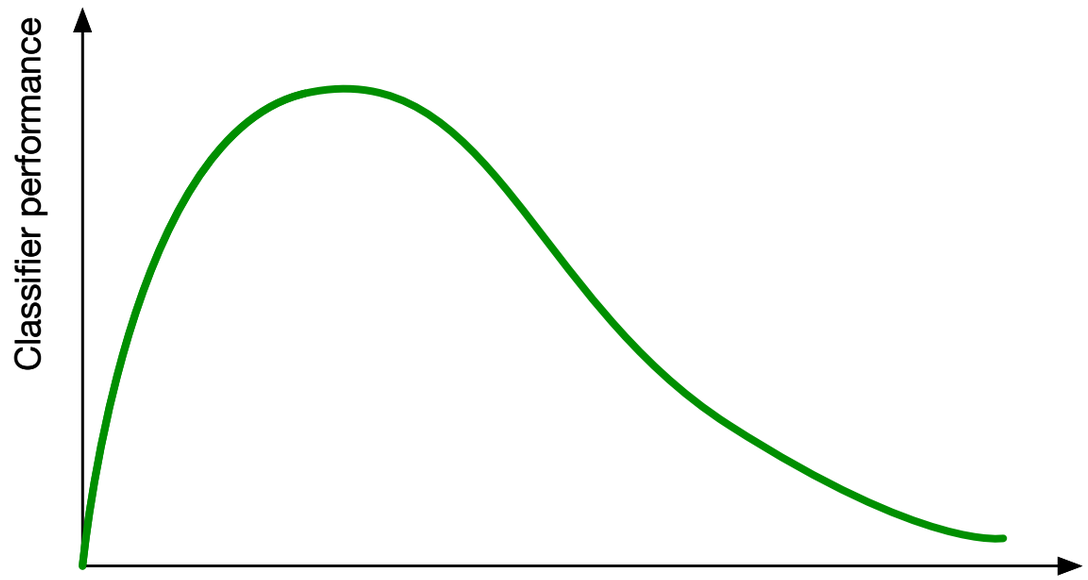
What is it?

Why that?

Does it impact k-nn?



Curse of dimensionality - what is it?

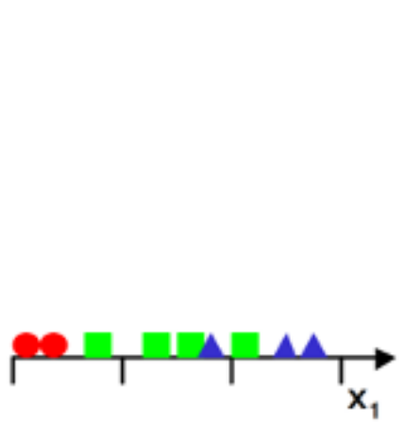


ML Engineering tip: adding features typically increases the performance

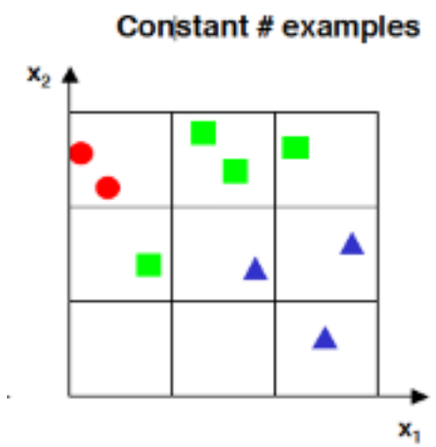
- Yes, having extra information (features) usually helps
- However, we often observe a decrease of performances when inserting more features

The **curse of dimensionality** is when we observe a decrease of performance when increasing the number of features. This is due to the lack of samples N with respect to the dimensions D of the input space.

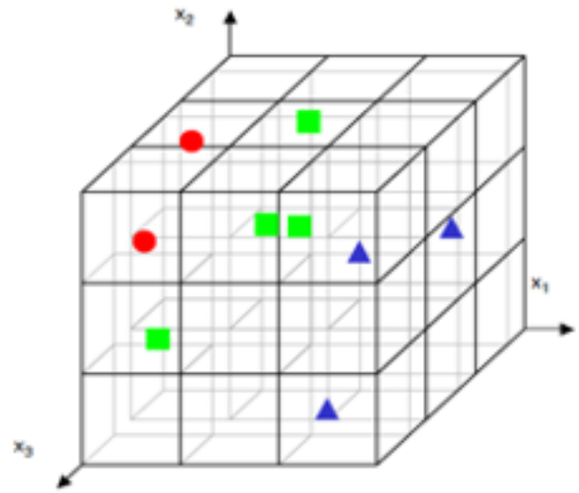
Curse of dimensionality - why that?



D=1
3 regions



D=2
9 regions



D=3
27 regions

… 3^D regions

- Data sparsity occurs when moving to higher dimensions.
- The volume of the space (the number of regions) grows exponentially with the number of features.

Revised **ML Engineering tip**: adding features typically increases the performance, up to a certain point where the curse of dimensionality impacts the performance.

Curse of dimensionality - impact on k-nn

- Distance-based classifiers as k-nn are impacted by the curse of dimensionality
  - The distance is indeed computed considering the D dimensions of the features
- Side note: a decrease of performance when inserting more features could also be the result of:
  - Noisy features
  - Redundant features
  - Un-normalized features
- Using k-nn often involves **feature selection** and well-thought **normalisation** procedures[1]

(1) Stay tuned and visit chapter on data preparation in few weeks to a full range of potential normalisations

Conclusions on K-NN

- K-NN is an instance-based approach - vs a model-based approach
  - Quite bad in terms of cpu at inference time
- Simple, intuitive **supervised classification** algorithm
  - Find the k nearest neighbours in the training set
  - Predict the label according to a majority vote on the neighbours
- Can also be used to perform **supervised regression**
  - Find the k nearest neighbours in the training set
  - Predict the value according to, e.g. the averaged y values of the neighbours
- Distance metric and K are **hyperparameters**
- Choosing hyperparameters using the training set through a **validation** or **cross-validation** procedure is a good idea
- Beware of the **curse of dimensionality** when using k-nn