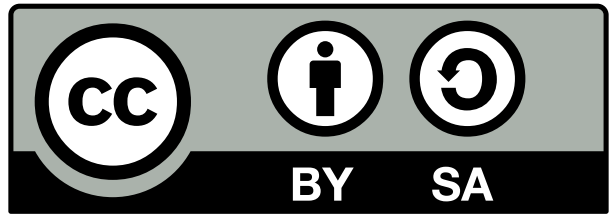


This presentation is released under the terms of the
Creative Commons Attribution-Share Alike license.

You are free to reuse it and modify it as much as you want as long as:

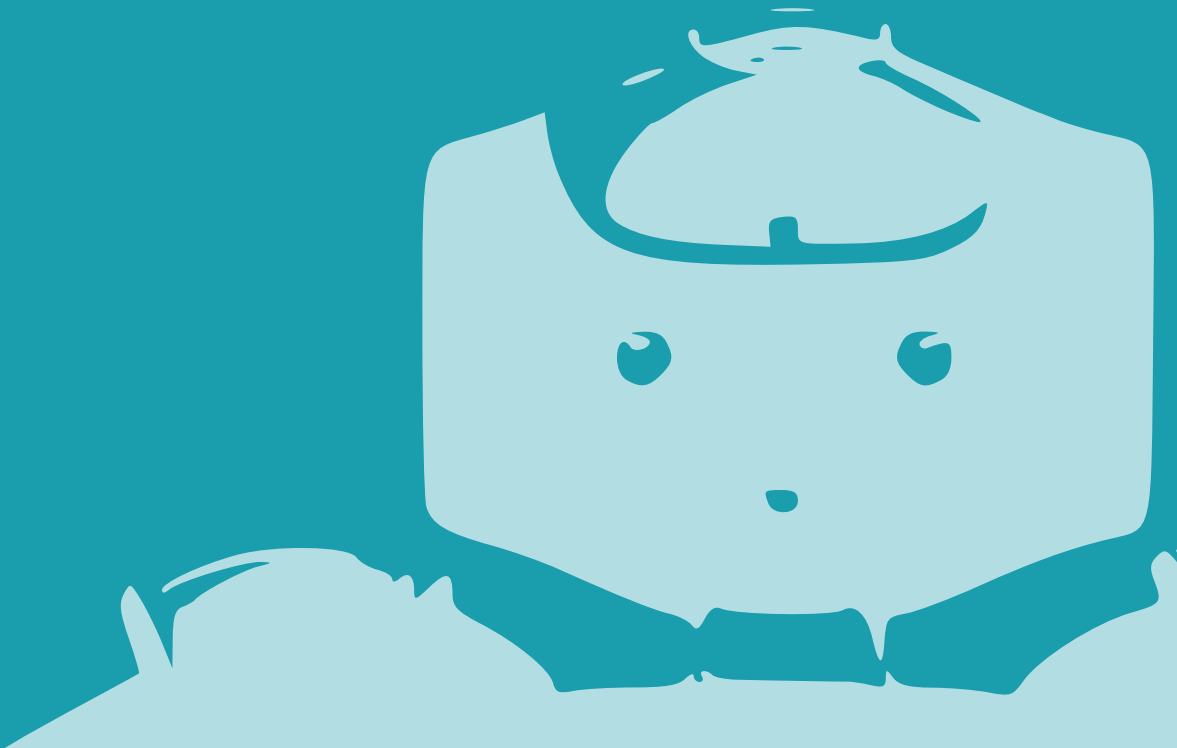
(1) you mention Séverin Lemaignan as being the original author,
(2) you re-share your presentation under the same terms.

You can download the sources of this presentation here:

github.com/severin-lemaignan/lecture-hri-ml-for-hri

Human-Robot Interaction

Machine Learning for HRI



Séverin Lemaignan

Bristol Robotics Lab
University of the West of England

IN THIS LECTURE

- Build your own emotion classifier from scratch!
- Basic classifiers: kNNs, SVM, MLP
- ML to teach a robot social behaviours

EMOTIONS?

PAUL EKMAN AND BASIC EMOTIONS

Paul Ekman, a psychologist, found that when shown facial expressions people across the world all recognised six **basic emotions**.

Source: emotionresearcher.com

PAUL EKMAN AND BASIC EMOTIONS

Paul Ekman, a psychologist, found that when shown facial expressions people across the world all recognised six **basic emotions**.

Anger, disgust, fear, happiness, sadness and surprise.

PAUL EKMAN AND BASIC EMOTIONS

Paul Ekman, a psychologist, found that when shown facial expressions people across the world all recognised six **basic emotions**.

Anger, disgust, fear, happiness, sadness and surprise.

In 1990 Ekman extended his list of basic emotion to include other emotions: *Amusement, contempt, contentment, embarrassment, excitement, guilt, pride in achievement, relief, satisfaction, sensory pleasure, and shame*

Attention

Ekman's model does not adequately reflect the real-world complexity of emotions. Today, we favour continuous models like valence-arousal, or Pleasure-Arousal-Dominance (PAD), or discuss more general *facial expressions*.

CLASSIFICATION

CLASSIFICATION

Social signal processing often relies on **classification.**

- **Classification** is deciding on which **category** a new observation belongs to based on training data.
- The training data contains data and known categories.
- Classification is a **supervised** learning algorithm.

CLASSIFICATION

Social signal processing often relies on **classification.**

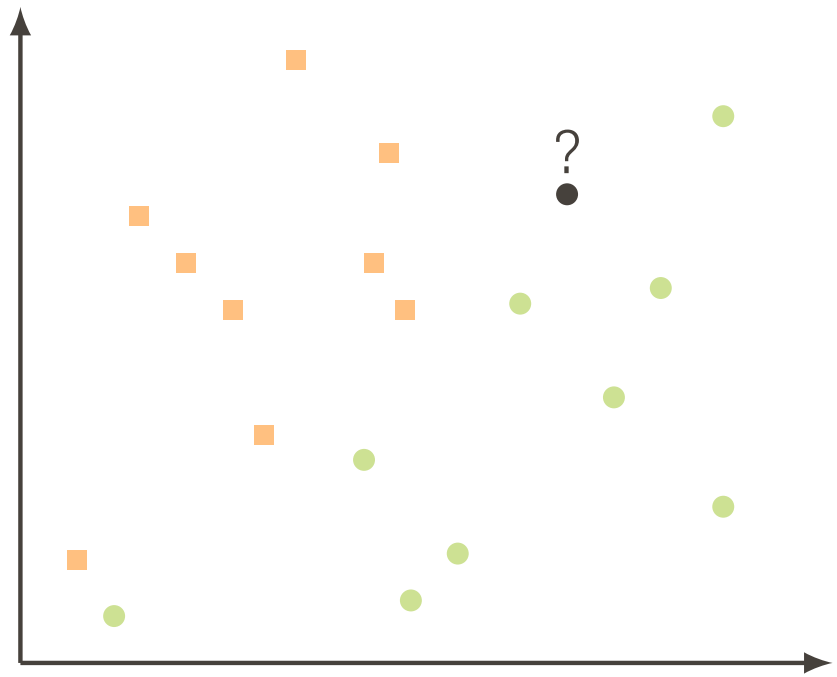
- **Classification** is deciding on which **category** a new observation belongs to based on training data.
- The training data contains data and known categories.
- Classification is a **supervised** learning algorithm.

Examples:

- An incoming tweet needs to classified as being positive or negative.
- An radar ping of a flying objects needs to be classified as belonging to one of ***n*** possible planes.
- The prosody of speech needs to be classified as belonging to one of six basic categories of emotion (happy, sad, angry, bored, surprised, neutral).
- A gesture filmed through a camera needs to be classified as meaning stop, go, left or right.

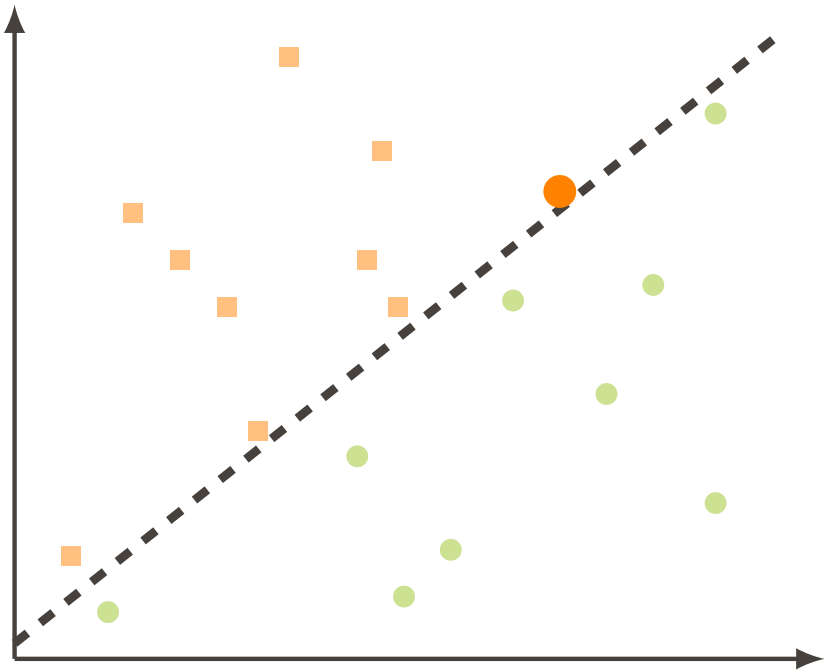
CLASSIFICATION: EXAMPLE

- Two dimensional problem
- Two categories, with training data for both categories
- To which category does a new observation belong?



CLASSIFICATION: EXAMPLE

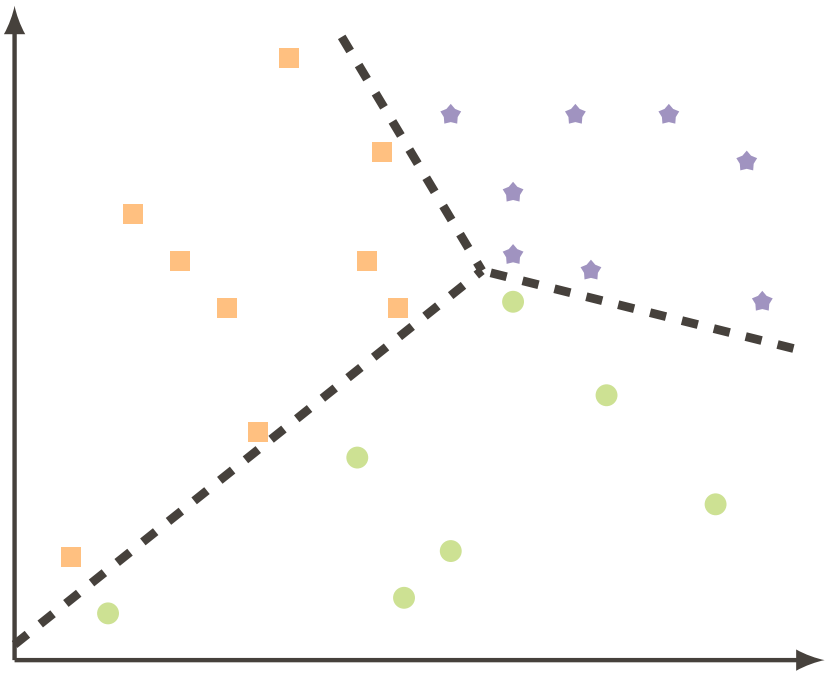
- Two dimensional problem
- Two categories, with training data for both categories
- To which category does a new observation belong?



When categories can be linearly separated, we have a very easy classification problem.

CLASSIFICATION: EXAMPLE

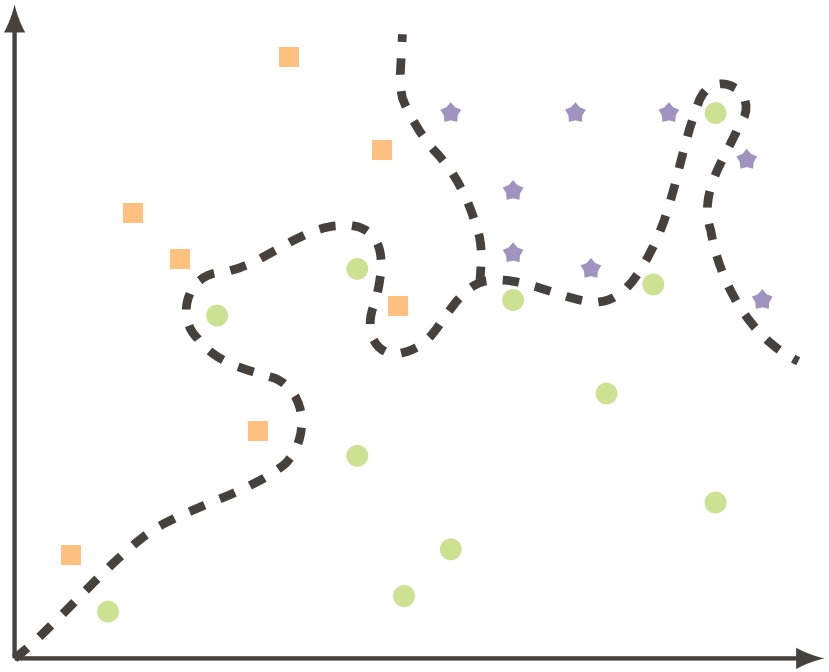
- Two dimensional problem
- Two categories, with training data for both categories
- To which category does a new observation belong?



Three or more can still be linearly separated.

CLASSIFICATION: EXAMPLE

- Two dimensional problem
- Two categories, with training data for both categories
- To which category does a new observation belong?



But what if categories are not linearly separable?

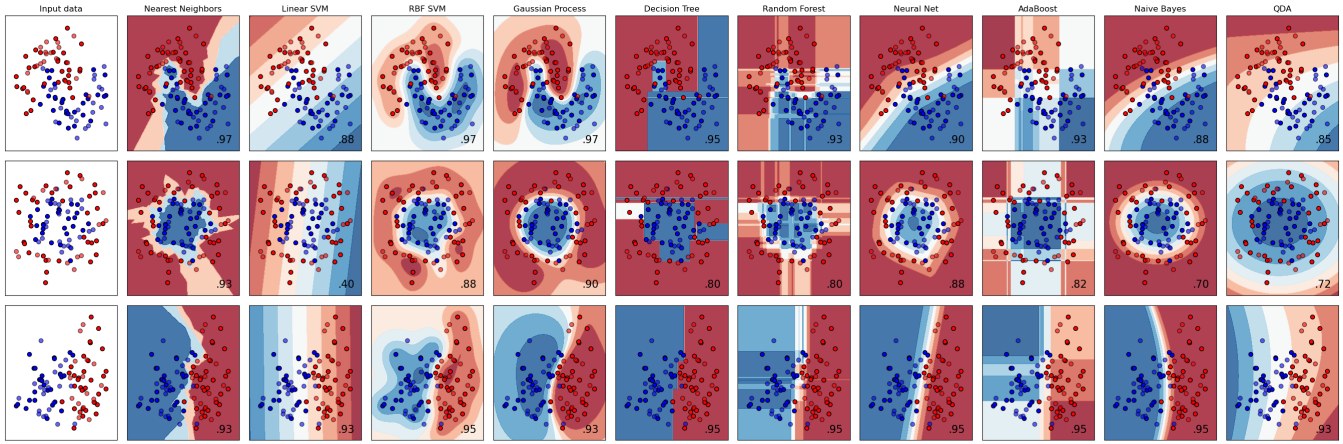
THREE CLASSIFICATION METHODS

Three methods will be explained here:

- *k*-nearest Neighbours (kNN)
- Support Vector Machines (SVM)
- Multi-layer perceptrons (MLP)

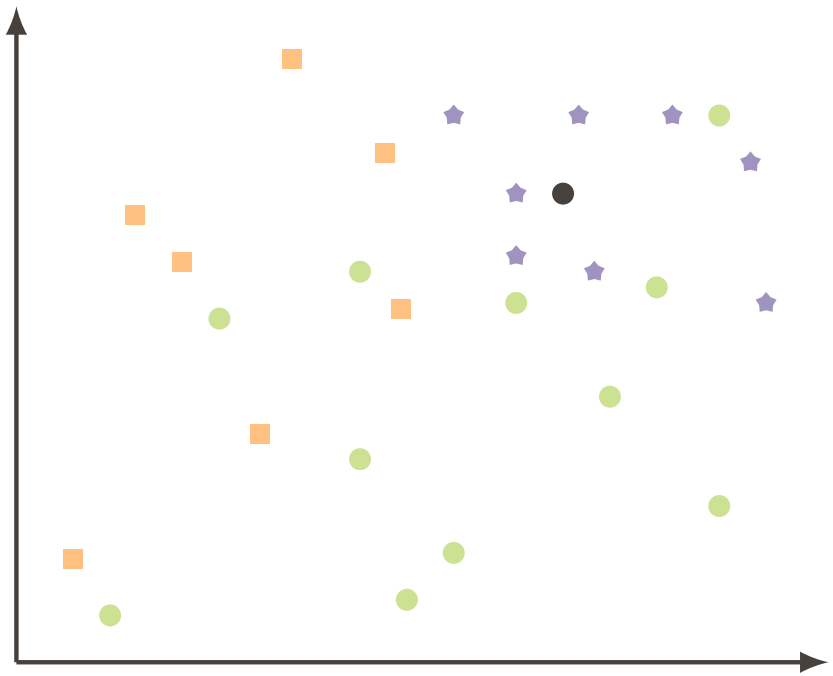
Many other classifiers are possible (Comparison of classifiers with scikit-learn):

- Decision trees/Random forest
- Bayes classifiers
- The large family of neural networks (MLP is the simplest one)
- ...



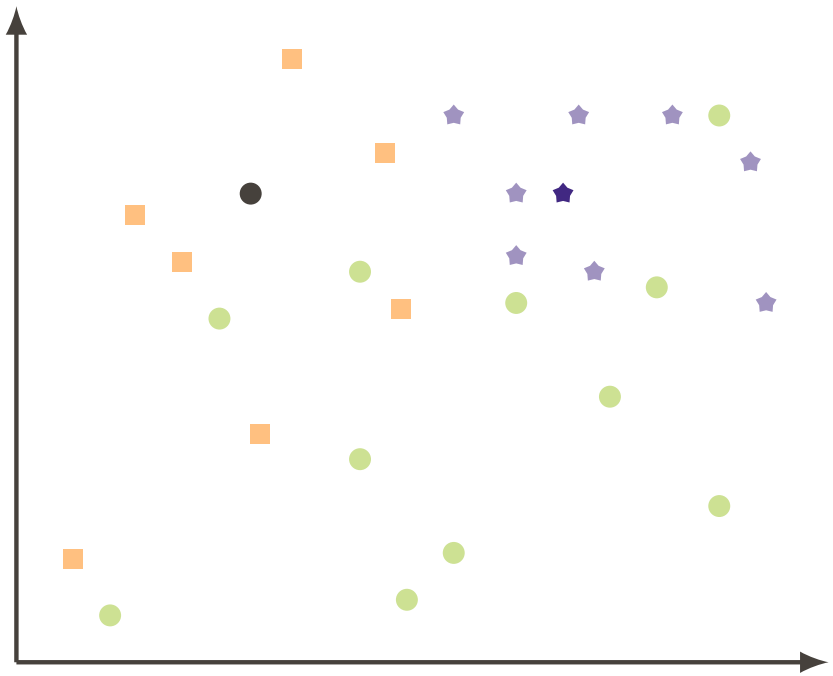
K-NEAREST NEIGHBOURS

When an observation comes in, calculate the distance to the *k* nearest neighbours. The observation belongs to the class the most frequent amongst neighbours.



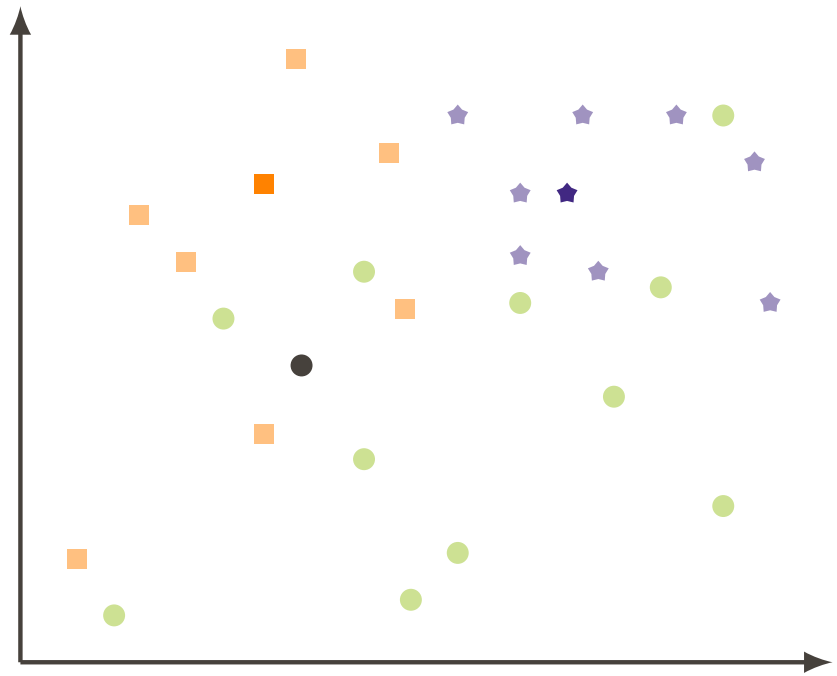
K-NEAREST NEIGHBOURS

When an observation comes in, calculate the distance to the *k* nearest neighbours. The observation belongs to the class the most frequent amongst neighbours.



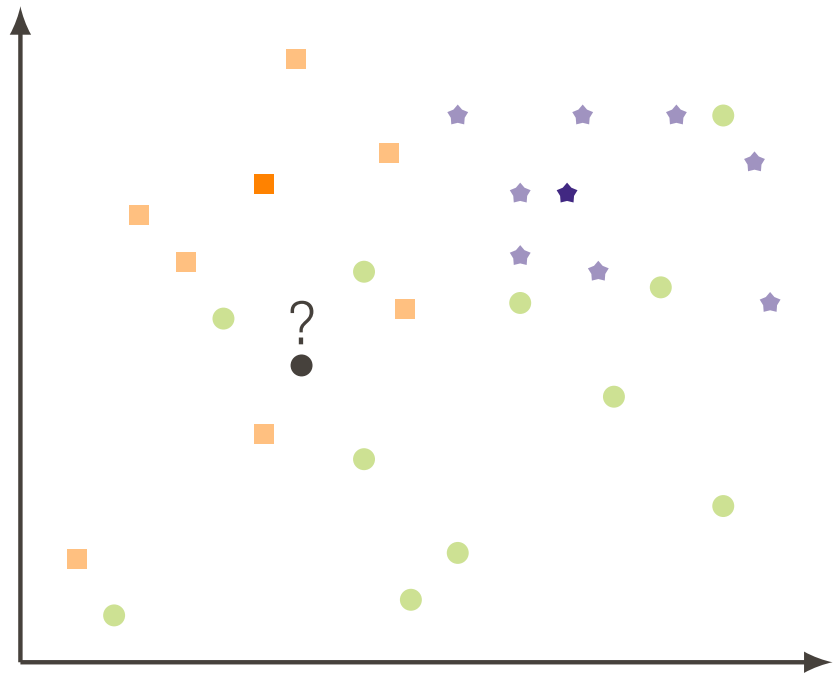
K-NEAREST NEIGHBOURS

When an observation comes in, calculate the distance to the *k* nearest neighbours. The observation belongs to the class the most frequent amongst neighbours.



K-NEAREST NEIGHBOURS

When an observation comes in, calculate the distance to the *k* nearest neighbours. The observation belongs to the class the most frequent amongst neighbours.



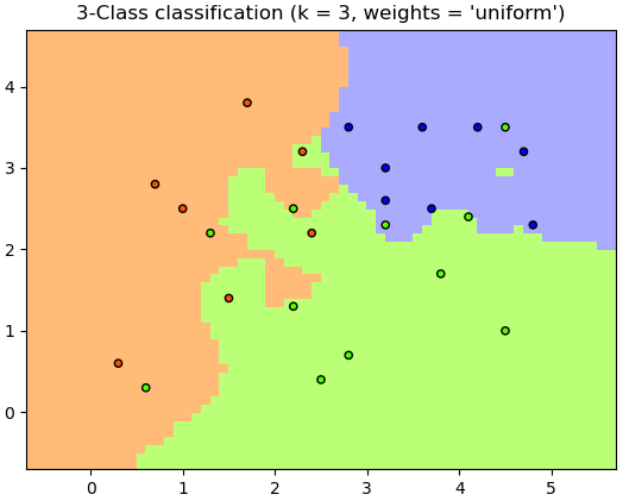
PYTHON IMPLEMENTATION OF *K*-NEAREST NEIGHBOURS

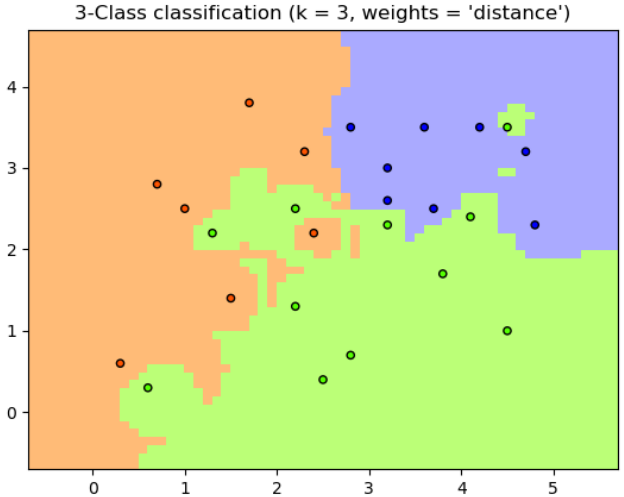
Complete example, with the data above:

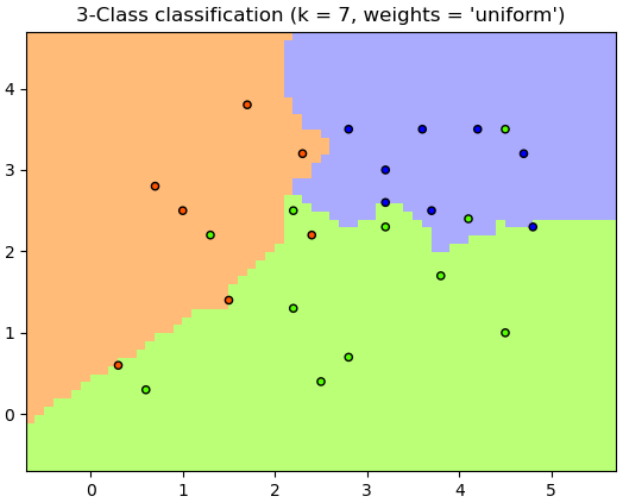
```
""" data.csv:
2.2,1.3,0 -> circles
3.2,2.3,0
...

2.3,3.2,1 -> squares
0.3,0.6,1
...

2.8,3.5,2 -> stars
3.2,2.6,2
...
"""
```

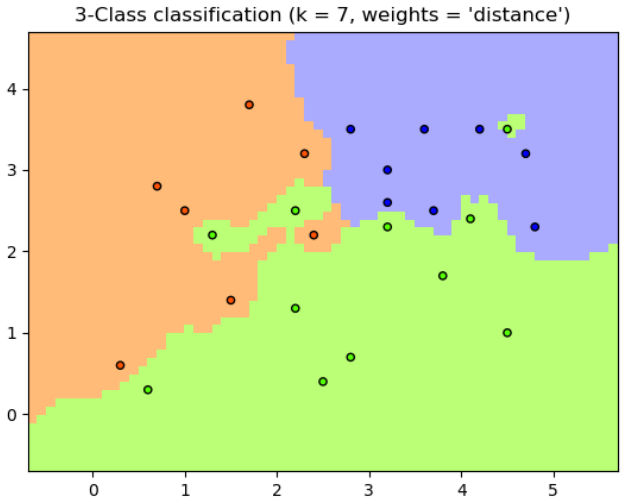
```
from numpy import genfromtxt
from sklearn import neighbors

csv = genfromtxt('data.csv', delimiter=',')
data = csv[:,:2]
categories = csv[:,2]

inputs = [ [3.5,3], [1.5,3], [1.8,1.9] ]

k = 3
knns = neighbors.KNeighborsClassifier(k)
knns.fit(data, categories)

predictions = knns.predict(inputs)
print(predictions)

>>>  [2.  1.  1.]
```

CHOICE OF *K* AND WEIGHTS









K-NEAREST NEIGHBOURS: SUMMARY

- *k*-nearest Neighbours usually does really well on a large number of classification problem, but can underperform near the classification boundary.

K-NEAREST NEIGHBOURS: SUMMARY

- *k*-nearest Neighbours usually does really well on a large number of classification problem, but can underperform near the classification boundary.
- might struggle with very large datasets, as it needs to calculate the distance to all training data every time you classify a new observation (workarounds exist, relying on approximate distances and optimisation of the algorithmic code)

K-NEAREST NEIGHBOURS: SUMMARY

- *k*-nearest Neighbours usually does really well on a large number of classification problem, but can underperform near the classification boundary.
- might struggle with very large datasets, as it needs to calculate the distance to all training data every time you classify a new observation (workarounds exist, relying on approximate distances and optimisation of the algorithmic code)
- commonly, the neighbours are weighted with the inverse of the distance $\frac{1}{d}$ to the observation (i.e. closer neighbours have stronger influence). With `sklearn`:

```
neighbors.KNeighborsClassifier(k, weights='distance')
```

SUPPORT VECTOR MACHINES

A very popular classification algorithm introduced by Vapnik in 1992.

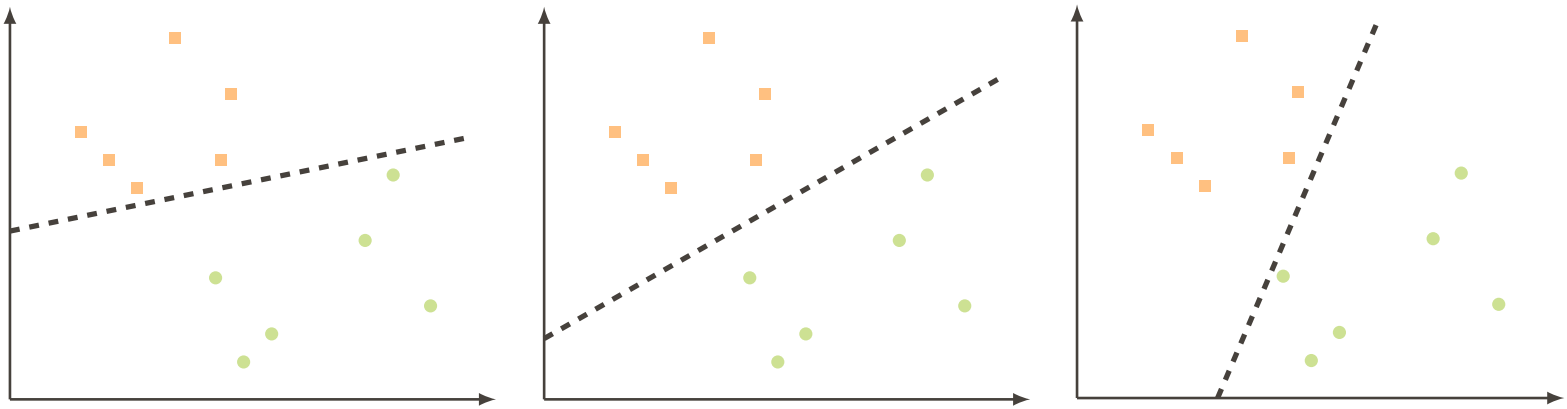
SUPPORT VECTOR MACHINES

A very popular classification algorithm introduced by Vapnik in 1992.

Often (but not always) provides very impressive classification performance on reasonably sized datasets.

SVM PRINCIPLE

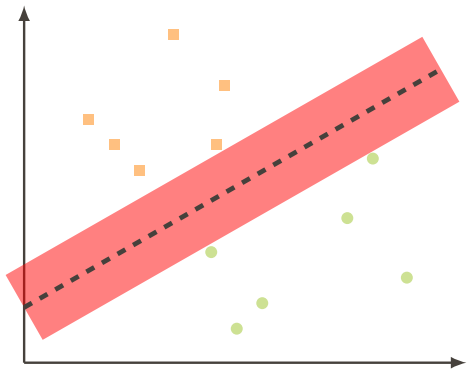
Three different classification lines. All are correct, but is there any reason why one is better than the others?



Intuitively, the line that is most distant to the training data is the best division.

SVM PRINCIPLE

Three different classification lines. All are correct, but is there any reason why one is better than the others?



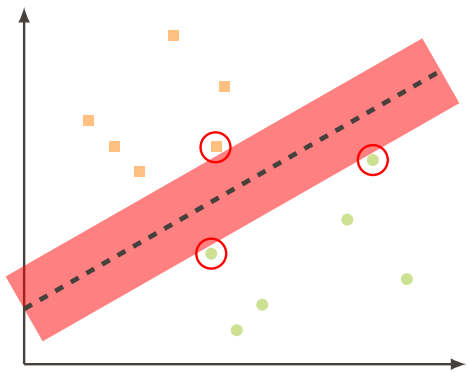
Intuitively, the line that is most distant to the training data is the best division.

The empty area near the division line is symmetric. It forms a bar in 2D space, a cylinder in 3D space, and a hyper-cylinder in n-D space.

SVM PRINCIPLE

The classifier in the middle is called the **maximum margin classifier**.

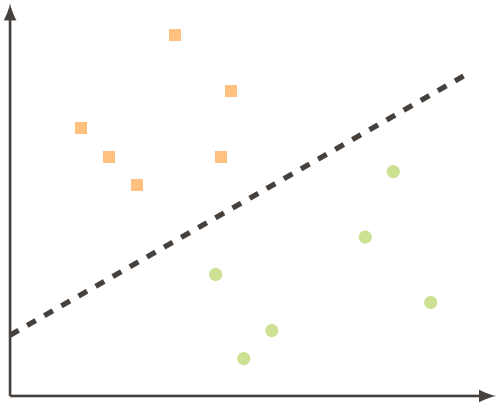
The data points nearest the classifier are called **support vectors**.



Two observations

- The margins should be as large as possible.
- The support vectors are the most useful datapoints because they are the ones that we might get wrong.

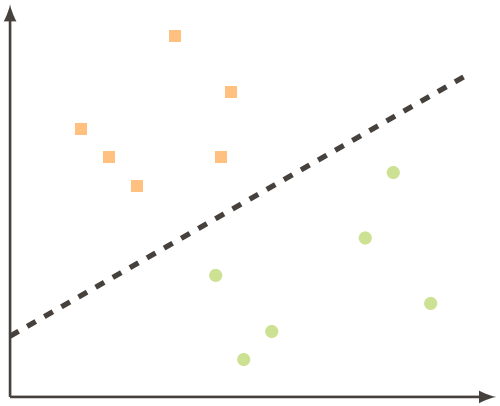
LINEAR CLASSIFIER



A linear classifier for two categories (**binary classifier**) can be written as $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$.

- $\mathbf{w} \cdot \mathbf{x} = \sum_i w_i x_i$. This is called the scalar product or inner product. Can also be written as a matrix multiplication $\mathbf{w}^T \mathbf{x}$.
- $\mathbf{w}$ is a **weight vector**, tilting the line
- $\mathbf{x}$ is a data point (of dimension n)
- b is a **bias**, lifting the line up along the y-axis

LINEAR CLASSIFIER

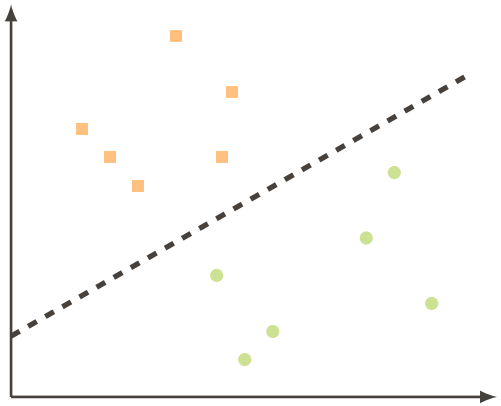


w has to be calculated such that:

$$f(\mathbf{x}) < 0 \Rightarrow \mathbf{x} \in \{\blacksquare\}$$

$$f(\mathbf{x}) > 0 \Rightarrow \mathbf{x} \in \{\circ\}$$

LINEAR CLASSIFIER



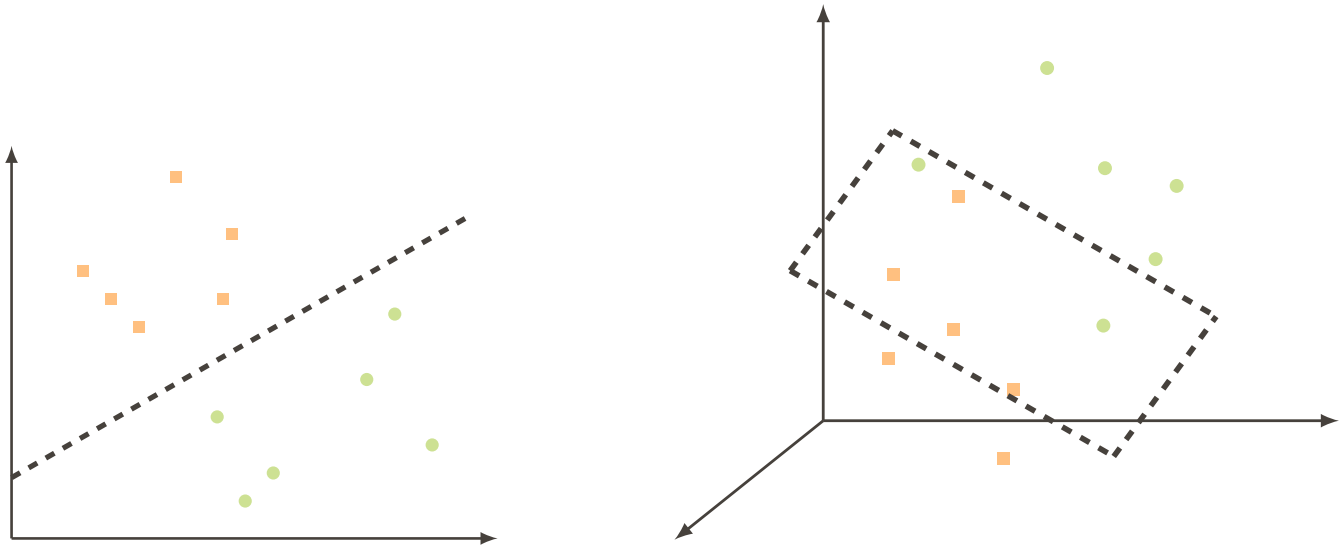
w has to be calculated such that:

$$f(\mathbf{x}) < 0 \Rightarrow \mathbf{x} \in \{\blacksquare\}$$

$$f(\mathbf{x}) > 0 \Rightarrow \mathbf{x} \in \{\circ\}$$

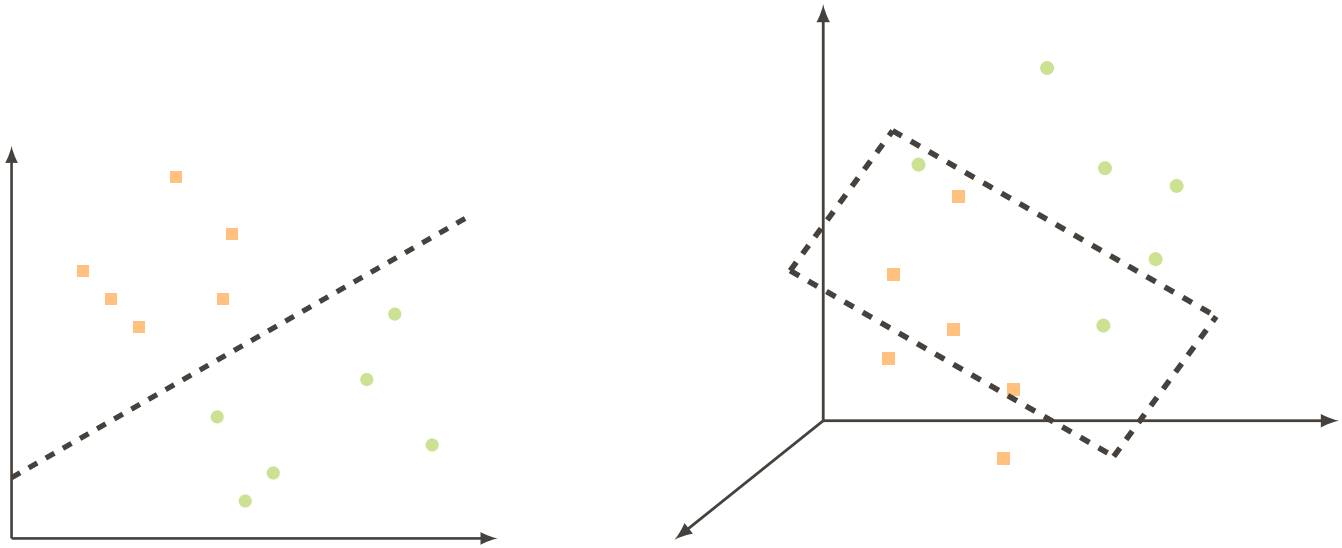
for kNNs, we have to carry along the training data; with a linear classifier, once **w** is found, we can discard the training data

LINEAR CLASSIFIER



- In 2D, the **discriminant** is a line
- In 3D, the discriminant is a plane
- In n-D , the discriminant is a hyperplane

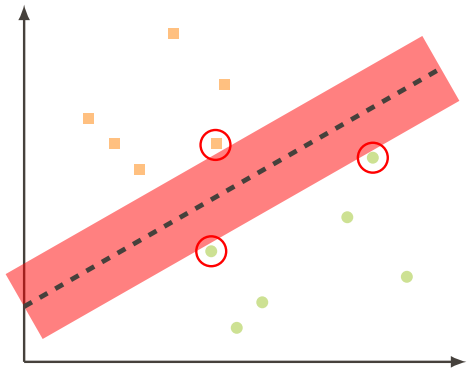
LINEAR CLASSIFIER



- In 2D, the **discriminant** is a line
- In 3D, the discriminant is a plane
- In n-D , the discriminant is a hyperplane

the 'dimensions' are the **features** of our inputs,
e.g. for a human, the size, age, skin colour, gender...

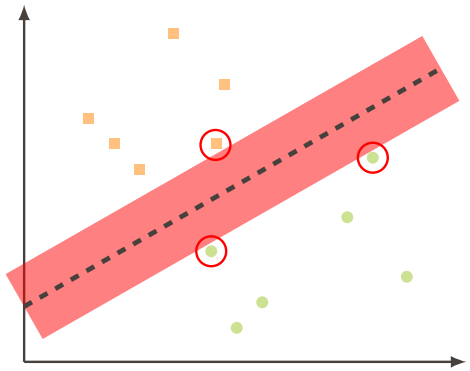
SUPPORT VECTOR CLASSIFIER



In a SVM, we want to **maximise the margin** M. We can rewrite the linear classifier.

If $\mathbf{w}^T\mathbf{x} + b \geq M$, then an observation belongs to ■, for $\mathbf{w}^T\mathbf{x} + b \leq -M$, it belongs to ○.

SUPPORT VECTOR CLASSIFIER



In a SVM, we want to **maximise the margin** M. We can rewrite the linear classifier.

If $\mathbf{w}^T\mathbf{x} + b \geq M$, then an observation belongs to ■, for $\mathbf{w}^T\mathbf{x} + b \leq -M$, it belongs to ○.

A point $\mathbf{x}^{■}$ that lies on the ■ class boundary line, i.e. $\mathbf{w}^T\mathbf{x}^{■} + b = M$, is a **support vector**.

FINDING THE OPTIMAL CLASSIFIER

The problem is to find the **optimal** values for **w** and *b*.

The classifier needs to satisfy two conditions

- It needs to correctly classify the training data,
- and needs the margin from the classifier to be as large as possible.

FINDING THE OPTIMAL CLASSIFIER

The problem is to find the **optimal** values for **w** and *b*.

The classifier needs to satisfy two conditions

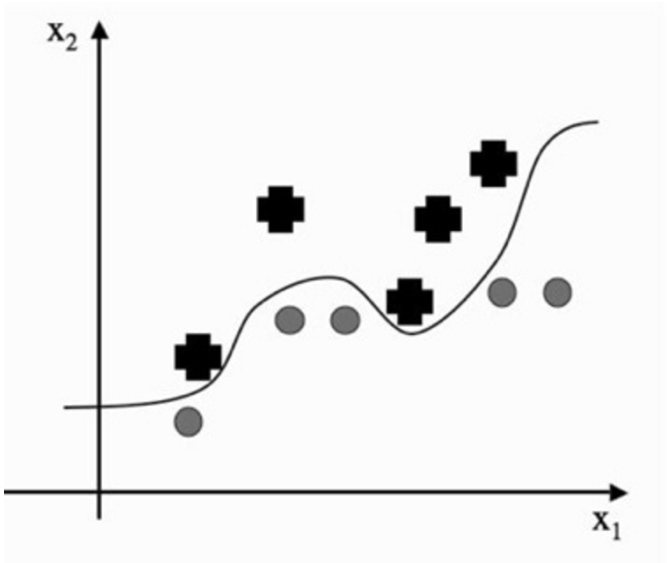
- It needs to correctly classify the training data,
- and needs the margin from the classifier to be as large as possible.

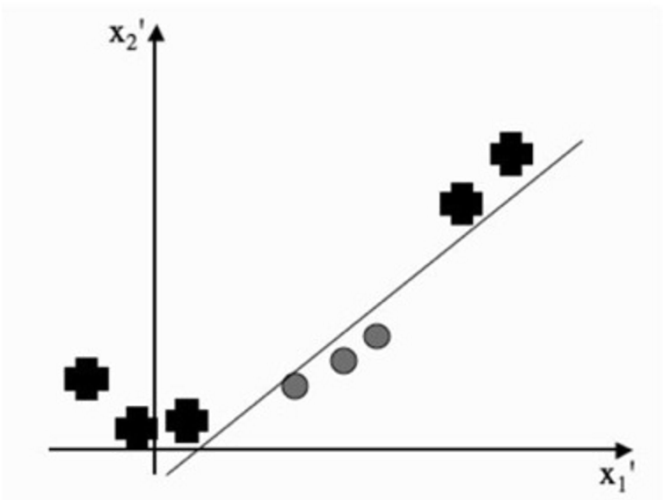
A **quadratic programming solver** is used to find the optimal values for **w** and *b*.

Oxford lecture on SVM to learn more about the mathematical formulation and the *perceptron algorithm*.

TRANSFORMATION OF DATA

- Unfortunately, ***all this still assumes that the data is linearly separable***. But what if it is not, and we are not prepared for a few misclassifications?
- The solution is to **transform** the data: move data points in the n-D space until the training data is linearly separable again





FEATURE MAPS

We can transform data points (= move them) or even add more dimensions using a function $\theta(\mathbf{x}_i)$ from input $\mathbf{x}_i$. θ is a **feature map**.

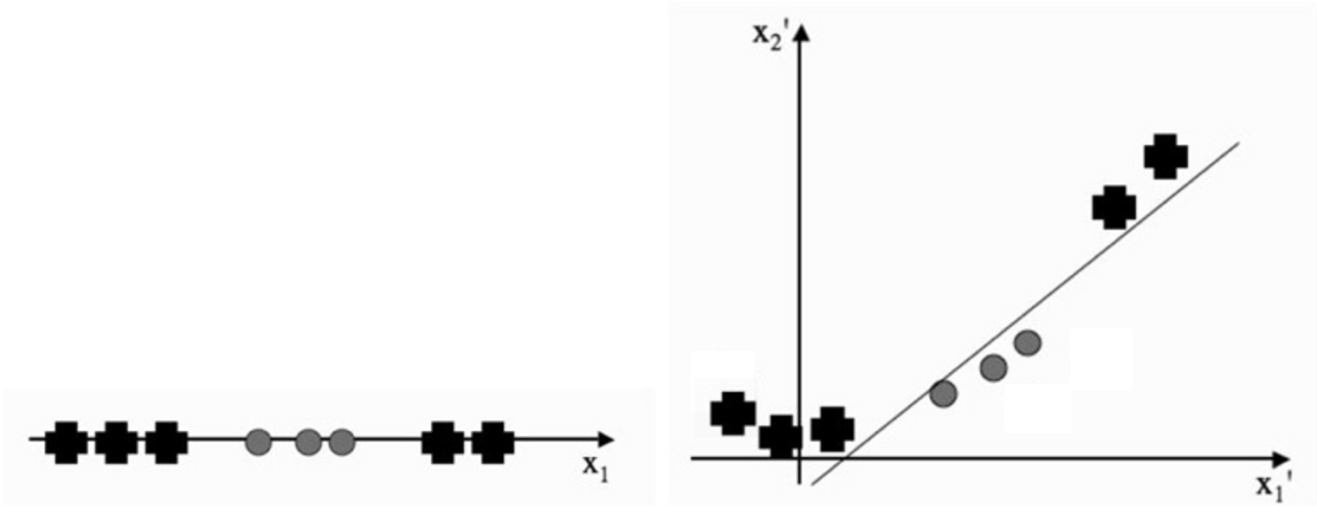
Example:

FEATURE MAPS

We can transform data points (= move them) or even add more dimensions using a function $\theta(\mathbf{x}_i)$ from input $\mathbf{x}_i$. θ is a **feature map**.

Example:

$$\theta(\mathbf{x}_i) = [\mathbf{x}_i, \mathbf{x}_i^2]$$

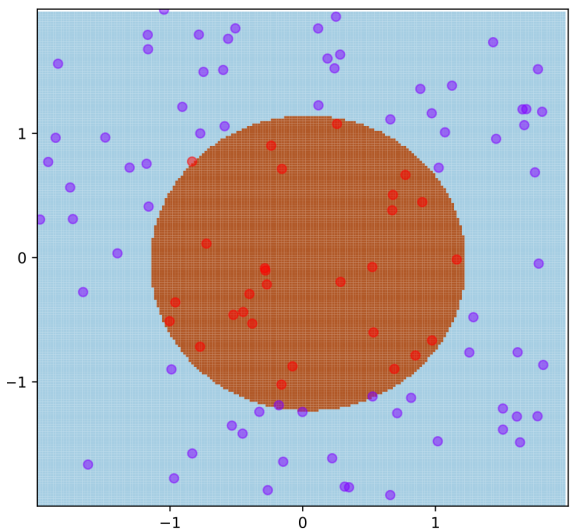


$\rightarrow$ now, linearly separable

FEATURE MAPS

We can transform data points (= move them) or even add more dimensions using a function $\theta(\mathbf{x}_i)$ from input $\mathbf{x}_i$. θ is a **feature map**.

Example:

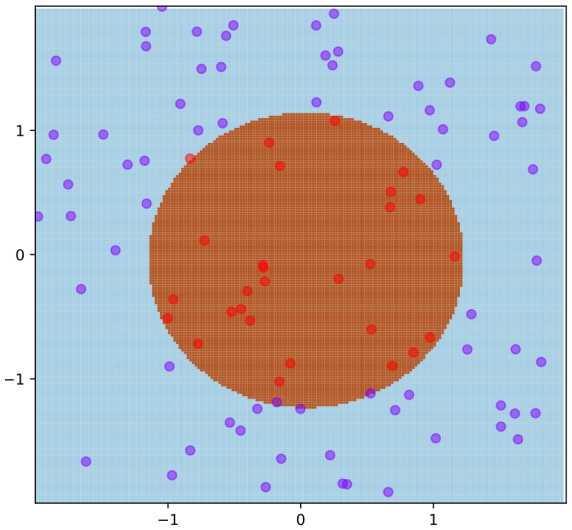


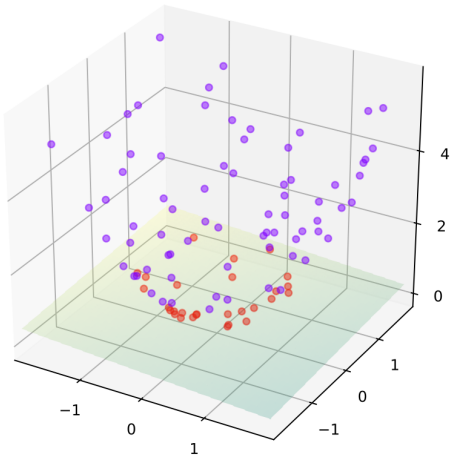
FEATURE MAPS

We can transform data points (= move them) or even add more dimensions using a function $\theta(\mathbf{x}_i)$ from input $\mathbf{x}_i$. θ is a **feature map**.

Example:

$$\theta(< x_i, y_i >) = [x_i, y_i, x_i^2 + y_i^2]$$





$\rightarrow$ now, linearly separable

FEATURE MAPS

If we know something about the structure of the data, then we might be able to identify feature maps that would be effective.

Often however we do not have such domain knowledge.

THE KERNEL TRICK

$$\theta : \mathbf{x} \rightarrow \theta(\mathbf{x}), \mathbb{R}^d \rightarrow \mathbb{R}^D$$

$$f(\mathbf{x}) = \mathbf{w}^T \theta(\mathbf{x}) + b$$

- Simply map $\mathbf{x}$ to $\theta(\mathbf{x})$ where data is separable
- Solve for $\mathbf{w}$ in high dimensional space $\mathbb{R}^D$
- If $D >> d$ then there are many more parameters to learn for $\mathbf{w}$. Can this be avoided?

THE KERNEL TRICK

$$\theta : \mathbf{x} \rightarrow \theta(\mathbf{x}), \mathbb{R}^d \rightarrow \mathbb{R}^D$$

$$f(\mathbf{x}) = \mathbf{w}^T \theta(\mathbf{x}) + b$$

It can be shown that you do not actually need to calculate explicitely $\theta(\mathbf{x})$. Instead, you only need $\theta(\mathbf{x}_i)^T \theta(\mathbf{x}_j)$ (with $\mathbf{x}_i$ the training data) to compute $\mathbf{w}$.

We call the **kernel function** $K(\mathbf{x}_i, \mathbf{x}_j) = \theta(\mathbf{x}_i)^T \theta(\mathbf{x}_j)$. The SVM classifier can be learnt and applied with only K, and the complexity depends only on N (size of training data), not on D.

STANDARD KERNELS

- **Polynomials** up to some degree s in the elements x_k of the input vector (e.g. x_3^3 or $x_1 x_4$). This can be written as:

$$K(x, y) = (1 + x^T y)^s$$

- **Sigmoid functions** of the x_k with parameters κ and δ, and kernel:

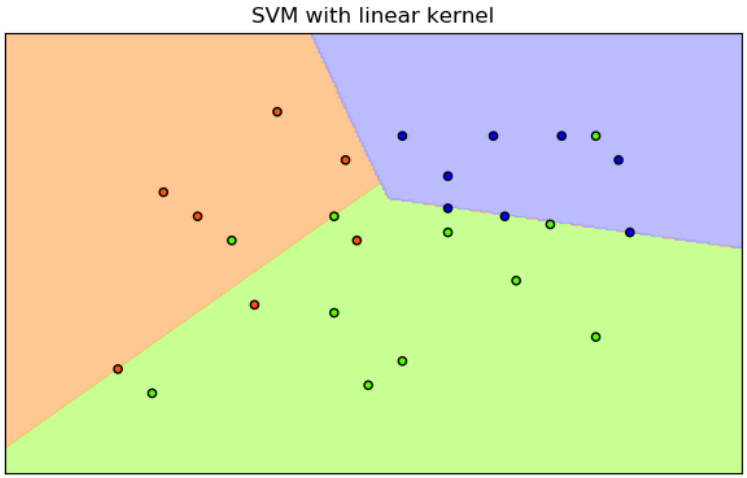
$$K(x, y) = tanh(\kappa x^T y - \delta)$$

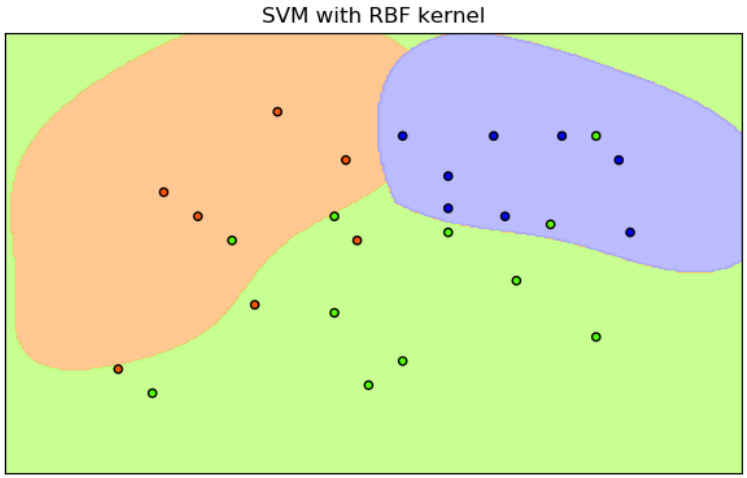
- **Radial basis function** expansions of the x_k with parameter σ and kernel:

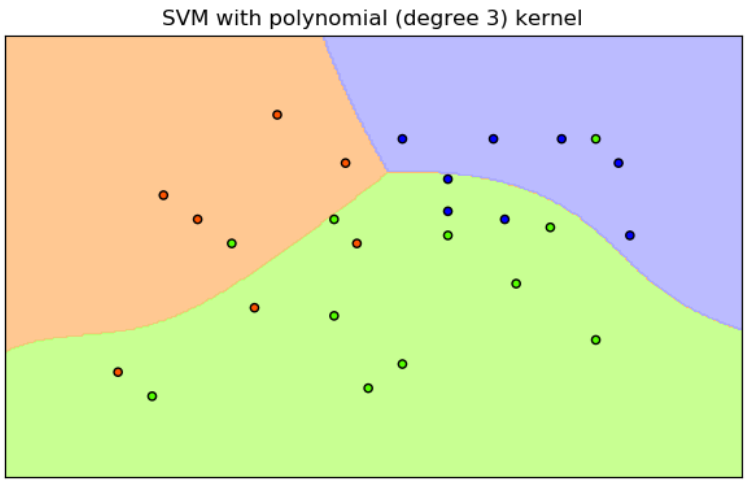
$$K(x, y) = exp(-\frac{(x - y)^2}{2\sigma^2})$$

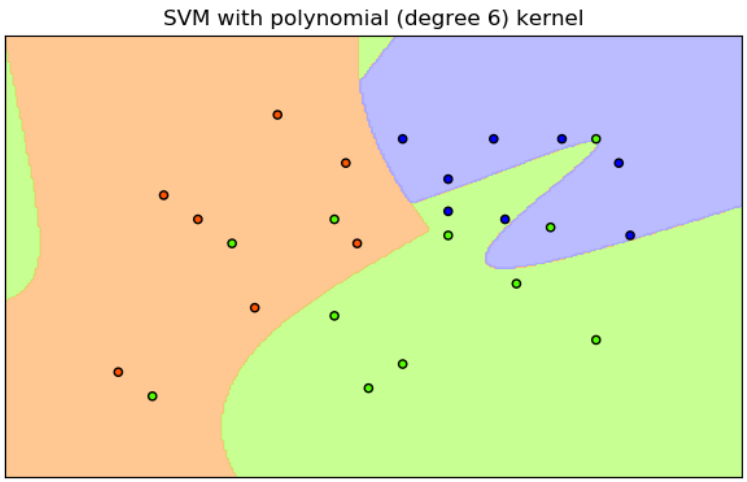
(a Gaussian kernel)

COMPARISON OF KERNEL FUNCTIONS









PYTHON IMPLEMENTATION OF SUPPORT VECTOR MACHINE

```
from sklearn import svm

C = 1.0  # SVM regularization parameter
         # +- 'how acceptable are misclassification'
clf = svm.SVC(kernel='linear', C=C) # kernel='poly', 'rbf', 'sigmoid'...

clf.fit(data, categories)

predictions = clf.predict(inputs)
```

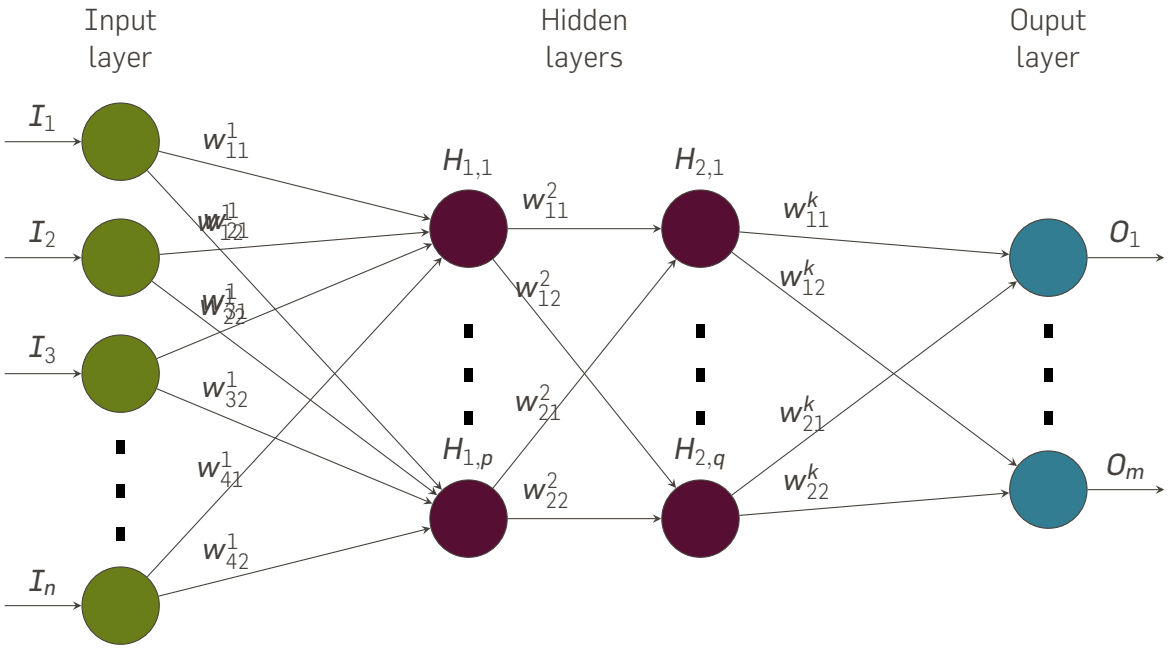
PYTHON IMPLEMENTATION OF SUPPORT VECTOR MACHINE

Complete example, with our data:

```
""" data.csv:
2.2,1.3,0 -> circles
3.2,2.3,0
...

2.3,3.2,1 -> squares
0.3,0.6,1
...

2.8,3.5,2 -> stars
3.2,2.6,2
...
"""
```

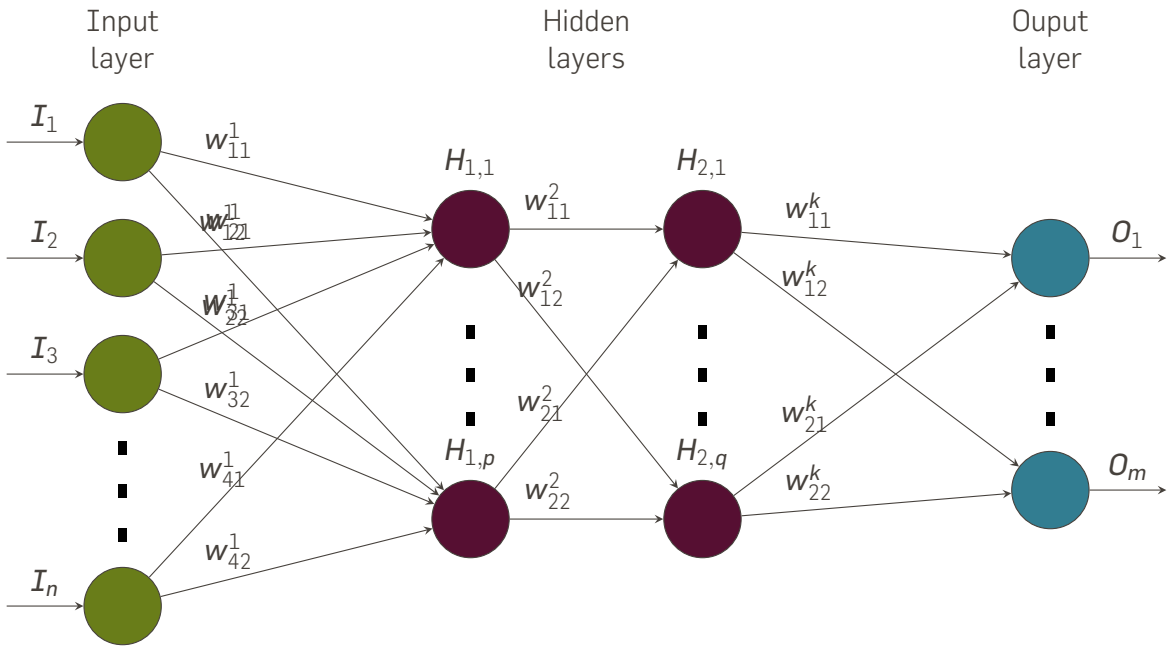
```
from numpy import genfromtxt
from sklearn import svm

csv = genfromtxt('data.csv', delimiter=',')
data = csv[:,:2]
categories = csv[:,2]


inputs = [ [3.5,3], [1.5,3], [1.8,1.9] ]

clf = svm.SVC(kernel='rbf',
              gamma = 0.7,
              C=1.0)
clf.fit(data, categories)

predictions = clf.predict(inputs)
print(predictions)

>>>  [2.  1.  0.]
```

MULTI-LAYER PERCEPTRONS (MLP)



- ***fully connected feed-forward*** artificial neural network
- one ***input layer***, n (n>=1) ***hidden layers***, one ***output layer***
- between each layer, a non-linear ***activation function*** (often sigmoid or ReLU)

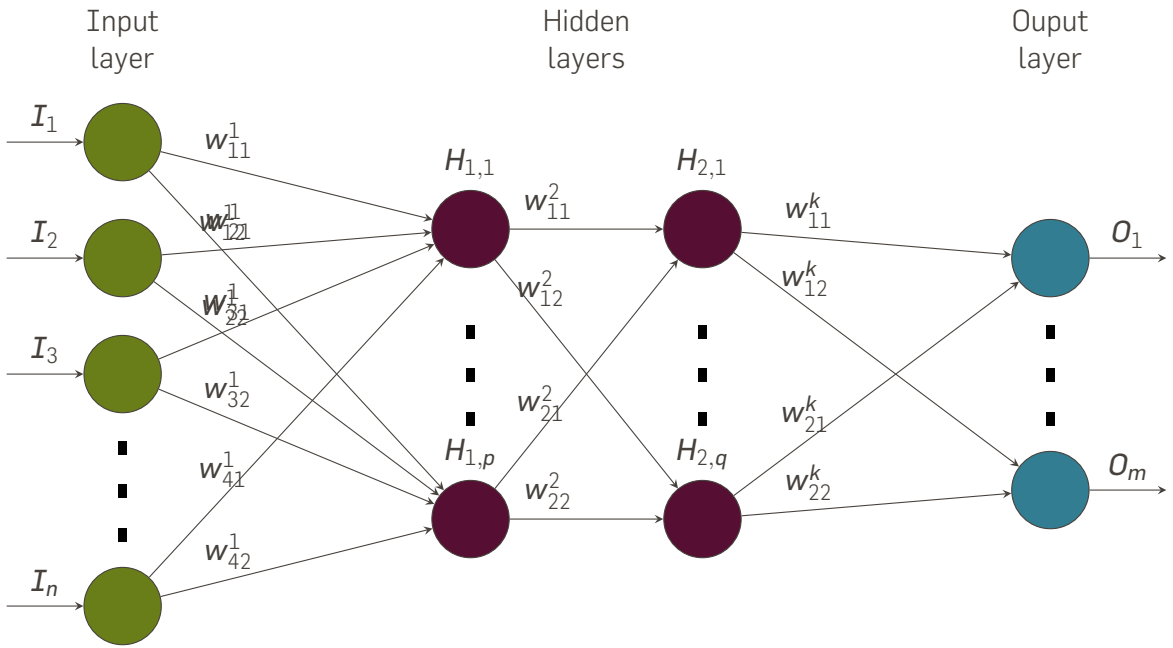
MULTI-LAYER PERCEPTRONS (MLP)



In our case:

- 2 coordinates: size ***n*** of the input layer: 2
- 3 categories: size ***m*** of the output layer: 1 (or 3 if using ***one-hot encoding*** for categories)
- nb and size of hidden layers? **hyper-parameters**. Need to be determined as well! ($\rightarrow$ ***validation set***)

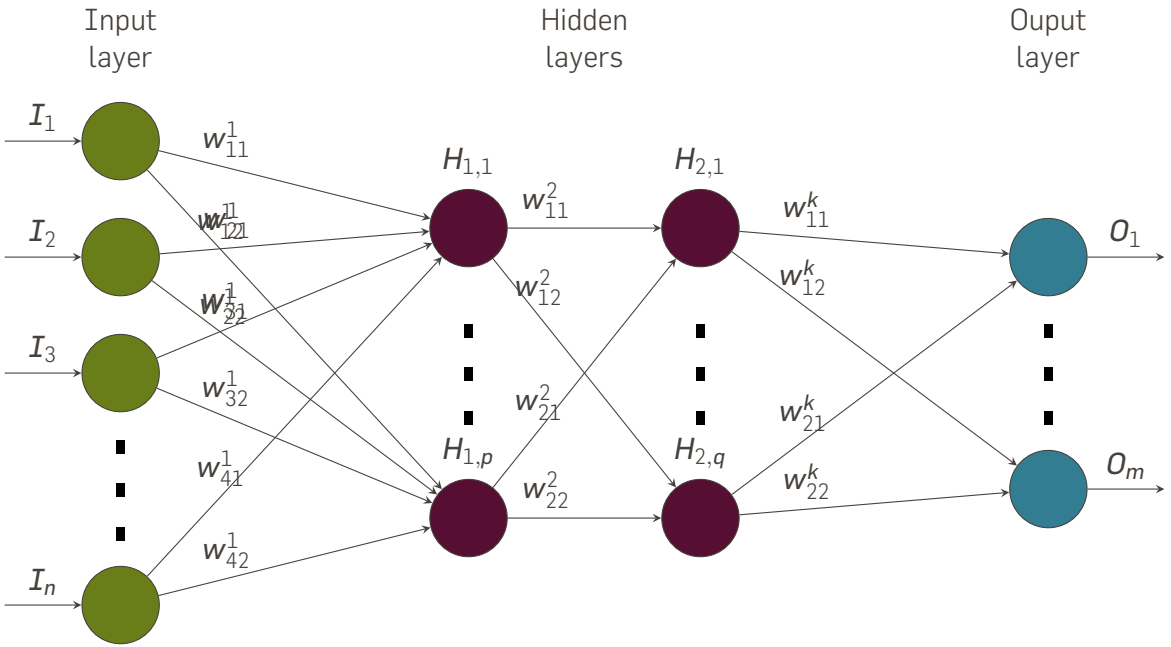
MULTI-LAYER PERCEPTRONS (MLP)



For each layer k: $\mathbf{x^k} = f(\mathbf{W}^T\mathbf{x^{k-1}} + b)$

- $\mathbf{x^{k-1}}$ the input vector of this layer
- $\mathbf{x^k}$ the output vector
- $\mathbf{W}$ the set of parameters or weights
- b the bias
- f the activation function (eg ReLU, sigmoid)

MULTI-LAYER PERCEPTRONS (MLP)



Training $\Rightarrow$ finding weights w^k_{ij} so that the outputs values match 'what we expect'.

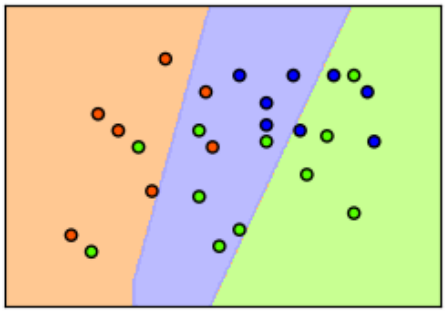
Main algorithm: ***backpropagation*** using ***gradient descent***

PYTHON IMPLEMENTATION OF MULTI-LAYER PERCEPTRONS

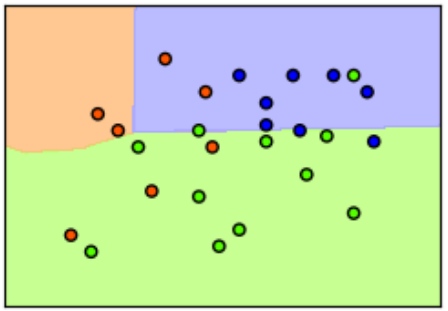
```
from numpy import genfromtxt

from sklearn.neural_network import MLPClassifier

csv = genfromtxt('data.csv', delimiter=',')
data = csv[:,:2]
categories = csv[:,2]

inputs = [ [3.5,3], [1.5,3], [1.8,1.9] ]

clf = MLPClassifier(hidden_layer_sizes=(4,4),
                    activation = "relu")
clf.fit(data, categories)

predictions = clf.predict(inputs)
print(predictions)

>>>  [2.  2.  0.]
```

MLP HYPER-PARAMETERS

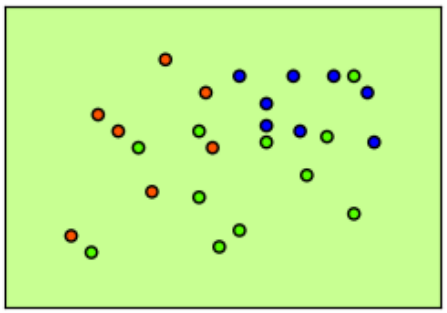
MLP, 1 hidden layer, 4 hidden units



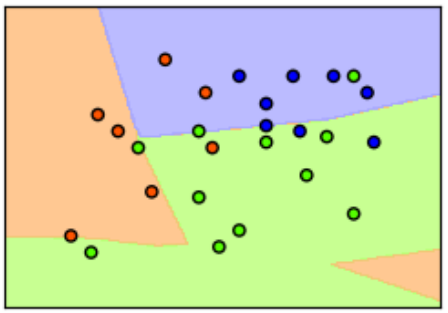
MLP, 2 hidden layer, 4 hidden units



MLP, 3 hidden layer, 4 hidden units

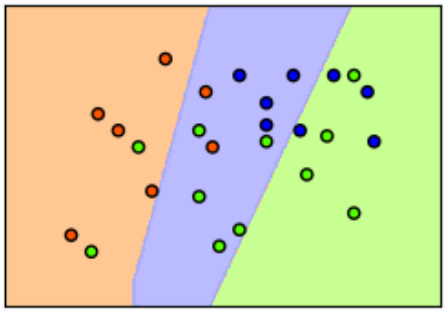


MLP, 1 hidden layer, 8 hidden units

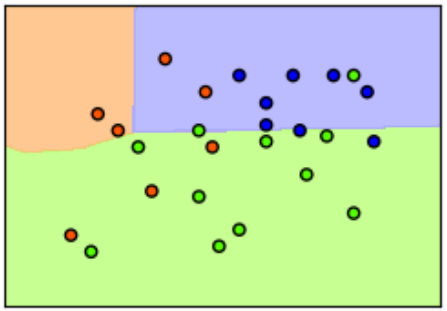


MLP HYPER-PARAMETERS

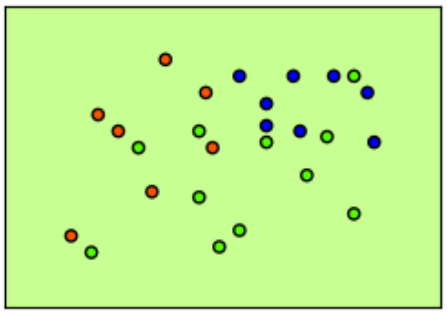
MLP, 1 hidden layer, 4 hidden units



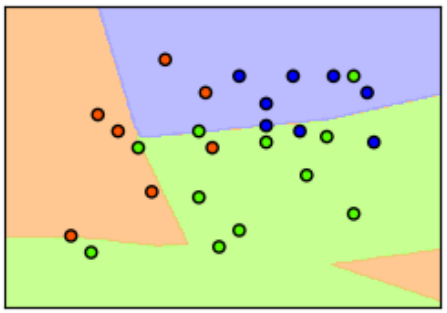
MLP, 2 hidden layer, 4 hidden units



MLP, 3 hidden layer, 4 hidden units



MLP, 1 hidden layer, 8 hidden units



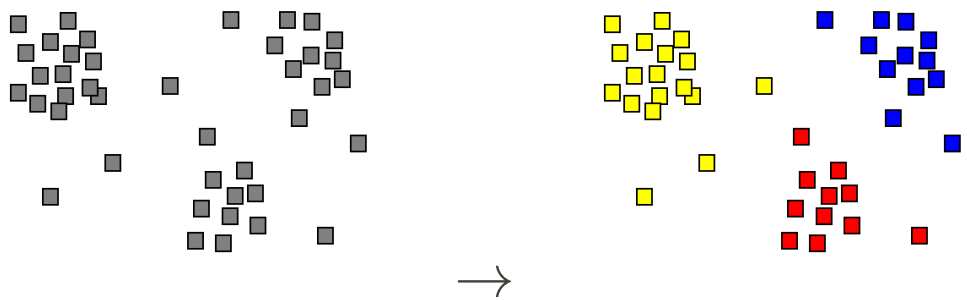
MLP does not perform well on this example: not enough data to converge!

CLUSTERING VS CLASSIFICATION

Clustering is sometimes confused with **classification**. (often because some algorithms have similar names,e.g. *k-means clustering* and *k-nearest neighbours*).

Clustering starts from unlabelled data points and tries to find *k* clusters in the data → **unsupervised learning**.

→ Used to find patterns in the data.



Classification starts from training data (→ **supervised learning**), and attempts to correctly classify an unknown observation.

LET'S BUILD OUR OWN EMOTION CLASSIFIER FROM SCRATCH

SHOPPING LIST

What do we need?

SHOPPING LIST

What do we need?

- a dataset of labelled faces

SHOPPING LIST

What do we need?

- a dataset of labelled faces
- some pre-processing to normalise the faces

SHOPPING LIST

What do we need?

- a dataset of labelled faces
- some pre-processing to normalise the faces
- features to extract

SHOPPING LIST

What do we need?

- a dataset of labelled faces
- some pre-processing to normalise the faces
- features to extract
- a classifier

Google

human face happiness

16

All | Images | Videos | Shopping | News | More | Settings | Tools | View saved | SafeSearch

smile | cartoon | male | boy | grandpa | black and white | real | happiness | joy | sad | gratitude | beauty | emotional | smiley







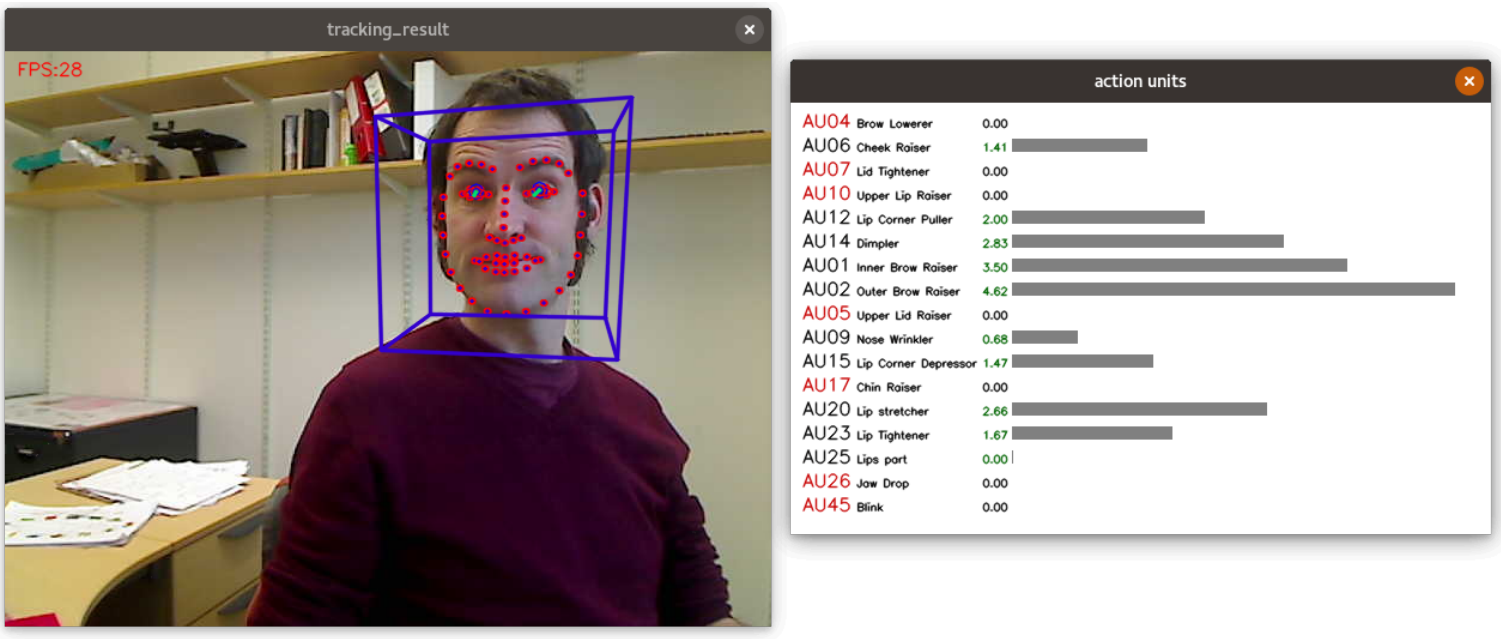




Disgust
Closed mouth

ACTION UNITS

Our features: **action units**.



github.com/TadasBaltrusaitis/OpenFace

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	emotion	AU01	AU02	AU04	AU05	AU06	AU07	AU09	AU10	AU12	AU14	AU15	AU17	AU20	AU23	AU25	AU26	AU45	face
2	0	1.07	0	0.05	2.95	0	0	0	0	0	0	0	0	0	0	1.36	0	0	fear/processed/aligned/fear-041_aligned.bmp
3	0	1.57	0	0.63	0	1.02	1.39	0	0.45	0	0	0	0	0	0	0.45	1.35	0	fear/processed/aligned/fear-053_aligned.bmp
4	0	1.38	0	1.17	0	1.36	0	1.85	1.21	0	0	0	0	0.05	0	0	0	0	fear/processed/aligned/fear-035_aligned.bmp
5	0	0	0.97	0.87	2.48	0	0	0	0	0	0	0.31	0	0.71	0.32	1.41	0.24	0	fear/processed/aligned/fear-018_aligned.bmp
6	0	3.79	2.52	0	4.61	0	0	0	0	0	0	0	0	0	0	1.53	0.67	0	fear/processed/aligned/fear-043_aligned.bmp
7	0	2.55	0	0	1.2	1.58	0	0	1.27	0.64	0	0.93	1.14	0	0	0	0	0	fear/processed/aligned/fear-044_aligned.bmp
8	0	1.93	2.72	0	3.86	0	0	0	0	0	0	1.38	0.67	1.23	0	0.15	0.83	0	fear/processed/aligned/fear-006_aligned.bmp
75	1	0	1	0	0	1.79	3.94	0	2.96	2.62	0	0.79	0	0	0.36	0	0	1.08	happiness/processed/aligned/happiness-098_aligned.bmp
76	1	0	0	0	1.78	0	0	0	0	0	0	0	0	0	0	2.14	1.55	0	happiness/processed/aligned/happiness-093_aligned.bmp
77	1	0	0	0	0	2.17	2.74	1.8	1.83	1.91	1.23	0	0	0.61	0	3.24	0	0.04	happiness/processed/aligned/happiness-095_aligned.bmp
78	1	0	0	2.91	0	1.38	0	2.71	2.11	1.04	1.38	0.15	0.71	1.4	0.23	0.56	0	0.65	happiness/processed/aligned/happiness-046_aligned.bmp
79	1	0.51	0	0	0	1.23	2.38	0.82	0.33	2.21	0.99	0.03	0	0	0	1.5	0.39	0.07	happiness/processed/aligned/happiness-025_aligned.bmp
80	1	0.51	0.76	0	0	1.56	0	0	1.23	2.56	1.12	0	0	0	0	0	0	0	happiness/processed/aligned/happiness-024_aligned.bmp
81	1	0	0	0	0	3.22	3.45	0	2.84	3.37	0	0	0	0	0	0	0	0	happiness/processed/aligned/happiness-075_aligned.bmp
196	2	0	0.1	3.45	1.14	0	0	1.75	0.65	0	0	0.11	1.04	2.05	0	0.67	0	0.12	anger/processed/aligned/anger-051_aligned.bmp
197	2	0	0.92	0	0	0.43	1.79	0	0.24	0	0	0	0	0	0	0	0	0	anger/processed/aligned/anger-027_aligned.bmp
198	2	1.27	0	2.58	0	0	0	0	0.39	0	0	1.35	2.39	1.28	0.28	0	0	0	anger/processed/aligned/anger-057_aligned.bmp
199	2	4.5	1.72	0	1.24	1.54	0	0	1.63	0	0	0.22	0	0	0	2.29	0	0	anger/processed/aligned/anger-047_aligned.bmp
200	2	0	0	0	0.29	0	0	0.56	0	0	0	0	0	0	0	1.5	0	0	anger/processed/aligned/anger-008_aligned.bmp
201	2	0	0	0	0.93	0.86	0	0.8	1.35	0.85	0	0	0	0	0.5	0	0	0	anger/processed/aligned/anger-031_aligned.bmp
202	2	2.73	0	0	1.11	1.55	0.32	3.07	1.75	1.21	1.34	1.15	0	0	0.03	2.07	0	0	anger/processed/aligned/anger-019_aligned.bmp
203	2	0	1.1	3.59	0	2.33	3.16	2.56	3.38	0	1.34	0	0	1.92	0	3.03	0	0	anger/processed/aligned/anger-016_aligned.bmp
280	3	0	0	0	0.33	0	0	0	0	0	0	0	0	0	0.59	0	0	0	surprise/processed/aligned/surprise-077_aligned.bmp
281	3	0.34	0	0	1.38	1.24	0	0	1.48	2.05	0	0.57	0	0	0.12	3.27	0	0	surprise/processed/aligned/surprise-079_aligned.bmp
282	3	1.44	0	0	0.59	0	0	0	1.66	0.76	0	0.49	0	0	0.08	1.31	0	0	surprise/processed/aligned/surprise-040_aligned.bmp
283	3	0.24	0	0.9	1.43	0	0.89	0	1.23	0	0	1.29	0	0.56	0.11	0	0	0	surprise/processed/aligned/surprise-004_aligned.bmp
284	3	0.95	0	0	0	0	0	1.04	0	0	0.13	0	1.07	2.53	0.36	0	0	0	surprise/processed/aligned/surprise-062_aligned.bmp
285	3	0.69	0.91	0	1.87	0	0	0	1.34	0	0	1.09	0	0	0	2.94	1.45	0	surprise/processed/aligned/surprise-005_aligned.bmp
286	3	1.19	0.68	0	2.03	0.33	0	0	0.58	1.73	1.65	0.22	0	0	0	1.92	0.7	0	surprise/processed/aligned/surprise-029_aligned.bmp
287	3	3.88	2.83	0	4.25	0	0	0	0.52	1.14	0	1.09	0	0	0.1	1.49	1.64	0	surprise/processed/aligned/surprise-017_aligned.bmp
288	3	0	0	0	1.48	1.46	0.79	0	2.65	1.39	0	1.08	0.4	0	0.89	0.11	0	0	surprise/processed/aligned/surprise-080_aligned.bmp
289	3	0	0	0.79	1.62	0	2.49	0	0	0	0	0	0	0	0.01	1.62	1.87	0	surprise/processed/aligned/surprise-060_aligned.bmp
290	3	1.87	0.79	0	2.75	0	0	0	0	0	0	0	0	0	0.56	1.76	0.58	0	surprise/processed/aligned/surprise-002_aligned.bmp
368	4	0	0	1.58	0.84	0	0	0	0	0	0.59	0	0	0	0	0	0	0	sadness/processed/aligned/sadness-025_aligned.bmp

OUR TINY 'EMOTIONS' DATASET

- A couple of Google queries
- a bit of manual filtering
- 558 faces, between 70 and 130 per emotion
- split into 2 datasets: a **training** dataset (50 face per emotion) and a **testing** dataset
- for each face, OpenFace pre-processing to:
  - extract facial landmarks
  - normalise and mask out the faces
  - extract action units

The dataset is available for you to play with on the lecture's GitHub repo.

CLASSIFICATION?

We want to train a classifier with our training dataset, and then check whether the emotions in the test faces can be properly recognised.

What are our options here?

CLASSIFICATION?

We want to train a classifier with our training dataset, and then check whether the emotions in the test faces can be properly recognised.

What are our options here?

1. kNN/SVM/MLP on the action units directly?

CLASSIFICATION?

We want to train a classifier with our training dataset, and then check whether the emotions in the test faces can be properly recognised.

What are our options here?

1. kNN/SVM/MLP on the action units directly?
2. PCA, then kNN/SVM/MLP on the action units?

CLASSIFICATION?

We want to train a classifier with our training dataset, and then check whether the emotions in the test faces can be properly recognised.

What are our options here?

1. kNN/SVM/MLP on the action units directly?
2. PCA, then kNN/SVM/MLP on the action units?
3. LDA, then kNN/SVM/MLP on the action units?

CLASSIFICATION?

We want to train a classifier with our training dataset, and then check whether the emotions in the test faces can be properly recognised.

What are our options here?

1. kNN/SVM/MLP on the action units directly?
2. PCA, then kNN/SVM/MLP on the action units?
3. LDA, then kNN/SVM/MLP on the action units?
4. Working with the (Eigen-)faces directly (eg, the pixels)?

Let’s open a notebook!

(link to the complete notebook)

MACHINE LEARNING METRICS

		Predicted class	
		Class=yes	*Class=no*
Actual class	*Class=yes*	True positive	False negative
	Class=no	False positive	True negative

Accuracy: $\frac{\text{correct predictions}}{\text{total observations}} = \frac{TP+TN}{TP+FP+FN+TN}$

Precision: $\frac{TP}{TP+FP}$

$\rightarrow$ Did we assign too many emotions to a face?

Recall (eg sensitivity): $\frac{TP}{TP+FN}$

$\rightarrow$ Did we 'miss' many emotions that were present?

F1 score: $2 \times \frac{\text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$

HOW TO IMPROVE IT FURTHER?

Ideas?

HOW TO IMPROVE IT FURTHER?

Ideas?

- better dataset (wider range of facial expressions)
- larger dataset
- use the raw images directly? use the facial landmark?
- different classifier $\rightarrow$ neural network (deep or not)?

Natural emotions are much harder

ML FOR SOCIAL BEHAVIOURS GENERATION

LEARNING FROM DEMONSTRATION

- acquire new skills by learning to imitate an expert
- most often used for manipulation
- 3 main approaches: *kinesthetic teaching*, *teleoperation*, *visual observation*
- Common algorithm: **reinforcement learning**

Video source: *Kormushev et al. 2010*

[Ravichandar et al. **Recent Advances in Robot Learning from Demonstration** Annual Reviews 2019]

LEARNING SEQUENCES

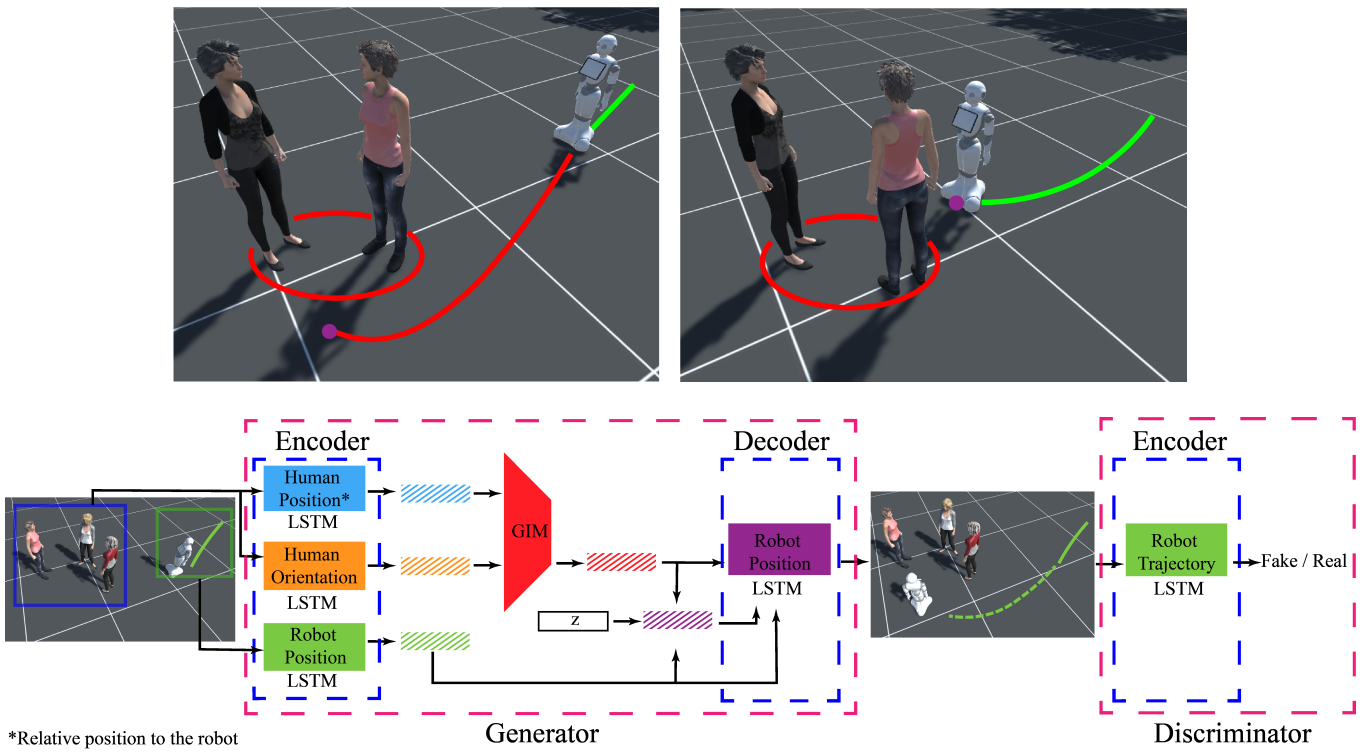
- 2 arms, 12 DoF
- Inputs: on-board 112×112px camera, joint state
- Training from 40 teleoperated demonstrations of ≈70s

Not only learning poses, but **sequences as well**; *Time-Delay Neural Network* (TDNN) to learn to predict the next step.

[Yang et al. **Repeatable Folding Task by Humanoid Robot Worker using Deep Learning** – Robotics and Automation Letters 2017]

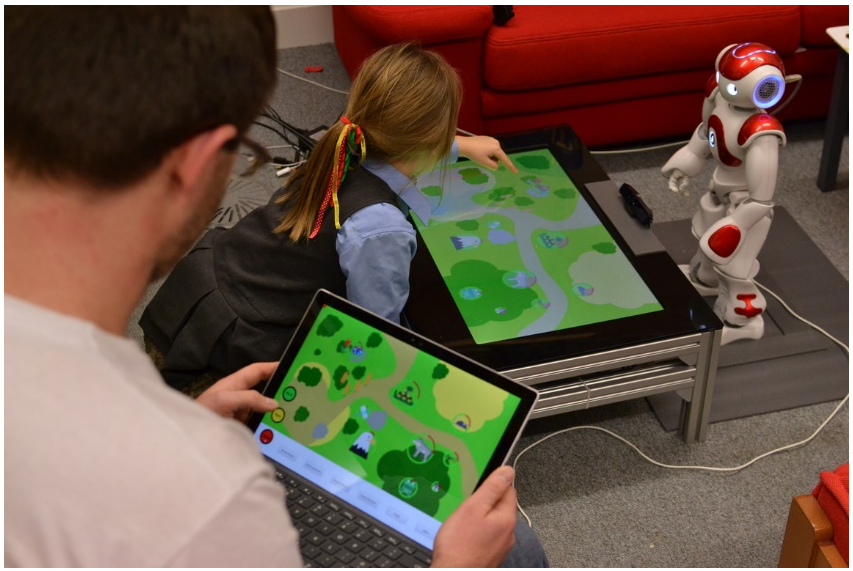
GENERATIVE ADVERSARIAL NETS FOR BEHAVIOUR GENERATION

Some of the most exciting recent work in using ML for robot behaviour generation involve **Generative Adversarial Networks** (GANs):



[Yang et al. **AppGAN: Generative Adversarial Networks for Generating Robot Approach Behaviours [...]** RoMan 2019]

REINFORCEMENT LEARNING APPLIED TO SOCIAL ROBOTICS

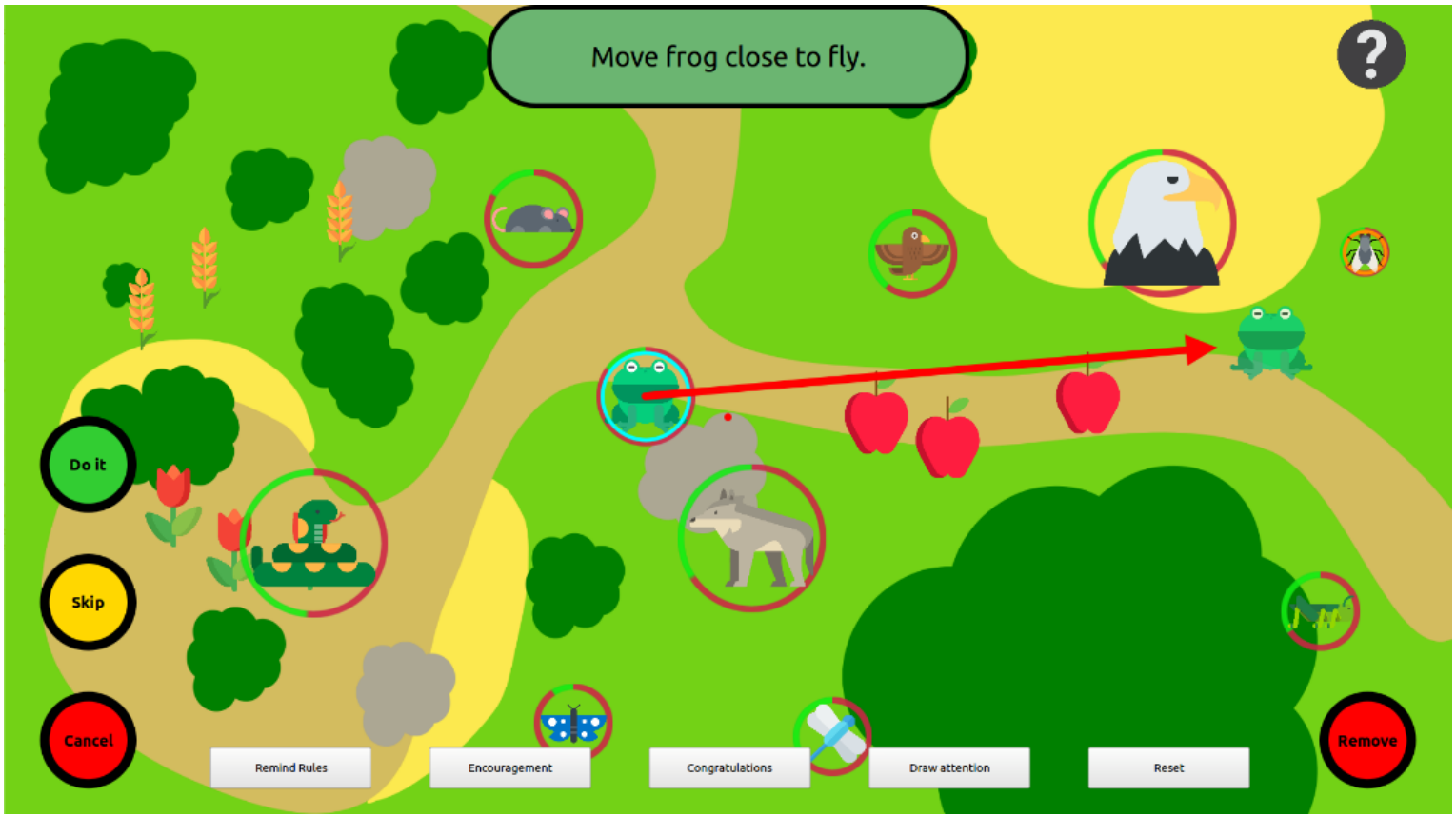


The children plays a game about food chains; the robot learns to guide them (*task-specific action policy*) and encourage them (*social action policy*)

Complex problem: $|state| = 210$ $|action\_space| = 655$

[Senft et al. **Teaching robots social autonomy from in situ human guidance** Science Robotics 2019]

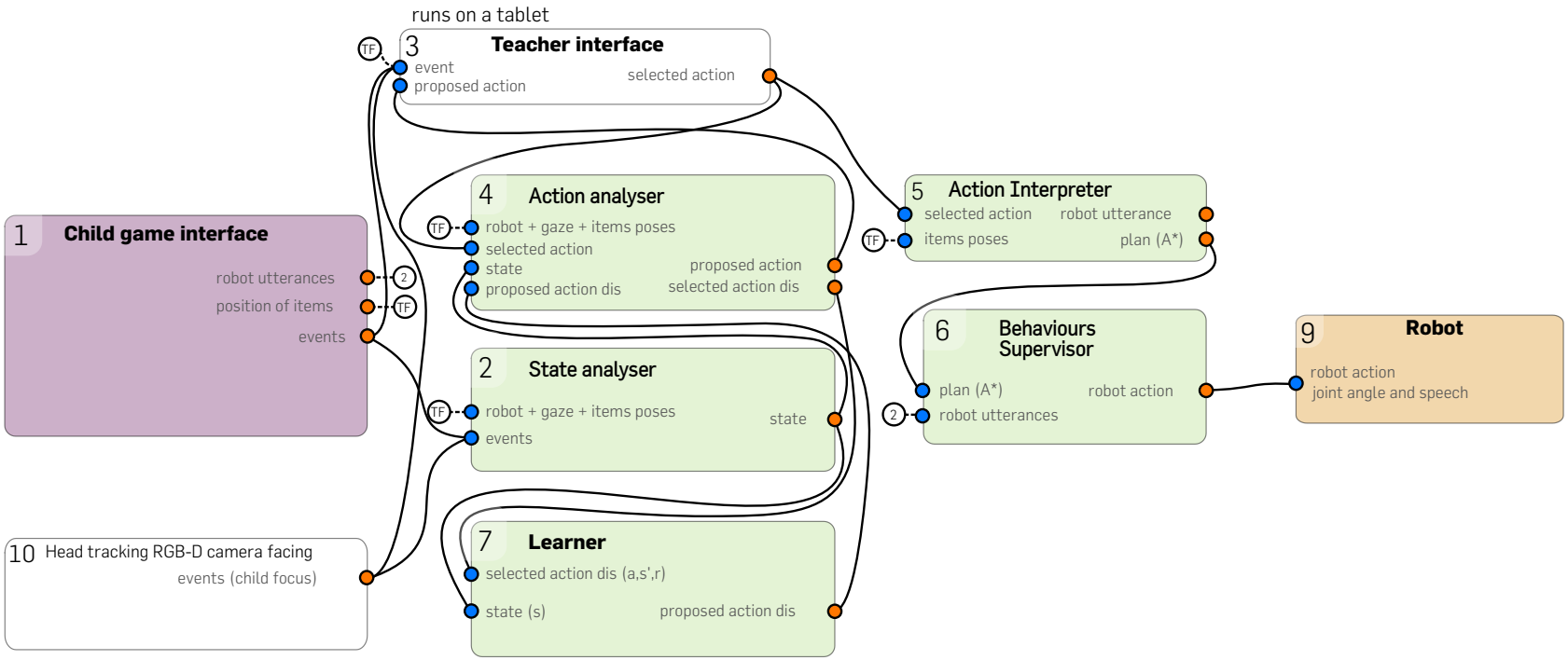
TEACHER'S INTERFACE



The robot's teacher (an end-user: might be the actual child's teacher) has a tablet interface that mirror's the child one, and add robot's teleoperation and rewards.

[Senft et al. **Teaching robots social autonomy from in situ human guidance** Science Robotics 2019]

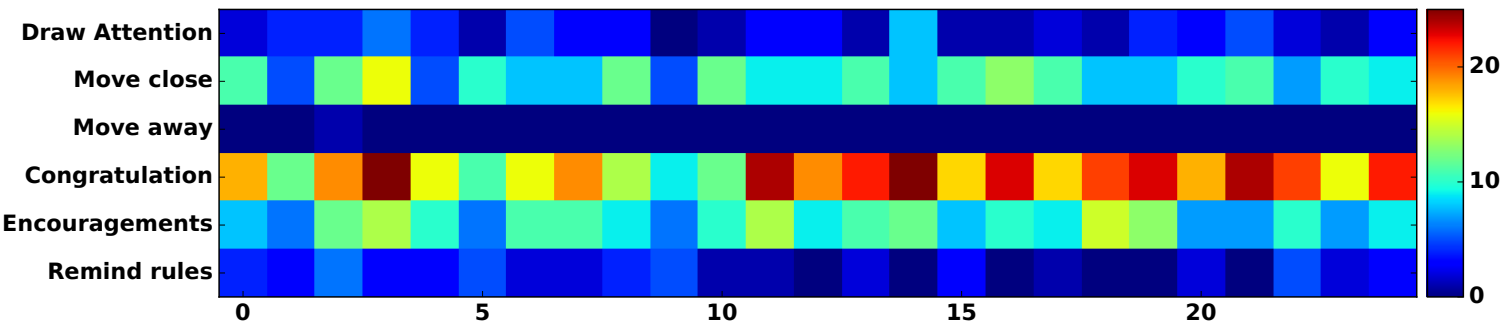
SOFTWARE ARCHITECTURE



[Senft et al. **Teaching robots social autonomy from in situ human guidance** Science Robotics 2019]

LEARNT ROBOT'S BEHAVIOUR

Distribution of actions for the 25 children participants:
Supervised



→

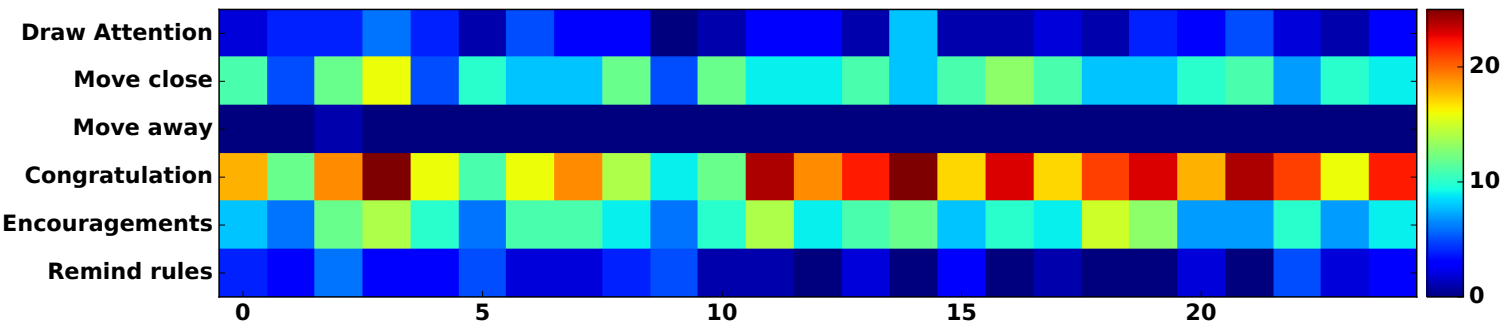
the robot personalises its action policies to the child's behaviour.

[Senft et al. **Teaching robots social autonomy from in situ human guidance** Science Robotics 2019]

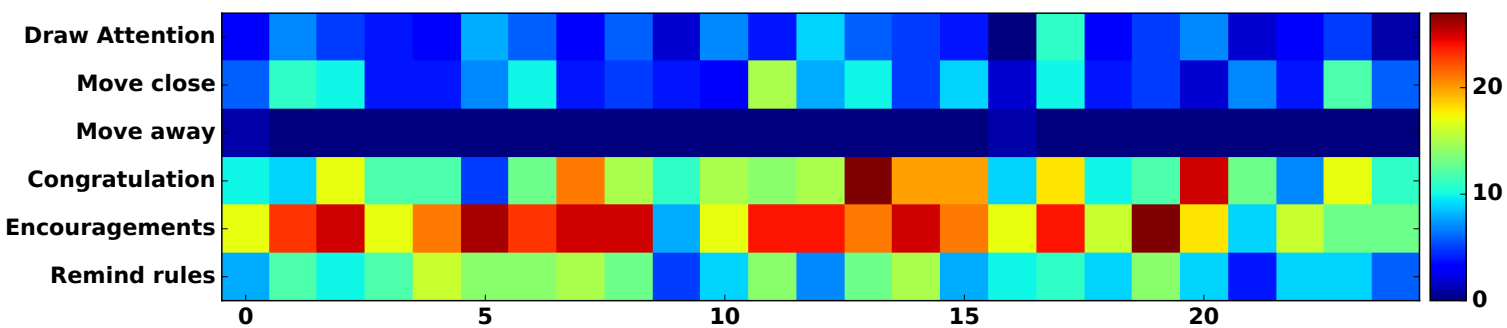
LEARNT ROBOT'S BEHAVIOUR

Distribution of actions for the 25 children participants:

Supervised



Autonomous



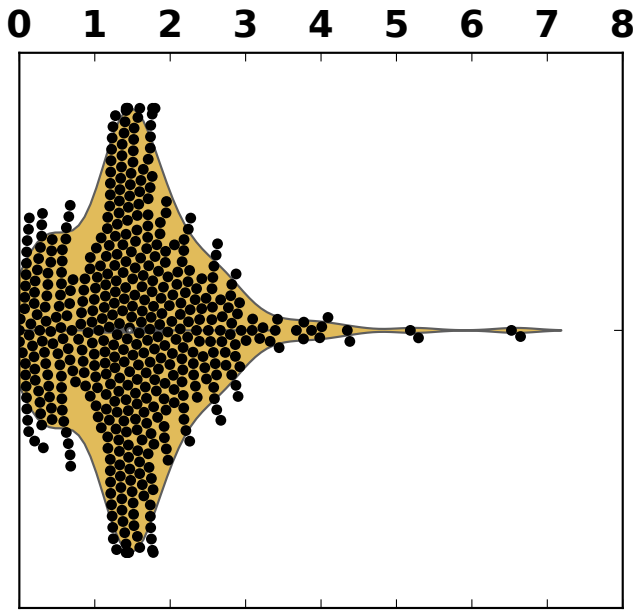
→

the robot personalises its action policies to the child's behaviour.

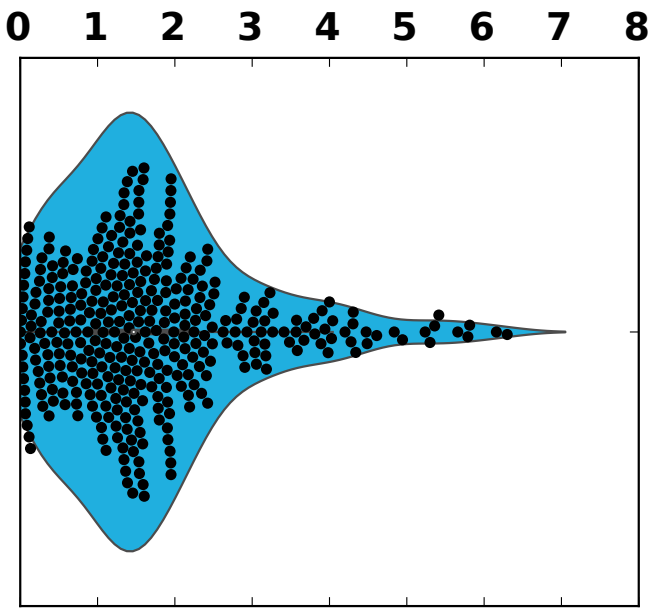
[Senft et al. **Teaching robots social autonomy from in situ human guidance** Science Robotics 2019]

LEARNT ROBOT'S BEHAVIOUR

Time between a child's successful action and a praise:



supervised



autonomous

→ the robot has also learnt an appropriate 'social timing'.

[Senft et al. **Teaching robots social autonomy from in situ human guidance** Science Robotics 2019]

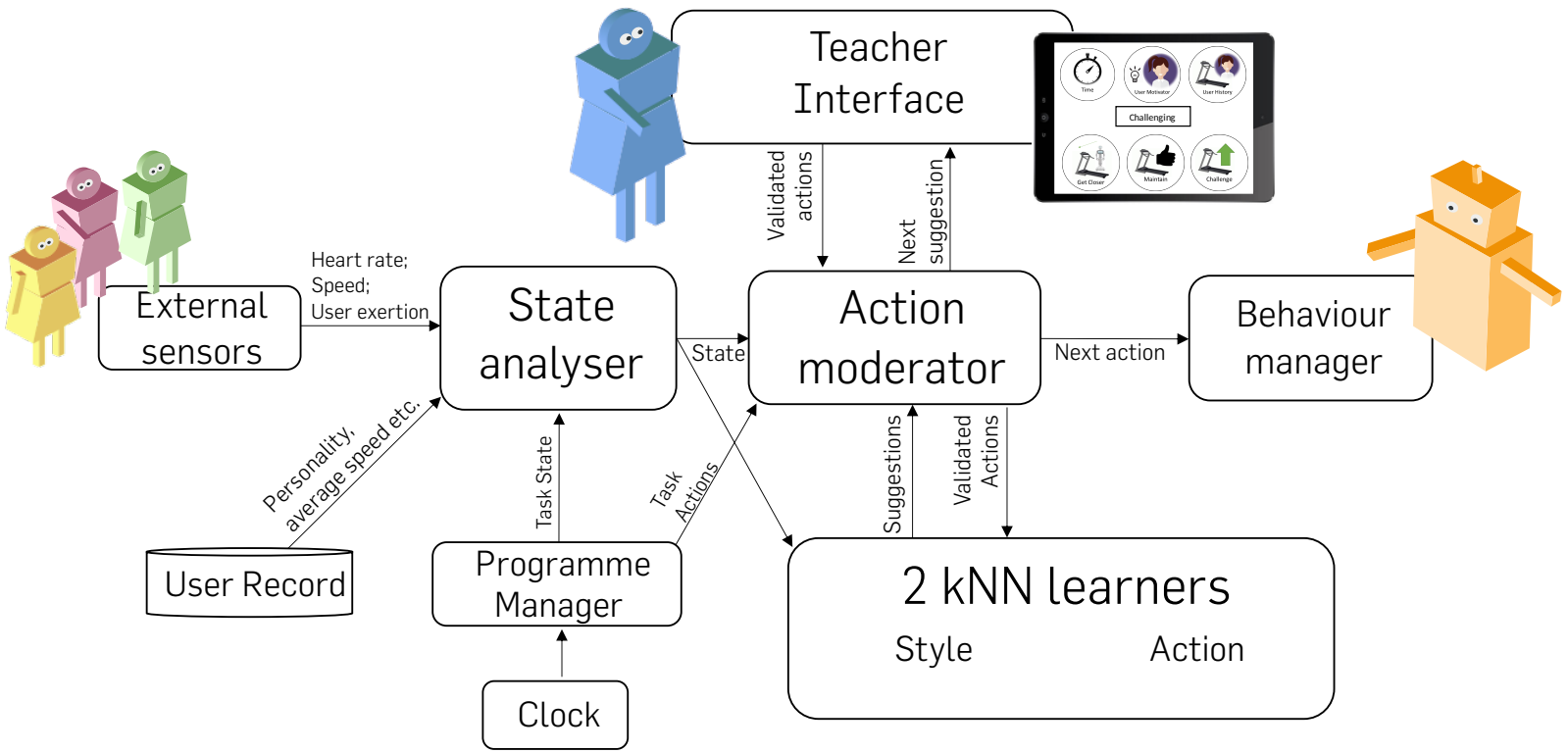
COUCH-TO-5KM: EXPERT-IN-THE-LOOP MACHINE LEARNING

[Winkle et al. **In-Situ Learning from a Domain Expert for Real World Socially Assistive Robot Deployment** RSS 2020]

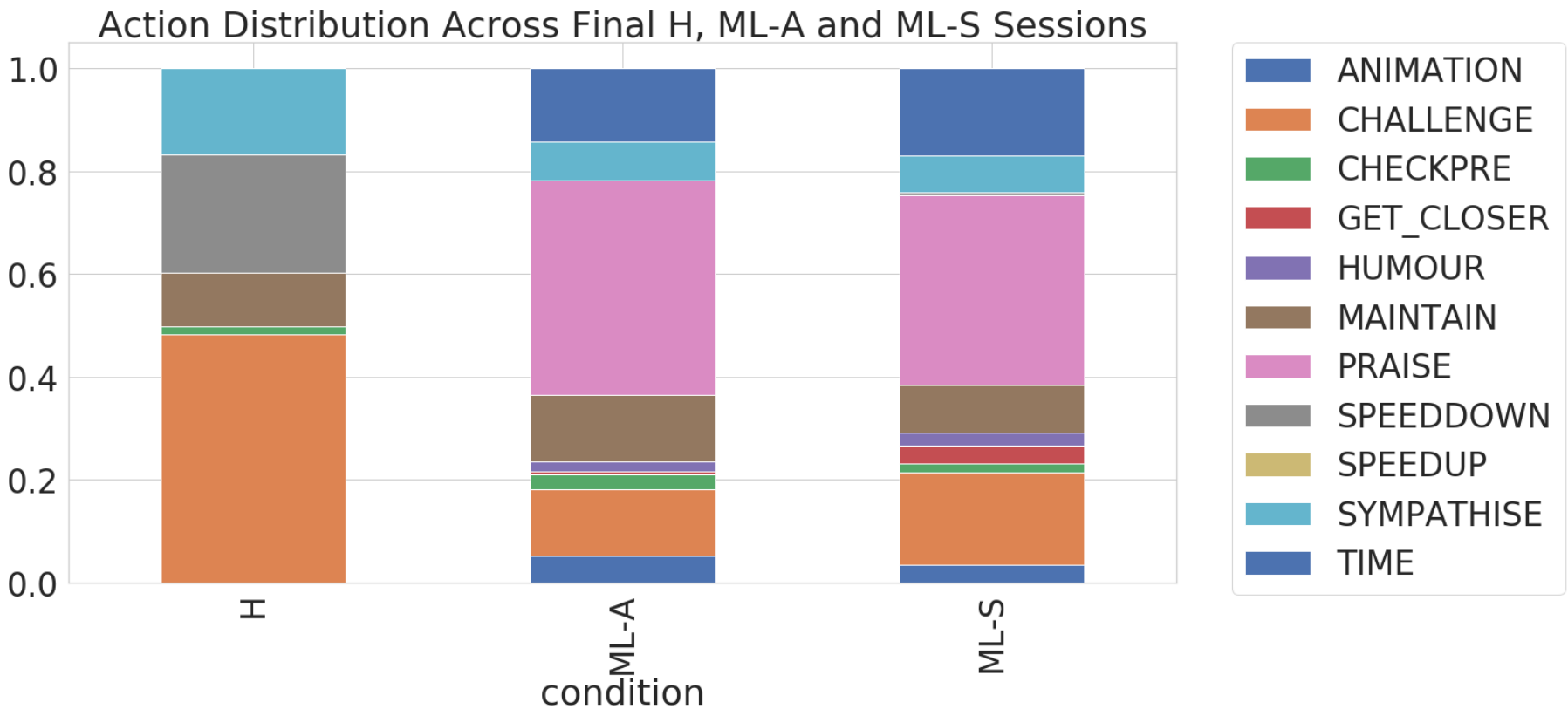
COUCH TO 5KM STUDY

- 9 participants
- 3 months; 27 one-hour sessions per participants
- 20 input features; 11 actions (task-specific or social)
- human-in-loop design and machine learning
- robot evolving from full teleoperation to full task and social autonomy

SOFTWARE ARCHITECTURE

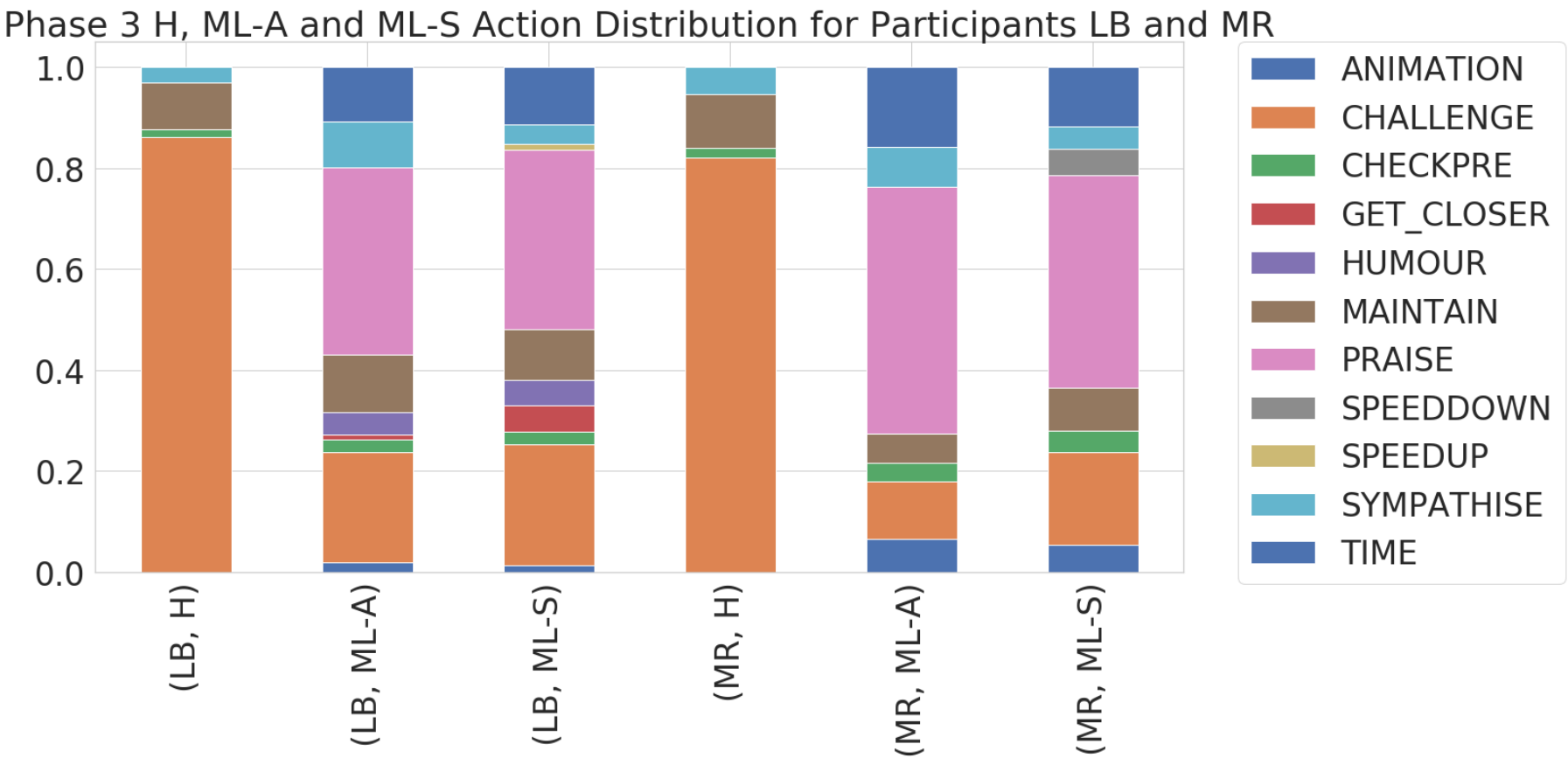


LEARNING TO IMITATE THE HUMAN EXPERT



[Winkle et al. **In-Situ Learning from a Domain Expert for Real World Socially Assistive Robot Deployment** RSS 2020]

LEARNING TO PERSONALISE



[Winkle et al. **In-Situ Learning from a Domain Expert for Real World Socially Assistive Robot Deployment** RSS 2020]

IN THIS LECTURE

- Basic classifiers: kNNs, SVM, MLP
- Build your own emotion classifier from scratch!
- ML to teach a robot social behaviours

That's all for today, folks!

Questions:
severin.lemaignan@brl.ac.uk

Slides:
github.com/severin-lemaignan/lecture-hri-ml-for-hri