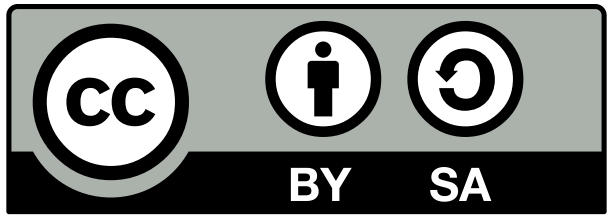
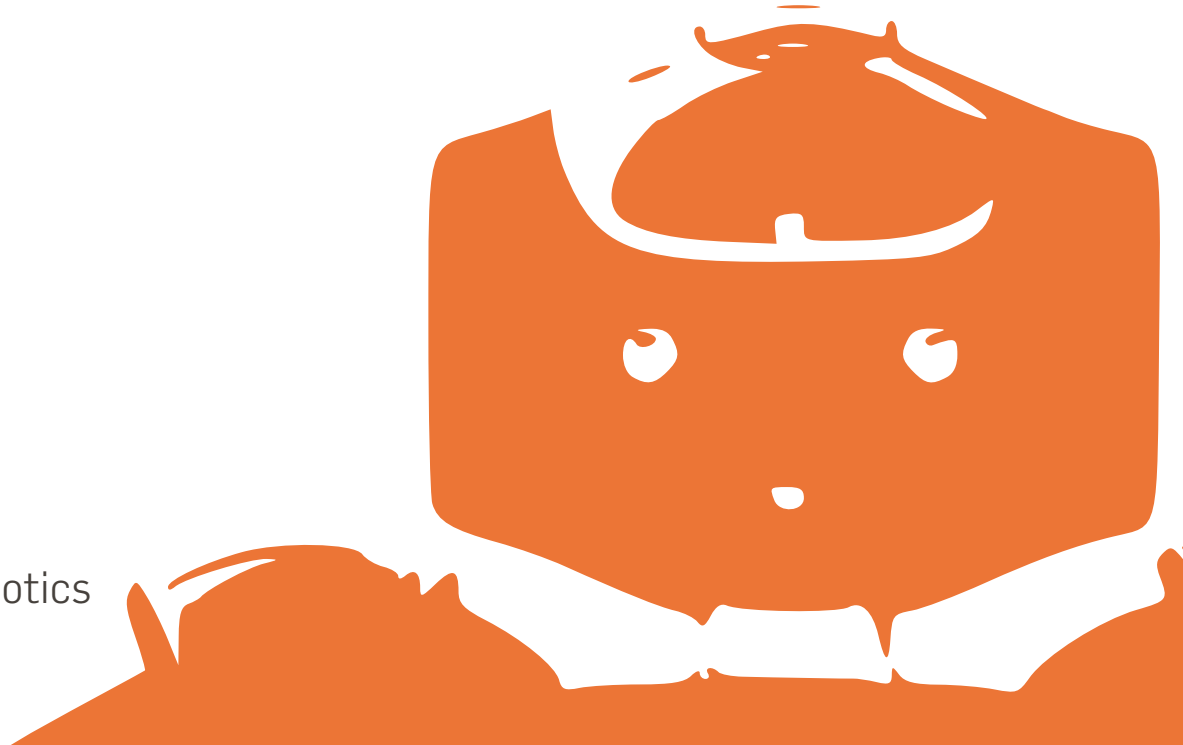


ROBOTICS WITH PLYMOUTH UNIVERSITY

# Social Signal Processing

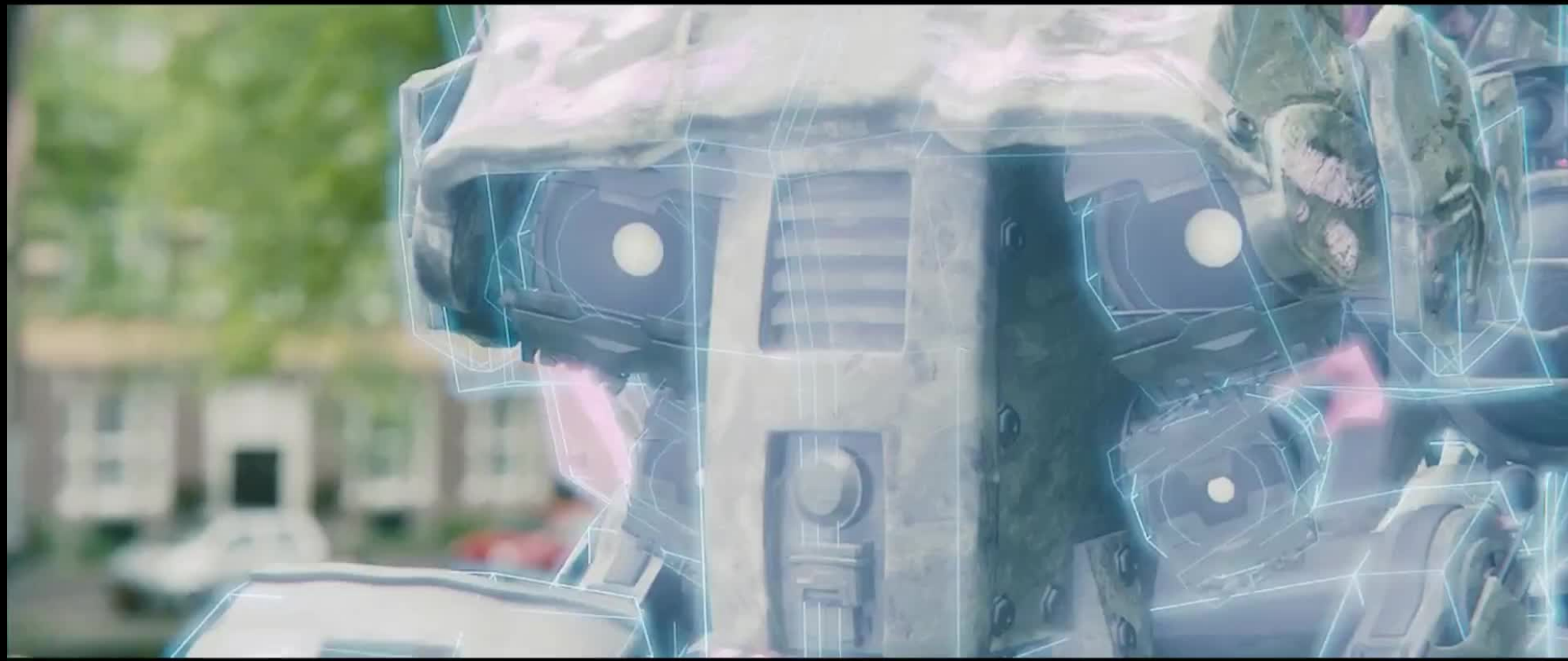
AINT512



Séverin Lemaignan

Centre for Neural Systems and Robotics
**Plymouth University**

# WHAT ARE SOCIAL SIGNALS?




Gestures
Emotions
Face identification
Gaze
Prosody
Verbal communication
Proxemics
Body posture

# WHAT ARE SOCIAL SIGNALS?

- Social signals are ***observable*** behaviours that people display during social interactions

[Burgoon, Magnenat-Thalmann, Pantic, Vinciarelli, **Social Signal Processing**, 2017]

# WHAT ARE SOCIAL SIGNALS?

- Social signals are ***observable*** behaviours that people display during social interactions
- Social signals from an individual ***produces changes*** in others (like creating a belief about the person, generating an appropriate social response, perform an actions)

[Burgoon, Magnenat-Thalmann, Pantic, Vinciarelli, **Social Signal Processing**, 2017]

# WHAT ARE SOCIAL SIGNALS?

- Social signals are ***observable*** behaviours that people display during social interactions
- Social signals from an individual ***produces changes*** in others (like creating a belief about the person, generating an appropriate social response, perform an actions)
- the changes are not random, they follow ***principles and laws*** (in particular, ***social norms***)

[Burgoon, Magnenat-Thalmann, Pantic, Vinciarelli, **Social Signal Processing**, 2017]

# WHY?

The ability to recognize human social signals and social behaviours like turn taking, politeness, and disagreement is essential when building social robots, human-robot interaction, or interactive systems

# WHY?

The ability to recognize human social signals and social behaviours like turn taking, politeness, and disagreement is essential when building social robots, human-robot interaction, or interactive systems

## 3 main problems

- *Modeling*: identification of the principles and laws
- *Analysis*: automatic detection and interpretation
- *Synthesis*: automatic generation of artificial social signals

# IS IT HARD?

On the following video, pay particular attention to turn taking

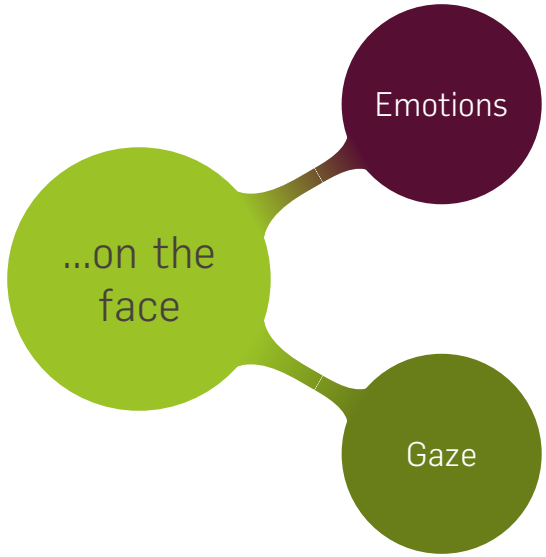
# IS IT HARD?

As with most human activities that seem easy to us, social signal processing is tremendously hard.

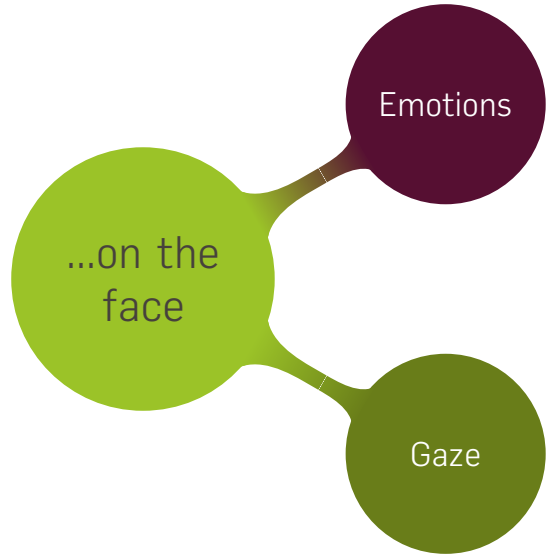
Often broken down in smaller, sometimes more manageable, tasks:

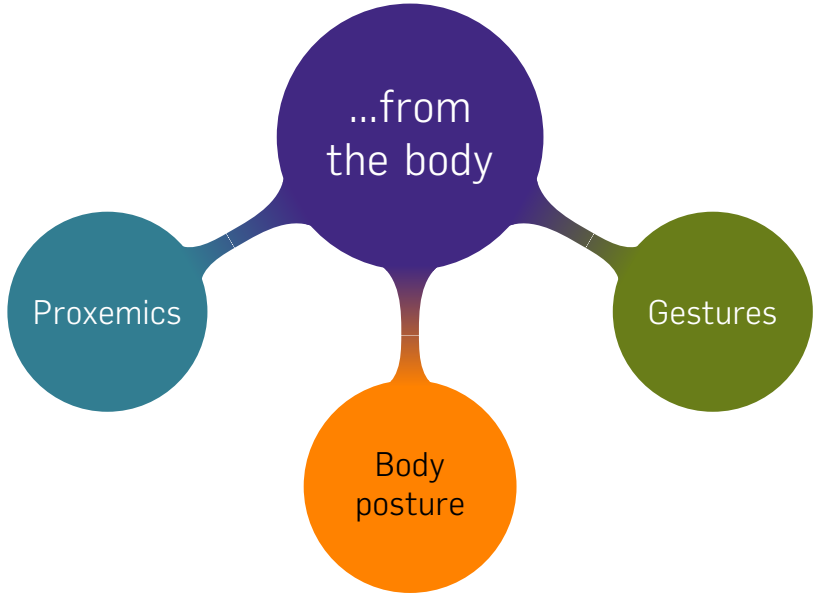
- People detection
- Face detection
- Face recognition
- Gesture recognition
- Gaze detection
- Facial expression reading (wink, blink, talking, …)
- Detection of social signals from verbal communication
- Emotion recognition (from faces, movement, speech, …)
- …

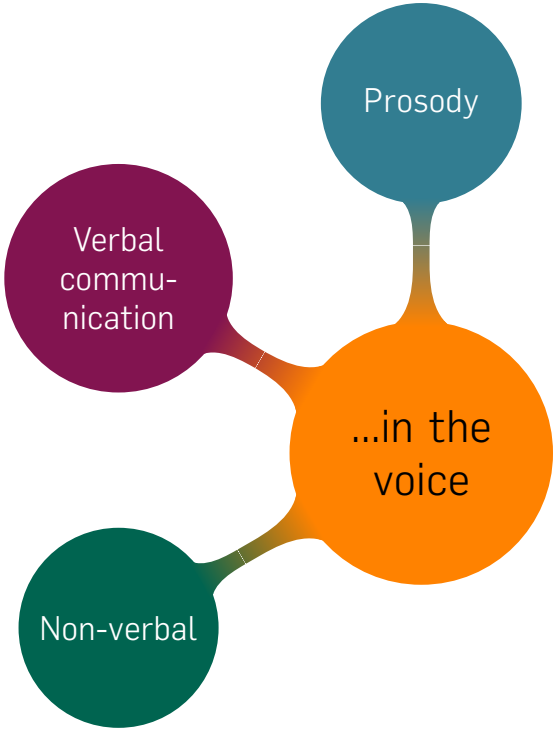
**Social Signals...**



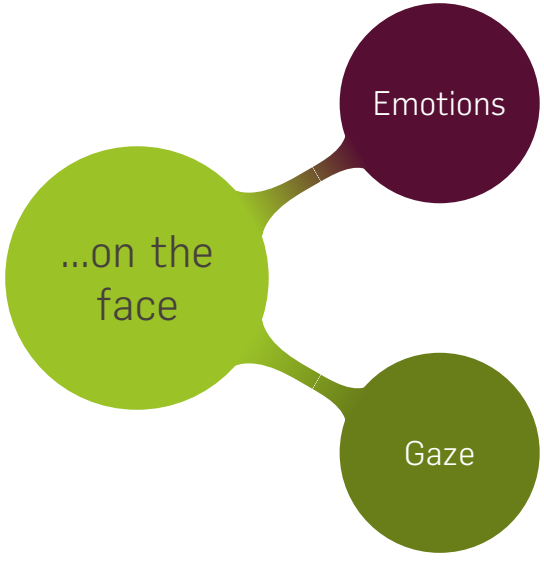
**Social Signals...**

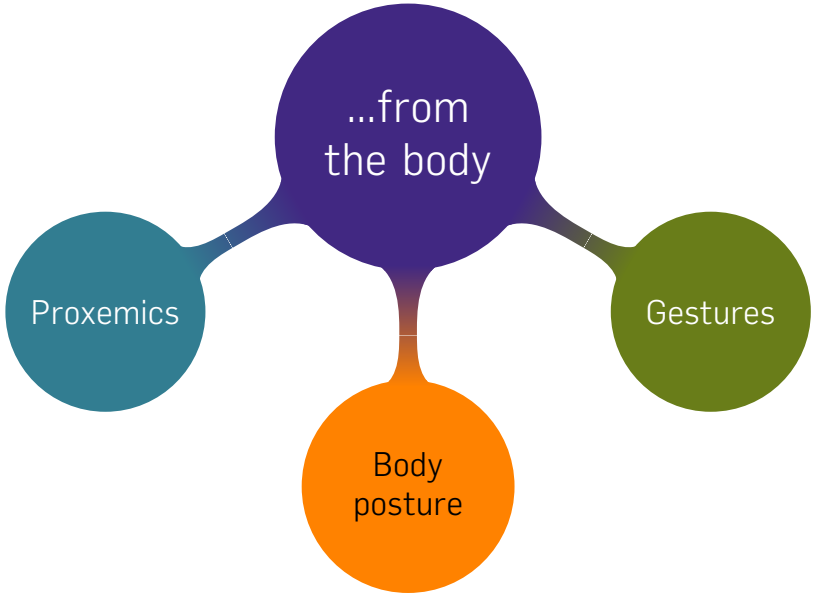





**Social Signals...**



# IN THIS LECTURE

- A historical case study: Kismet
- Recognising emotions from speech
- Speech recognition in the context of HRI
- Classification (k-nearest neighbours and Support Vector Machines)
- Classification for social signal processing

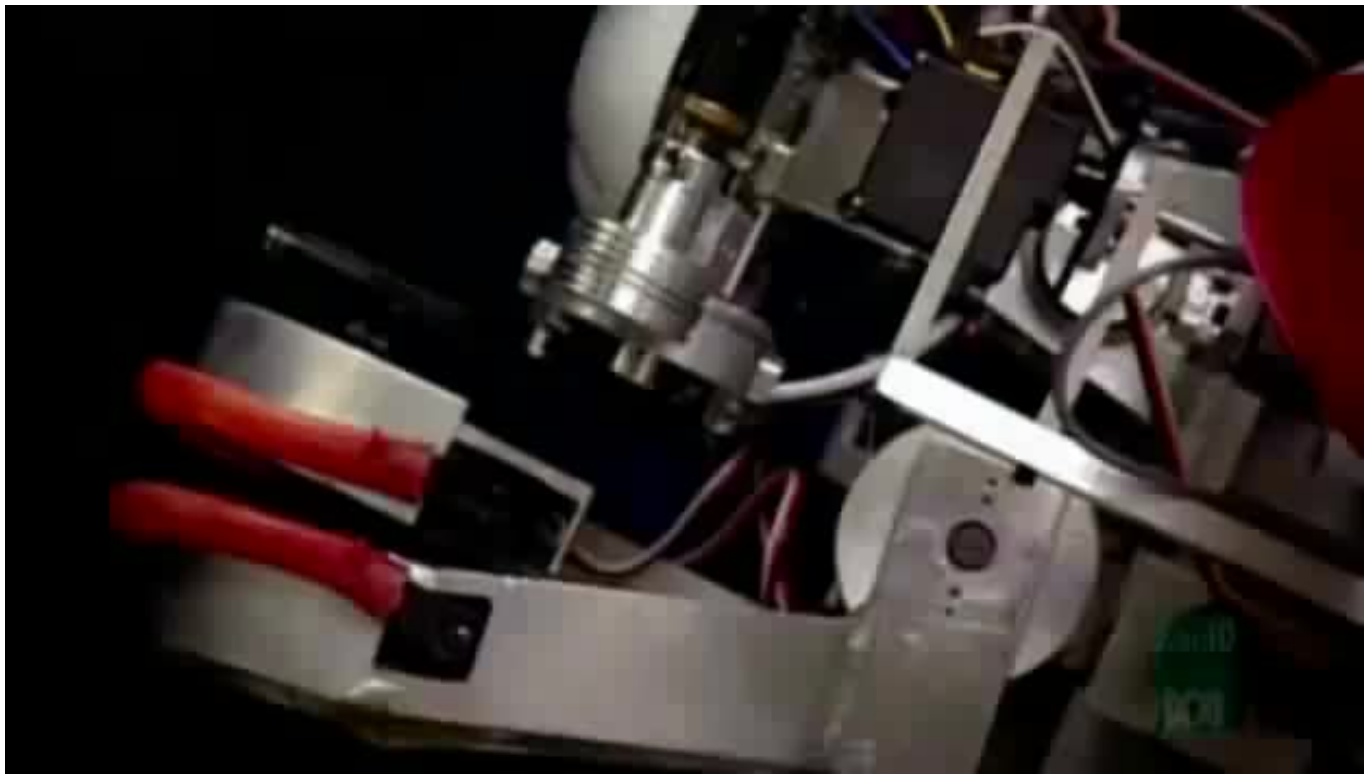
# NEXT 2 WEEKS

- How to extract facial features
- How to estimate gaze
- How to measure attention
- How to recognise a face
- How to model emotions

# CASE STUDY: KISMET'S VISION SYSTEM

# CASE STUDY: KISMET'S VISION SYSTEM

Remember Kismet?



Built by Cynthia Breazeal and a team of postgrad students at the Massachusetts Institute of Technology (MIT) in 1997.

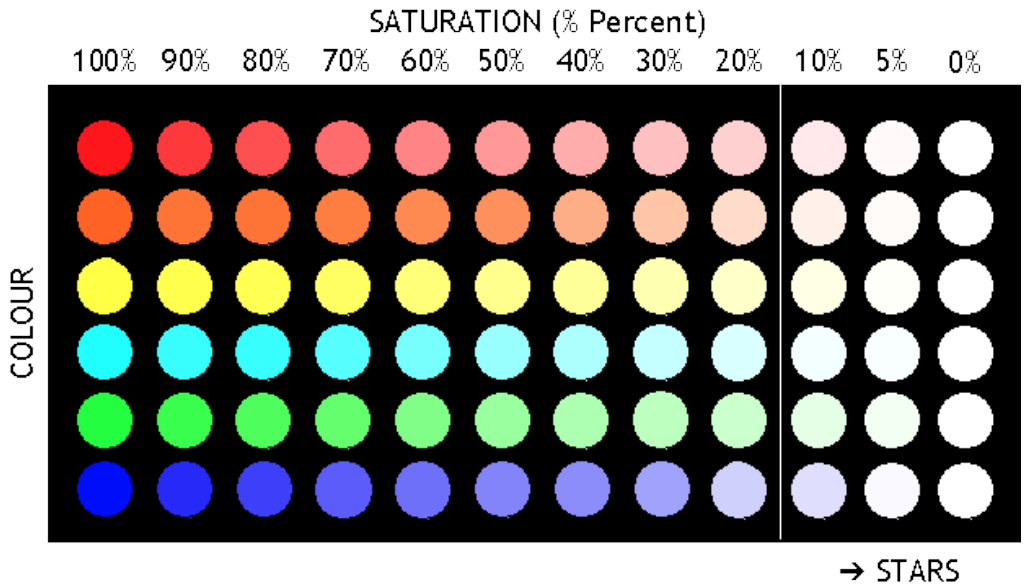
# CASE STUDY: KISMET'S VISION SYSTEM

Primary goal: implement an **attention system**: directing gaze towards salient features.

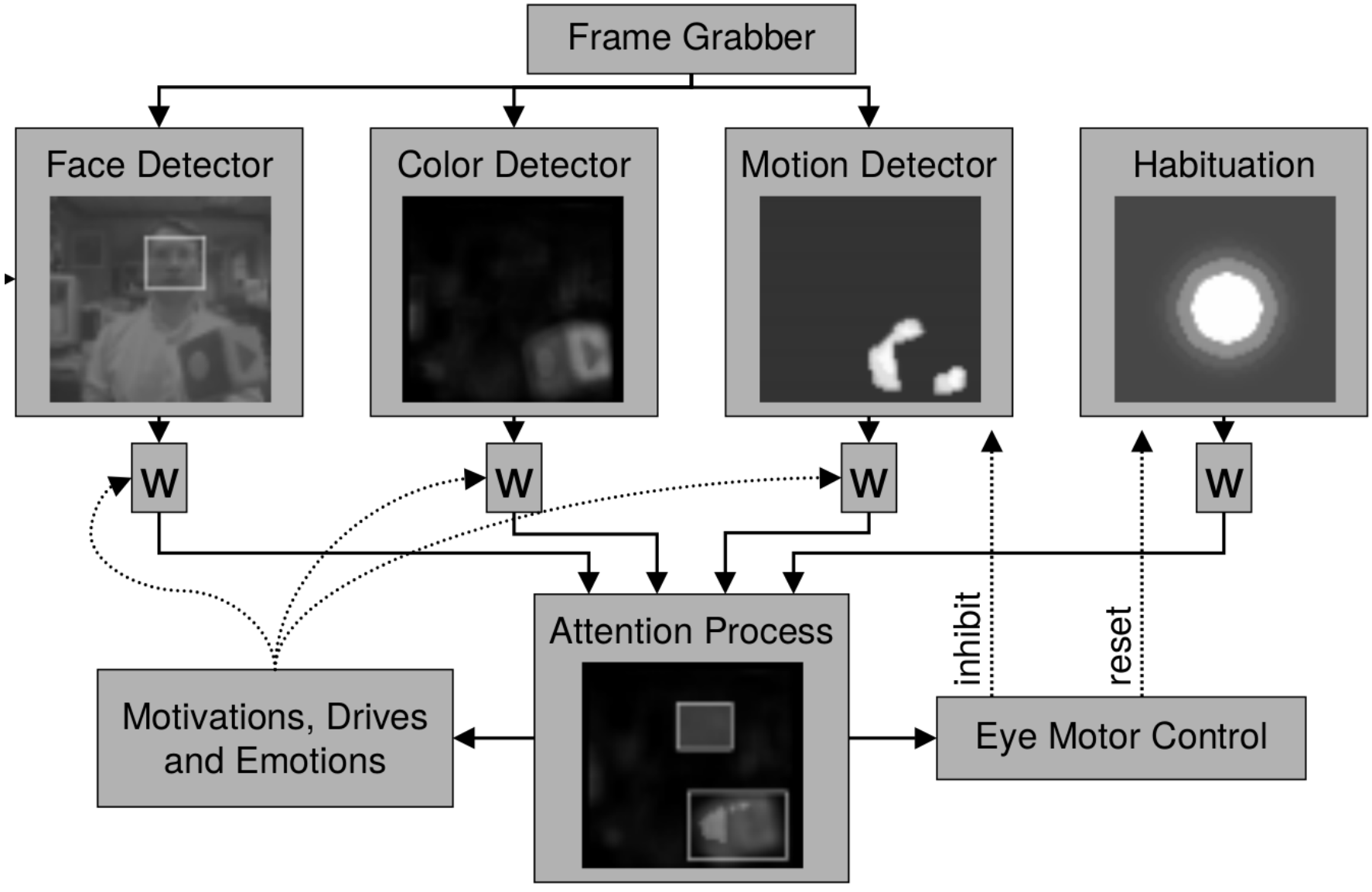
- This should be real-time.
- Should resemble the attention system of human infants.

Attention to several cues:

- Skin tone
- Saturated colours
- Motion




Note that detecting skin tone is easier than detecting faces, and often has the same result as face detection


Frame Grabber
Face Detector
Color Detector
Motion Detector
Habituation
W
W
W
W
Attention Process
inhibit
reset
Motivations, Drives and Emotions
Eye Motor Control

# SKIN TONE FEATURE DETECTOR

Incoming stream is 8-bit colour images, 128 by 128 pixels .

128 x 128 resolution limit caused by hardware through-put and computation available at the time, current hard-ware will be able to handle megapixel resolution in real-time.

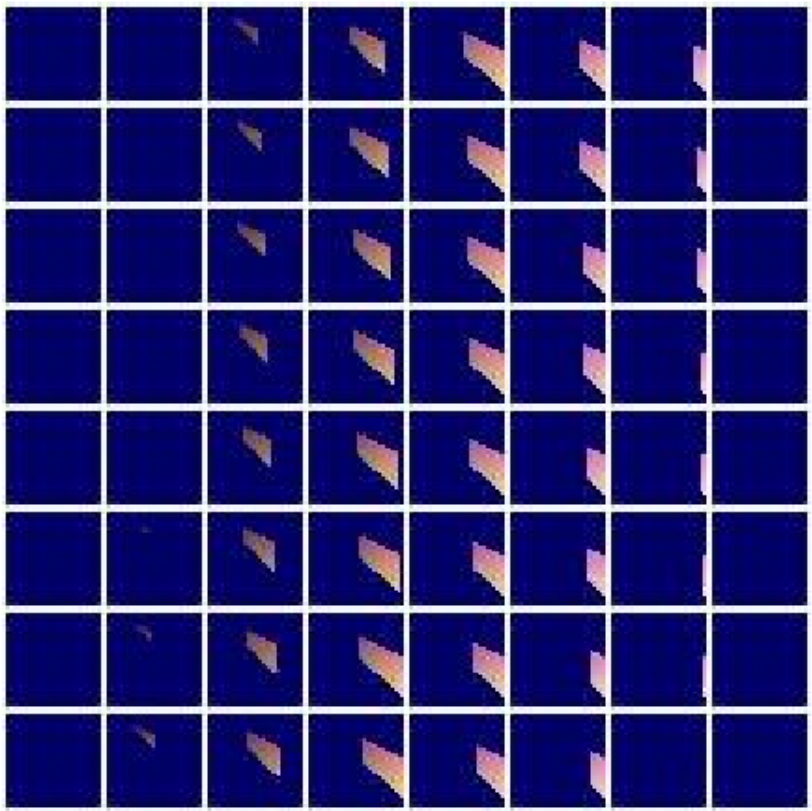
A pixel is ***not*** skin-toned if:

- $R < 1.1G$ and
- $R < 0.9B$ and
- $R > 2.0 \times max(G, B)$ and
- $R < 20$ and
- $R > 250$

This simple procedure works for most skin colours and under most lighting conditions.

It is claimed this even works with non-Caucasian skin, as it picks up a colour component in the blood.

# SKIN TONE FEATURE DETECTOR



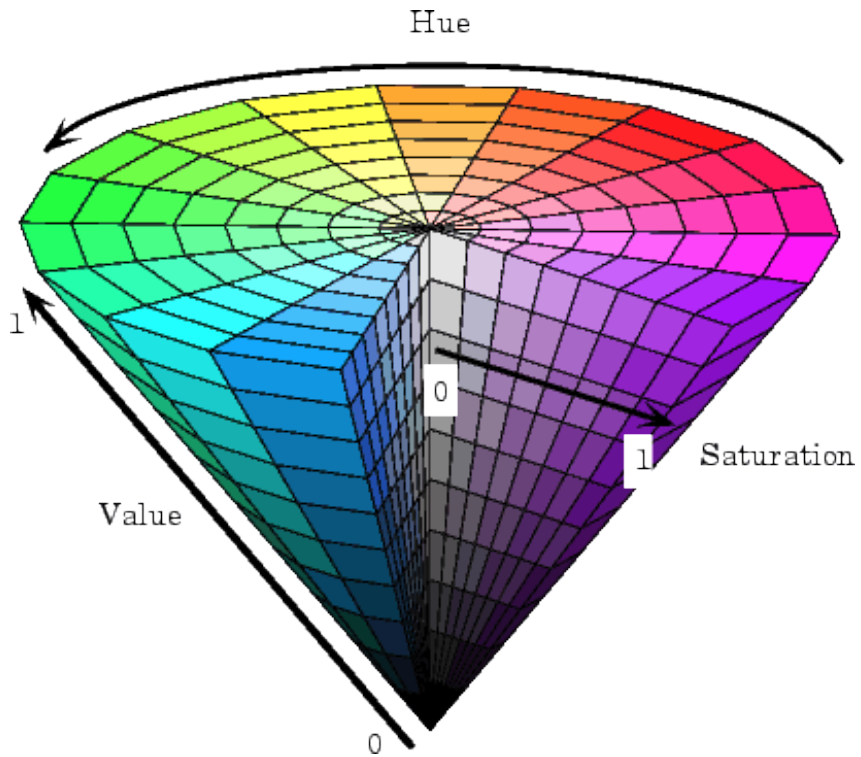
Cut through RGB 3-dimensional space, picking out only skin tone RGB values

Result of skin tone detector in action.

# SATURATION DETECTOR

Convert RGB to HSV values

- From `<red, green, blue>` to `<hue, saturation, value>` representation.
- **hue** is the "colour", **saturation** is how deep the colour is and **value** is how bright the pixel is.
- Usually an excellent first step in computer vision when processing colour images, as it removes the effect of uneven lighting.



# RGB TO HSV IN PYTHON

```
def rgb2hsv(r, g, b):
    r, g, b = r/255.0, g/255.0, b/255.0
    mx = max(r, g, b)
    mn = min(r, g, b)
    df = mx-mn
    if mx == mn:
        h = 0
    elif mx == r:
        h = (60 * ((g-b)/df) + 360) % 360
    elif mx == g:
        h = (60 * ((b-r)/df) + 120) % 360
    elif mx == b:
        h = (60 * ((r-g)/df) + 240) % 360
    if mx == 0:
        s = 0
    else:
        s = df/mx
    v = mx
    return h, s, v
```

or better:

```
import colorsys
colorsys.rgb_to_hsv(0.2, 0.4, 0.4)
--> (0.5, 0.5, 0.4)
colorsys.hsv_to_rgb(0.5, 0.5, 0.4)
--> (0.2, 0.4, 0.4)
```

*Source: Python recipes*

# SATURATION DETECTOR

- Convert image from RGB to HSV, and mark pixels (by setting them to WHITE) that have a saturation over a certain threshold.
- Focus on **Centre of Gravity** of all saturation pixels.

```
void centerOfGravity (int[][] image, int imageWidth, int imageHeight)
{
    int SumX = 0; int SumY = 0; int num = 0;

    for (int i=0; i<imageWidth; i++)
    {
        for (int j=0; j<imageHeight; j++)
        {
            if (image[i][j] == WHITE)
            {
                SumX = SumX + i; SumY = SumY + j;
                num++;
            }
        }
    }
    SumX = SumX / num; SumY = SumY / num;
    // The coordinate (SumX,SumY) is the center of gravity
```

# MOTION DETECTOR

- Motion is a strong social signal, essential for survival (both in detecting prey and predator). Robots are expected to be sensitive to motion as well.

```
int[][] motionDetection (int[][] image, int[][] imagePrevious,
                         int imageWidth, int imageHeight)
{
    int[][] imageMotion;
    int num = 0;
    imageMotion = (int**)malloc( imageWidth \ imageHeight)

    for (int i=0; i<imageWidth; i++)
    {
        for (int j=0; j<imageHeight; j++)
        {
            imageMotion[i][j] = abs(image[i][j] - imagePrevious[i][j]);
        }
    }
}
```

# OPTICAL FLOW

Today, motion detection is performed by computing the ***optical flow*** of a video stream.

The colours indicate the direction in which each pixel moves.

# POST-ATTENTIVE PROCESSING

After attention has been fixed, run some additional algorithms

- Eye detection: using template matching.
- Proximity estimation: using stereo vision.
- Loom detection: if the size of the motion blob increases rapidly.
- Threat detection: rapid motion or fast looming.

The results of these visual processes feed into the behaviour manager of the robot.

- Robot focuses on visual regions that draw its attention

# SELECTION OF ATTENTION

Kismet does not attend to all three features for computational reasons , but selects one according to its current behaviour.

For example, when the `seek_people` behaviour is active, it uses skin tone to direct its attention.

This was back in early 2000s. Current hardware can handle this computational load without problem. However, it is good practice to not spend processor cycles on signals you do not require.

# SELECTION OF ATTENTION

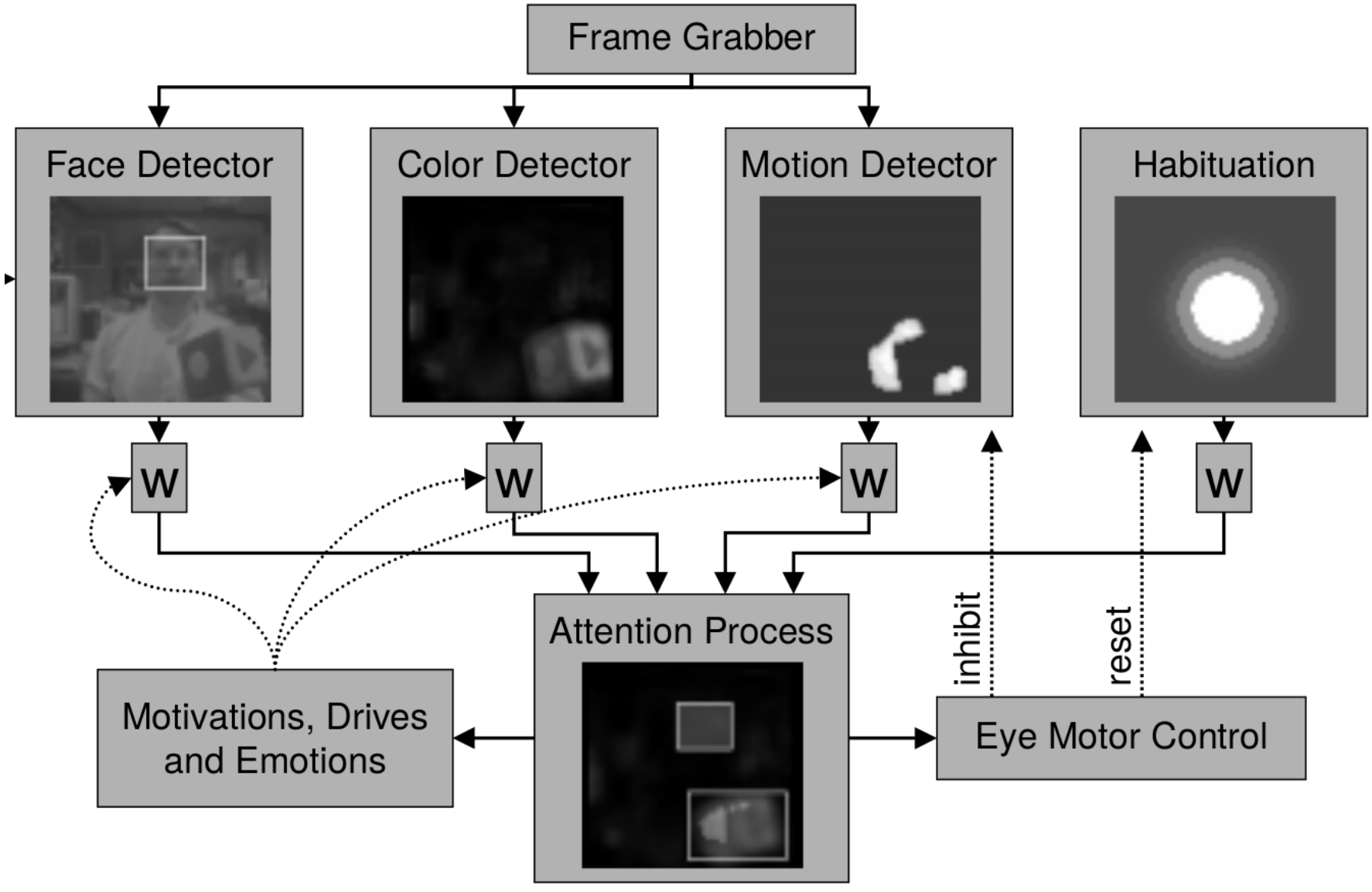
Kismet does not attend to all three features for computational reasons , but selects one according to its current behaviour.

For example, when the `seek_people` behaviour is active, it uses skin tone to direct its attention.

Implementation: an **attention activation map**

- A 128 by 128 map of activation for the feature detectors is constructed.
- A threshold is applied to get rid of noise.
- Connected regions are found using the 4-connect algorithm.
- Regions of less than 30 pixels are thrown away.
- Centroid is computed.
- Attention is drawn to largest region.

The attention map is independent of the feature (color, motion, ...)


Frame Grabber
Face Detector
Color Detector
Motion Detector
Habituation
W
W
W
W
Attention Process
inhibit
reset
Motivations, Drives and Emotions
Eye Motor Control

# THE AUDITORY SYSTEM

Kismet does not understand human language!

It only pays attention to emotional content of speech. This is done though monitoring **prosody**.

Advantages:

- rather easy and computationally inexpensive
- the robot is not committed to one single language
- the robot is speaker independent

# EMOTION FROM PROSODY

„Sie haben es gerade hochgetragen und jetzt gehen sie wieder runter“ (They just carried it upstairs and now they are going down again).

*Source: Berlin Database of Emotional Speech*

Which emotion do you recognise?

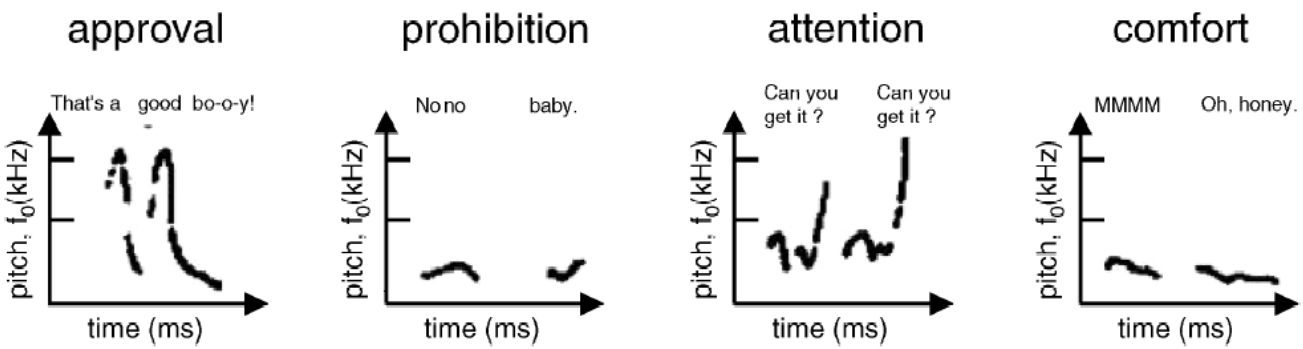
**Anger – Boredom – Disgust – Anxiety/Fear – Happiness – Sadness – Neutral**

Humans recognise emotions from prosody with approx. 80% accuracy (*caveat: this is for speech spoken by actors in a recording studio*)

# SPEECH PROCESSING SYSTEM

Compute features from the samples

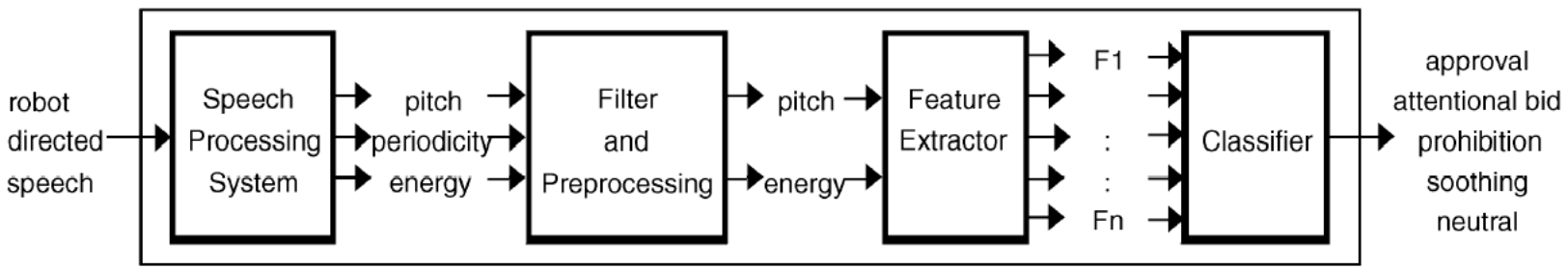
- Pitch mean
- Pitch variance
- Maximum pitch.
- Minimum pitch.
- Pitch range.
- ...



# AUDIO PROCESSING STREAM IN KISMET

Many different samples were recorded and classified by a human subject. Then an algorithm was trained (***Expectation Maximisation***) to classify the samples into five distinct classes.

- Approvals
- Attention drawing
- Prohibition
- Comforting
- Neutral



# SPEECH RECOGNITION

# SPEECH RECOGNITION

See Guido's lectures, but here's a small heads up.

Automated Speech Recognition has improved tremendously over the past few years.

- Improved recognition performance
- Speaker independence
- Can deal with a larger amount of background noise

# SPEECH RECOGNITION

Plenty of reports on increased performance

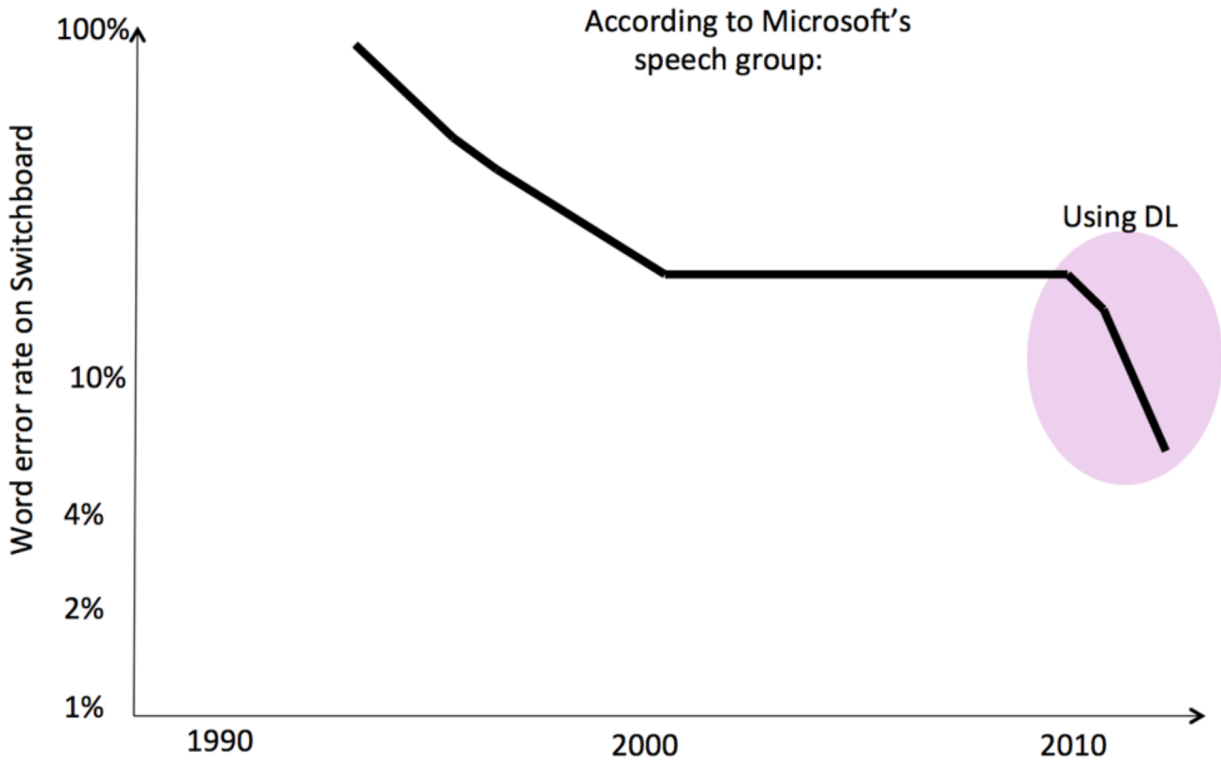
- "Historic Achievement: Microsoft researchers reach human parity in conversational speech recognition" (Oct 2016)
- Error rate (% of misunderstood words) on Switchboard (SWB) and CallHome (CH) corpus:

| Model | N-gram LM | | Neural net LM | |
|---|---|---|---|---|
| | CH | SWB | CH | SWB |
| Povey et al. [54] LSTM | 15.3 | 8.5 | - | - |
| Saon et al. [51] LSTM | 15.1 | 9.0 | - | - |
| Saon et al. [51] system | 13.7 | 7.6 | 12.2 | 6.6 |
| 2016 Microsoft system | 13.3 | 7.4 | 11.0 | 5.8 |
| Human transcription | | | 11.3 | 5.9 |

# SPEECH RECOGNITION

So what has changed?

- Introduction of new models for speech recognition based on Deep Neural Networks (DNN) or Convolutional Neural Networks (CNN), replacing models based on n-grams or Hidden Markov Models (HMM).

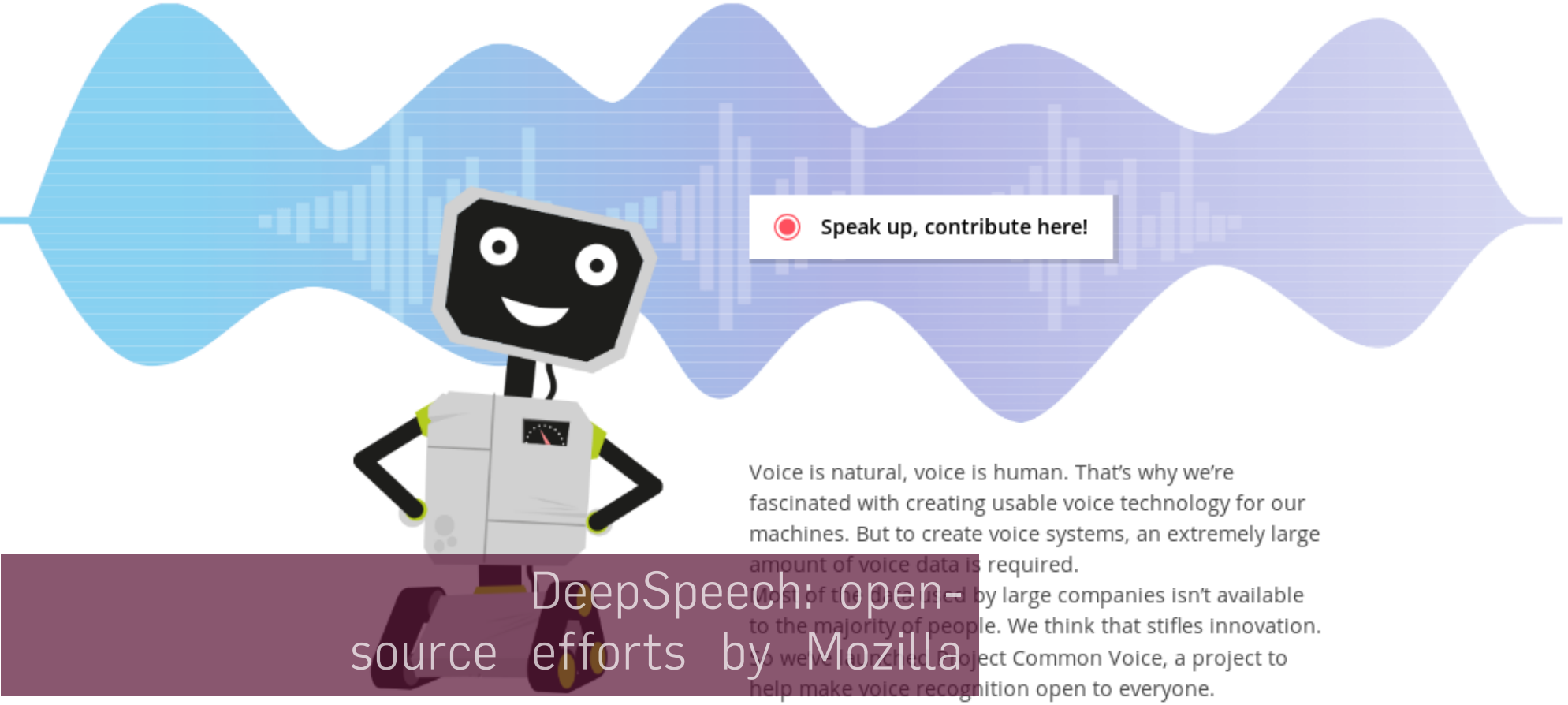


# SPEECH RECOGNITION

So what has changed?

- More availability of training data, often collected through speech command interfaces (such as Google Now, Microsoft's Xbox chat function).
- Computational power to train DNN has increased: High Performance Computing clusters and Graphical Processing Units (GPUs).
- Recognition happens in the cloud, rather than on-board a device. This offers more computational power and access to large networks.

The Common Voice project is Mozilla's initiative to help teach machines how real people speak.




Voice is natural, voice is human. That's why we're fascinated with creating usable voice technology for our machines. But to create voice systems, an extremely large amount of voice data is required.
Most of the data used by large companies isn't available to the majority of people. We think that stifles innovation.
So we've launched project Common Voice, a project to help make voice recognition open to everyone.

READ MORE

DeepSpeech: open-source efforts by Mozilla

# SPEECH RECOGNITION

Question

- So, given that ASR performance seems high, how good is it really for Human-Robot Interaction?

Human-Robot Interaction is unique

- There is often a noise environment.
- Users are at a distance from the robot (and any speakers carried on the robot). Almost all ASR engines have been trained on speech recorded close to the microphone (< 10cm).
- There might be a multi-party conversation (i.e. multiple speakers).

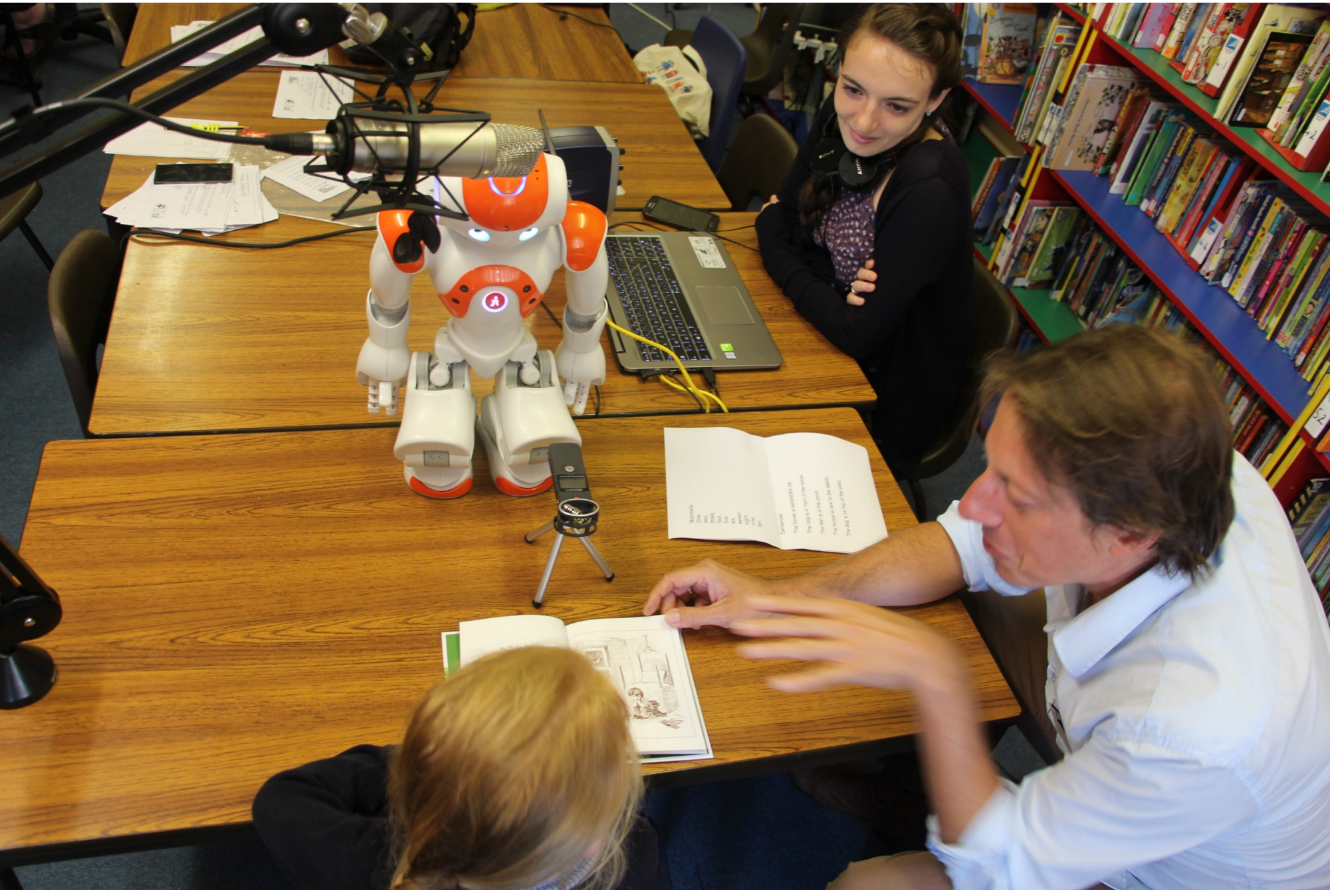
# SPEECH RECOGNITION FOR CHILDREN

How well does speech recognition perform for child speech?

- For counting (1,2,3, …) and short sentences **spoken by adults**, recognition rate is **90% using Nao microphone and on-board ASR** (Nuance VoCon 4.7). And up to 99% recognition with a high-quality microphone and Google ASR.
- Can we assume that ASR on children's speech will have the same performance?

Child speech is very different from adult speech.

- Higher pitch (due to small vocal tracts).
- higher number of disfluencies and, especially in younger children, language utterances are often ungrammatical (e.g. "The boy ***putted*** the frog in the box").


Numbers
One
two
three
four
five
six
seven
eight
nine
ten
Sentences
The horse is behind the car
The dog is in front of the horse
The fish is in the pond
The horse is next to the table
The dog is on top of the shed

# EXAMPLES OF RECORDINGS (CLEANED-UP, NORMALISED)

- Numbers (0 to 10)


- Sentences

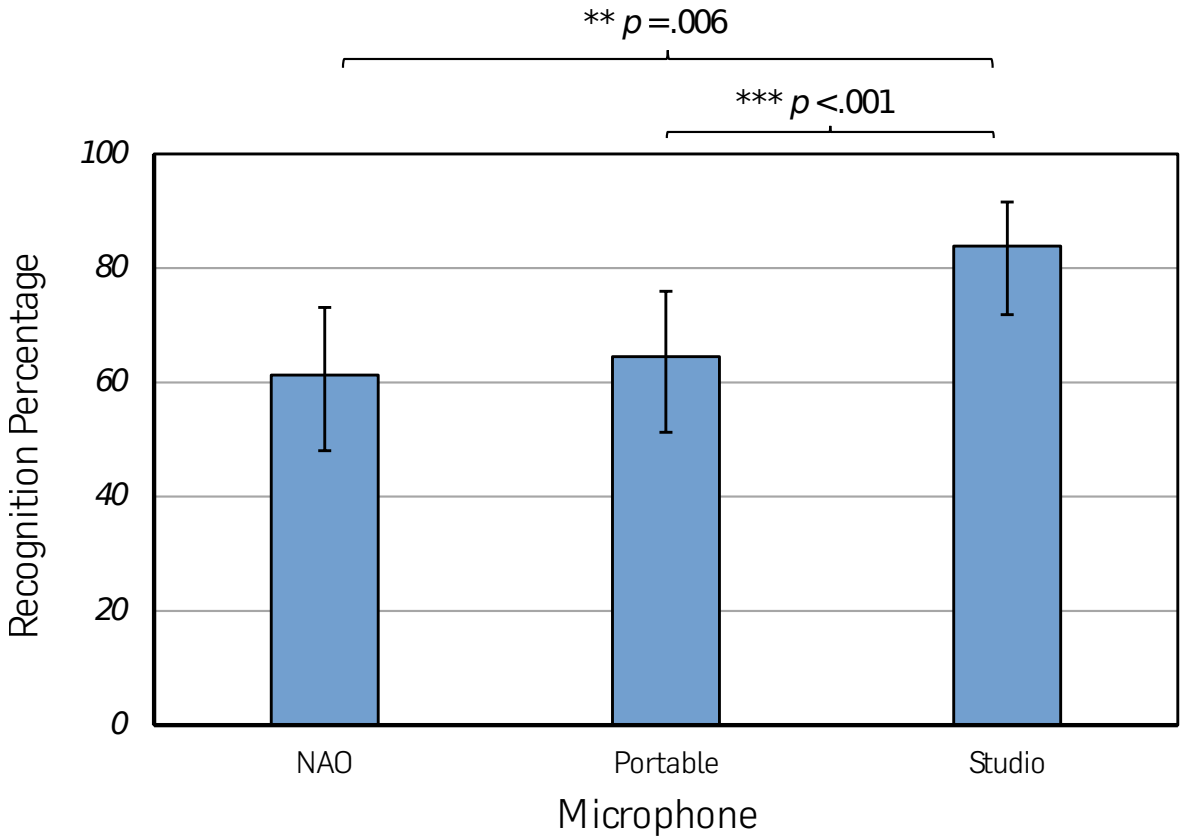

Clean Noisy

- Spontaneous speech



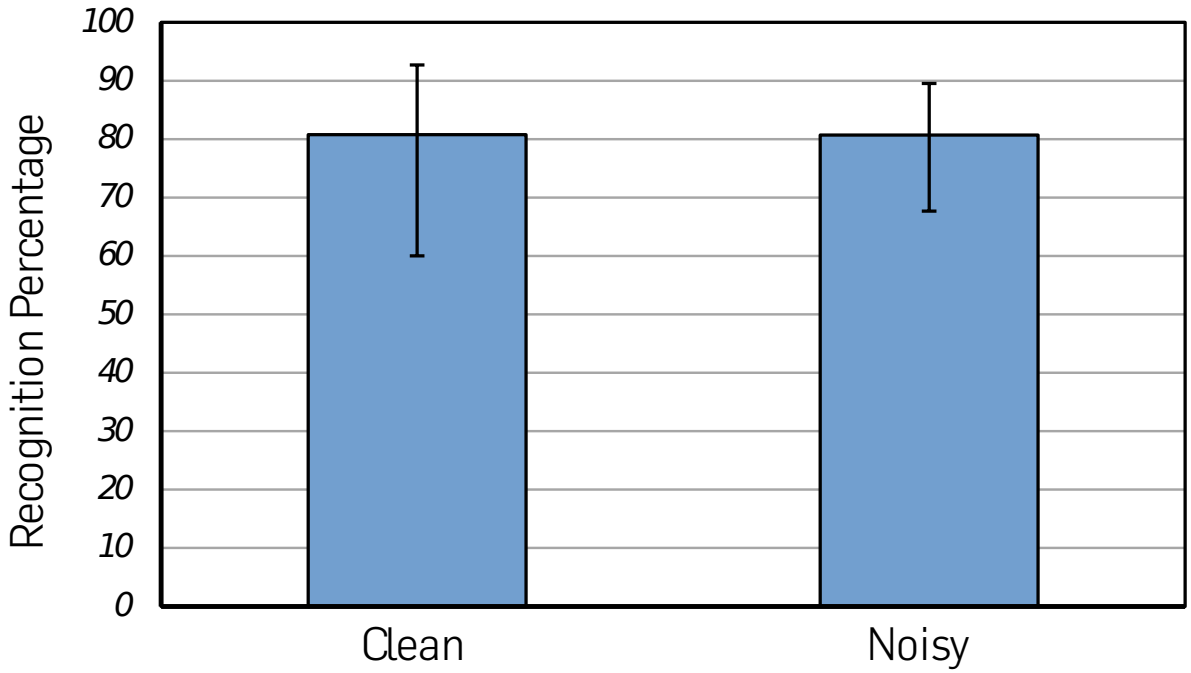

(Recorded with studio microphone)

# IMPACT OF MICROPHONE TYPE



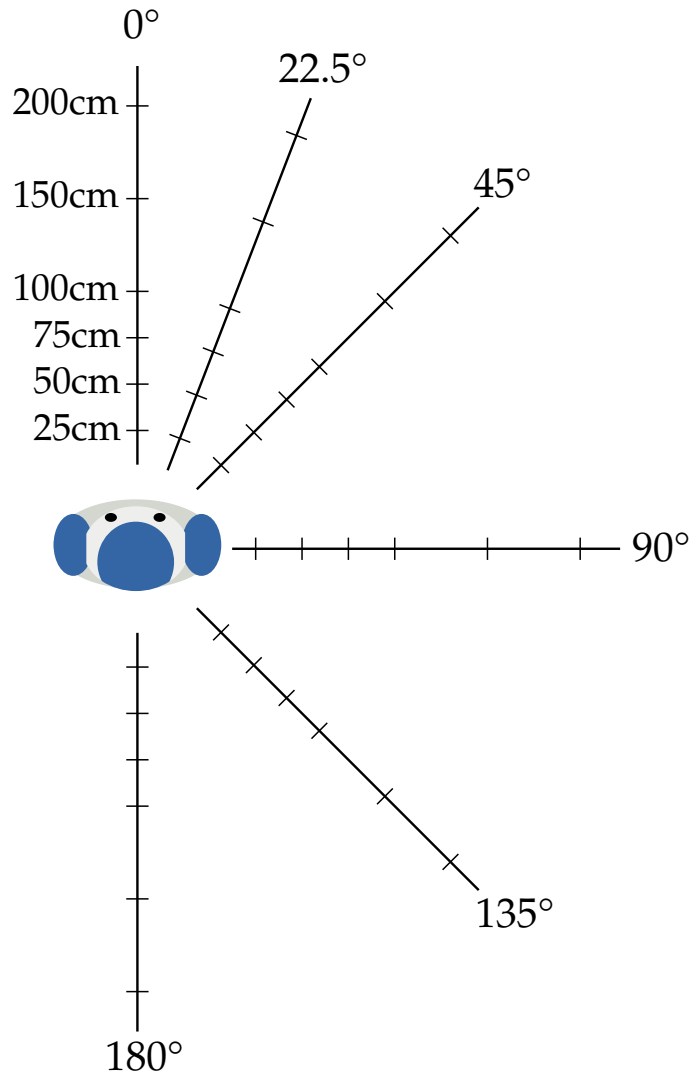

Recognition rate of numbers spoken by children, split by microphone type (62 utterances) using Nao's ASR engine (Nuance Vocon 4.7) constrained by a grammar containing only number words.

# IMPACT OF BACKGROUND NOISE

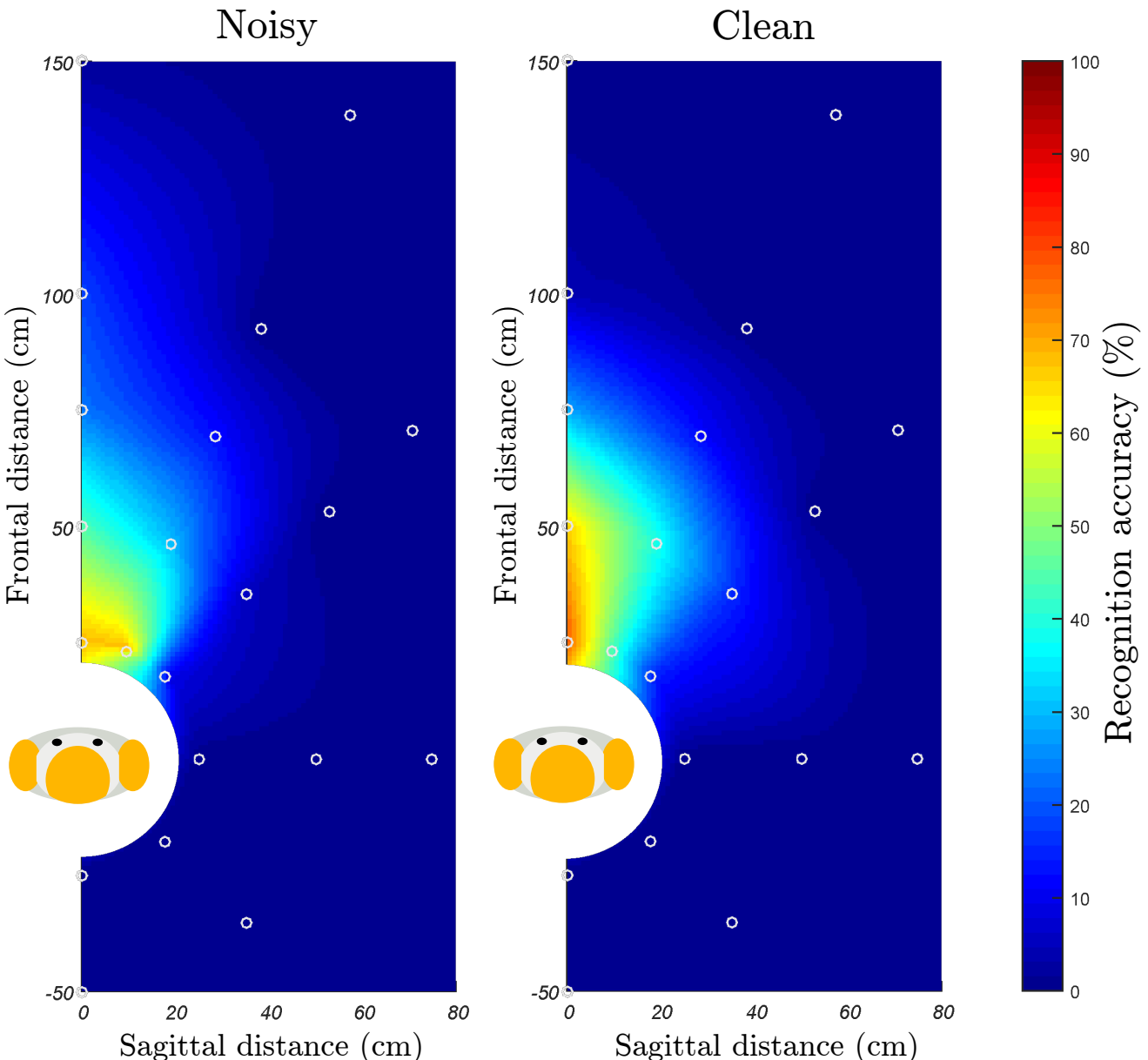



Recognition rate of number utterances spoken by children, split by background noise level (83 total utterances) using Nao's ASR engine (Nuance Vocon 4.7) constrained by a grammar containing only number words.

# LOCATION OF SPEAKER WRT. NAO



# LOCATION OF SPEAKER WRT. NAO



# 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.**

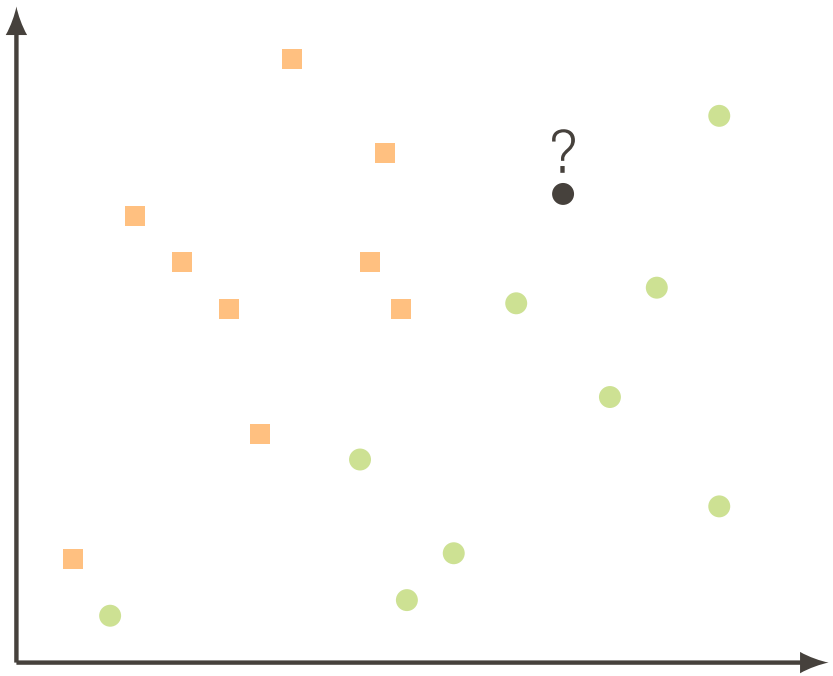
- **Classification** is deciding on which **category** a new observation belongs to based on training data.
- The training data contains data and known categories.
- Classification is a **supervised** learning algorithm.

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.

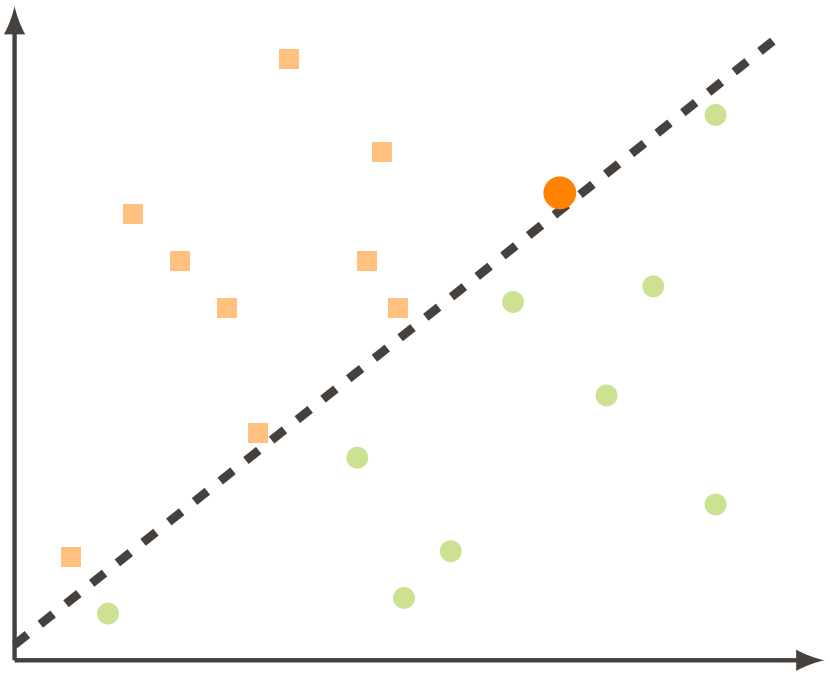
# CLASSIFICATION: EXAMPLE

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



# CLASSIFICATION: EXAMPLE

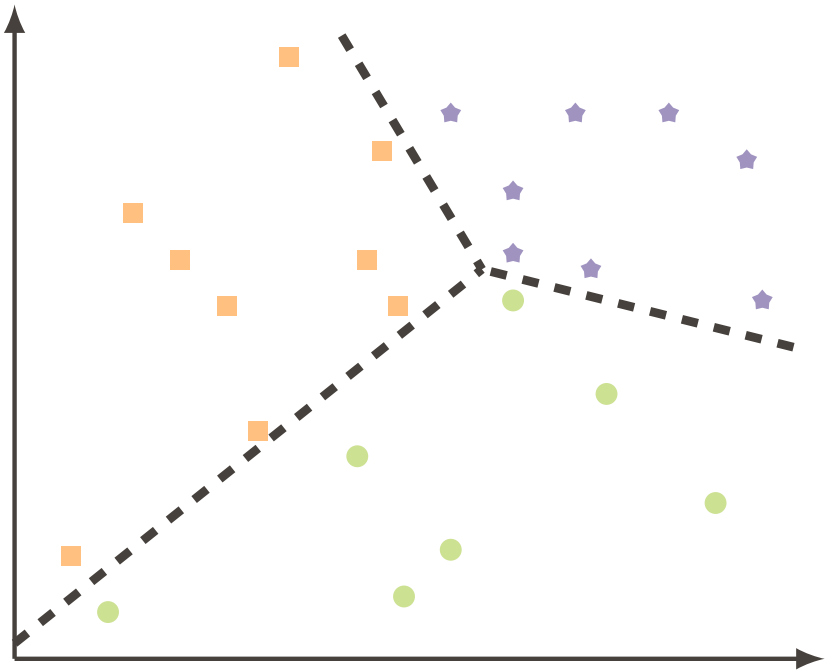
- Two dimensional problem
- Two categories, with training data for both categories
- To which category does a new observation belong?



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

# CLASSIFICATION: EXAMPLE

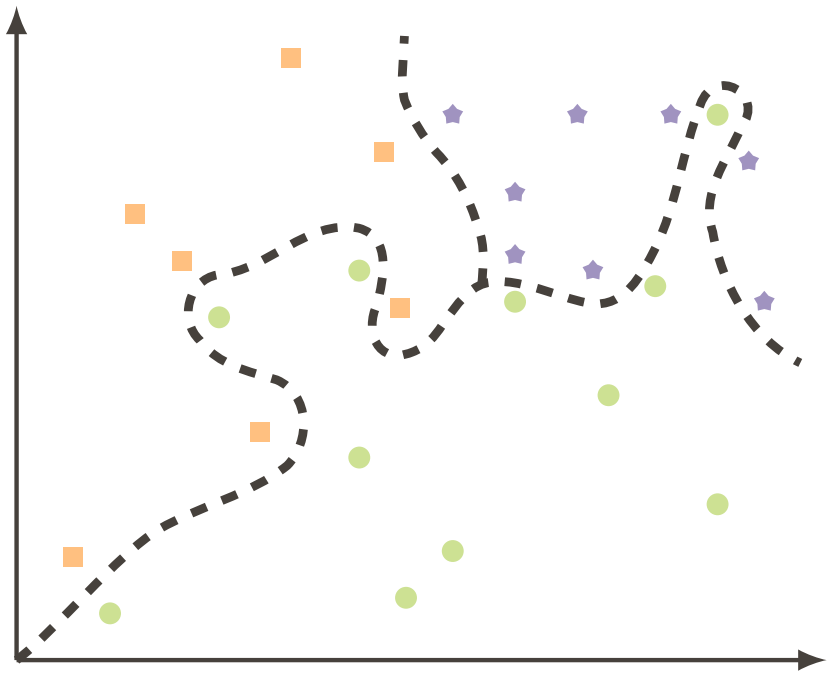
- Two dimensional problem
- Two categories, with training data for both categories
- To which category does a new observation belong?



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?**

# TWO CLASSIFICATION METHODS

Two methods will be explained here:

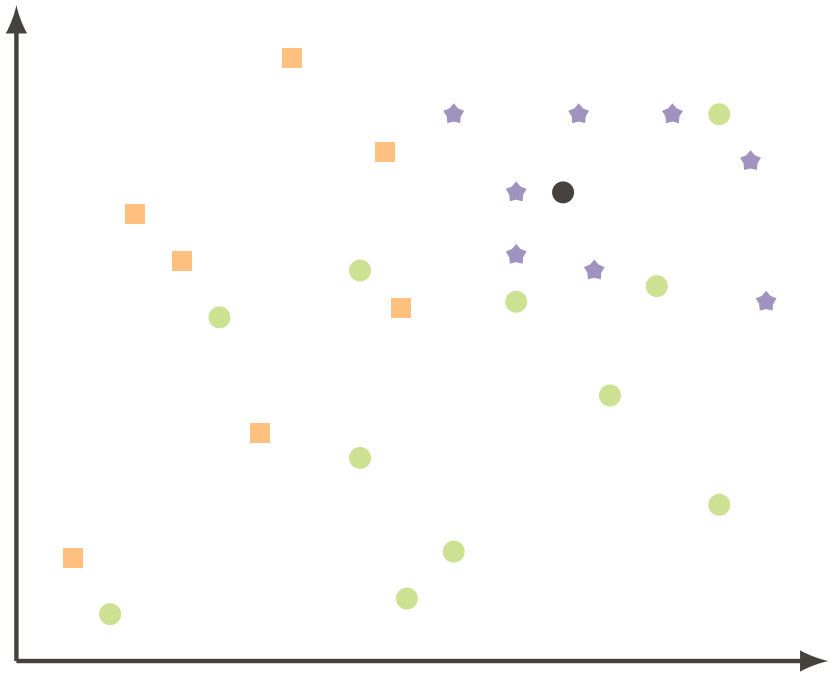
- *k*-nearest Neighbours (kNN)
- Support Vector Machines (SVM)

But there are hundreds of classifiers

- Decision trees
- Random forest
- Bayes classifiers
- Neural Networks
- ...

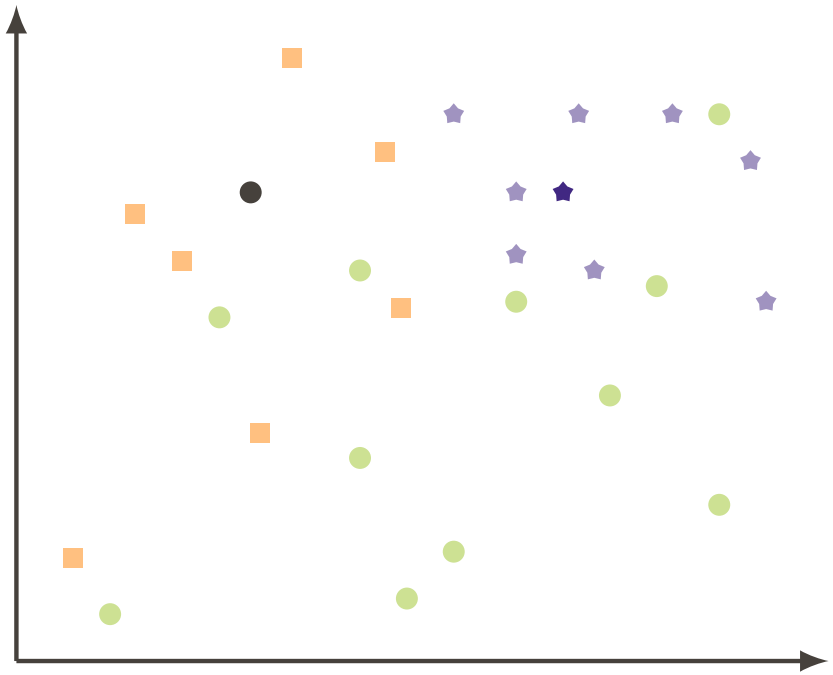
# *K*-NEAREST NEIGHBOURS

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



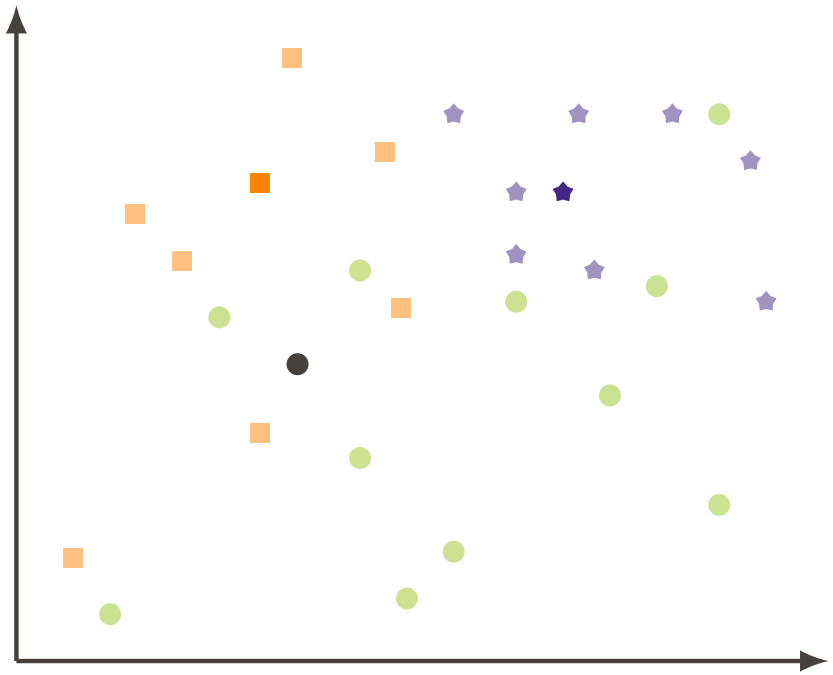
# *K*-NEAREST NEIGHBOURS

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



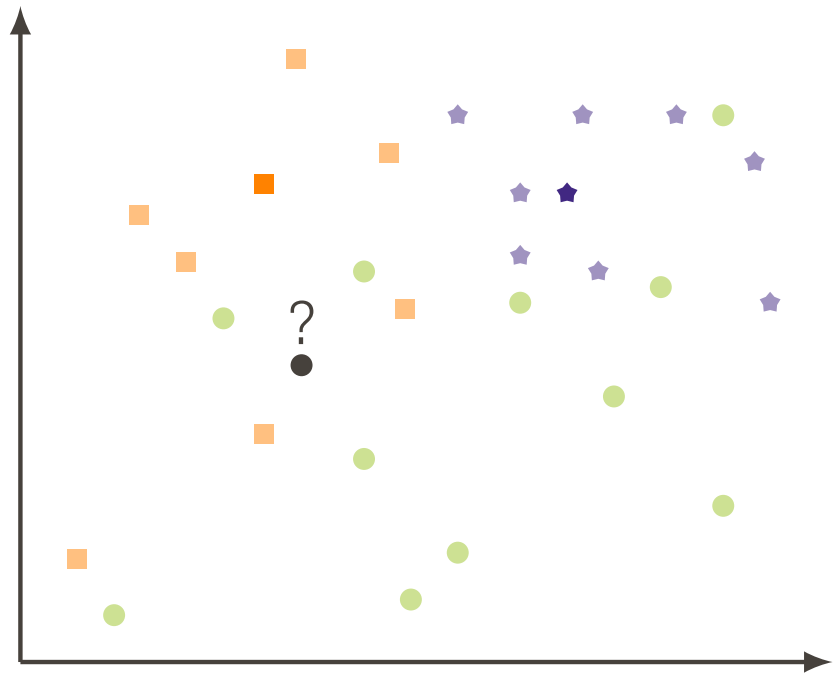
# *K*-NEAREST NEIGHBOURS

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

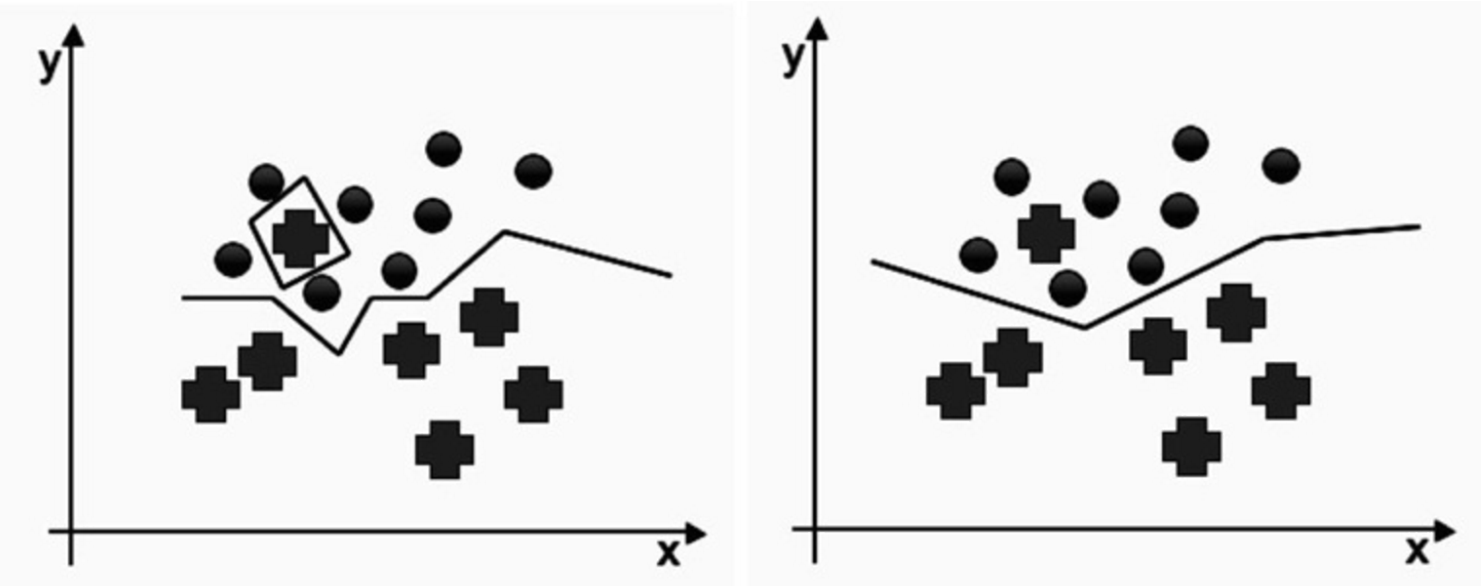
When an observation comes in, calculate the distance to the *k* nearest neighbours. The observation belongs to the class the most frequent amongst neighbours.



# CHOOSING *K*

- too small and it sensitive to classification noise.
- too large and classification accuracy decreases.

Decision border with $k = 1$ and $k = 2$:



# PYTHON IMPLEMENTATION OF *K*-NEAREST NEIGHBOURS

```
import numpy as np

def knn(k, data, categories, inputs):
  nb_inputs = np.shape(inputs)[0]
  closest = np.zeros(nb_inputs)

  for n in range(nb_inputs):
    # Compute distances
    distances = np.sum((data - inputs[n,:])\\2, axis=1)

    # Identify the nearest neighbours
    indices = np.argsort(distances, axis=0)

    classes = np.unique(categories[indices[:k]])

    if len(classes)==1:
        closest[n] = np.unique(classes)
    else:
        counts = np.zeros(max(classes) + 1)
        for i in range(k):
            counts[categories[indices[i]]] += 1
        closest[n] = np.argmax(counts, axis=0)
  return closest
```

# PYTHON IMPLEMENTATION OF *K*-NEAREST NEIGHBOURS

Alternatively, using `sklearn`:

```
from sklearn import neighbors

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

predictions = knns.predict(inputs)
```

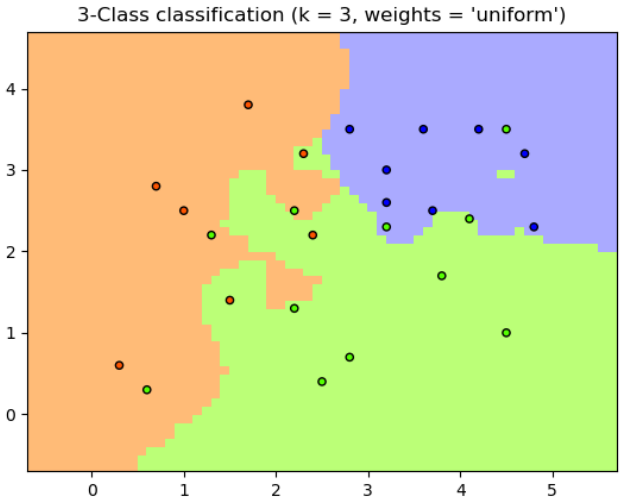
# PYTHON IMPLEMENTATION OF *K*-NEAREST NEIGHBOURS

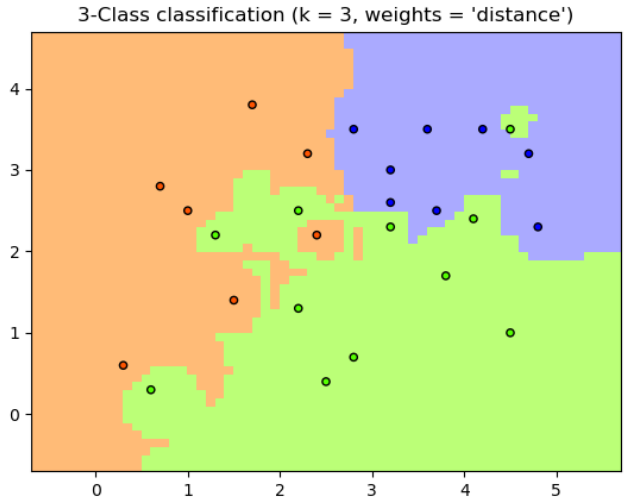
Complete example, with the data above:

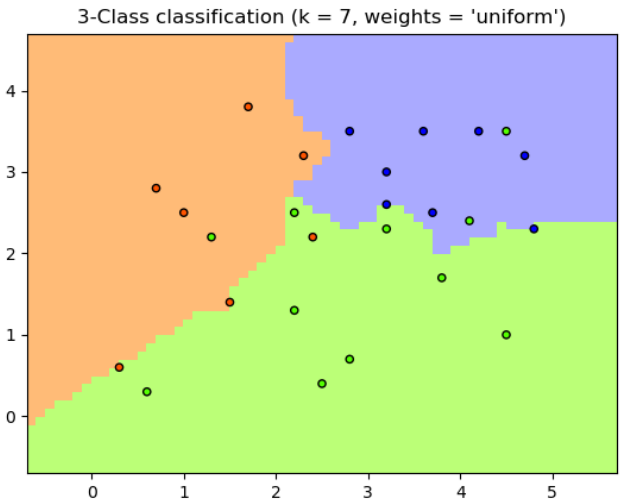
```
""" data.csv:
2.2,1.3,0 -> circles
3.2,2.3,0
...

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

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

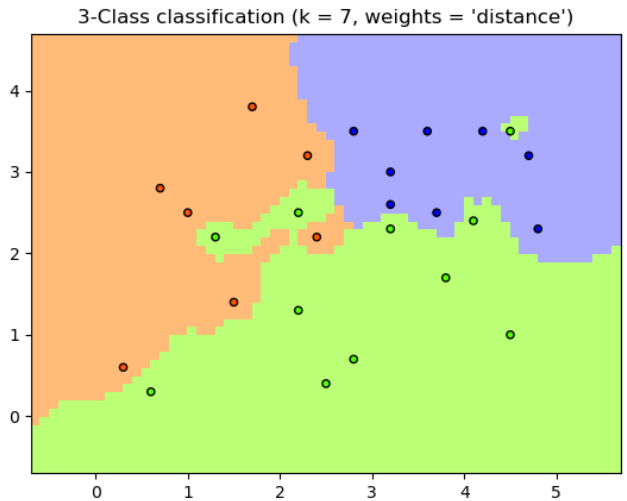
```
from numpy import genfromtxt
from sklearn import neighbors

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 (some work around are available, 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 (some work around are available, 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')
```

# THE CURSE OF DIMENSIONALITY

- to be effective, distance between neighbouring points must be $< d$. $d$ depends on the problem.

# THE CURSE OF DIMENSIONALITY

- to be effective, distance between neighbouring points must be $< d$. $d$ depends on the problem.
- In 1-D, we would need $n \sim \frac{1}{d}$ points.
- For instance, with one feature in $[0, 1]$ and $n$ training observations, new data will be no further away than $\frac{1}{n}$.

# THE CURSE OF DIMENSIONALITY

- to be effective, distance between neighbouring points must be $< d$. $d$ depends on the problem.
- In 1-D, we would need $n \sim \frac{1}{d}$ points.
- For instance, with one feature in $[0, 1]$ and $n$ training observations, new data will be no further away than $\frac{1}{n}$.
- Therefore, the nearest neighbour decision will be efficient as soon as $\frac{1}{n}$ is small compared to the scale of between-class feature variations.

# THE CURSE OF DIMENSIONALITY

- With $p$ features, you now require $n \sim \frac{1}{d^p}$ points.

# THE CURSE OF DIMENSIONALITY

- With $p$ features, you now require $n \sim \frac{1}{d^p}$ points.
- Let's say that we require 10 points in one dimension: now $10^p$ points are required in $p$ dimensions to pave the $[0, 1]$ space.
- As $p$ becomes large, the number of training points required for a good estimator grows exponentially.

# THE CURSE OF DIMENSIONALITY

- With $p$ features, you now require $n \sim \frac{1}{d^p}$ points.
- Let's say that we require 10 points in one dimension: now $10^p$ points are required in $p$ dimensions to pave the $[0, 1]$ space.
- As $p$ becomes large, the number of training points required for a good estimator grows exponentially.
- For example, if each feature is just a single 8-bytes number, then an effective k-NN estimator with $p \sim 20$ dimensions would require more training data than the current estimated size of the entire internet ($\pm$1000 Exabytes or so).

This is called **the curse of dimensionality**.

# 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.

# 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.

As opposed to *k*-nearest Neighbours, SVM are more difficult to understand (although no knowledge of the algorithms internals is needed to use it).

# SUPPORT VECTOR MACHINES

SVMs do not work well on extremely large datasets, since the computations do not scale well with the number of training examples, and so become computationally very expensive.

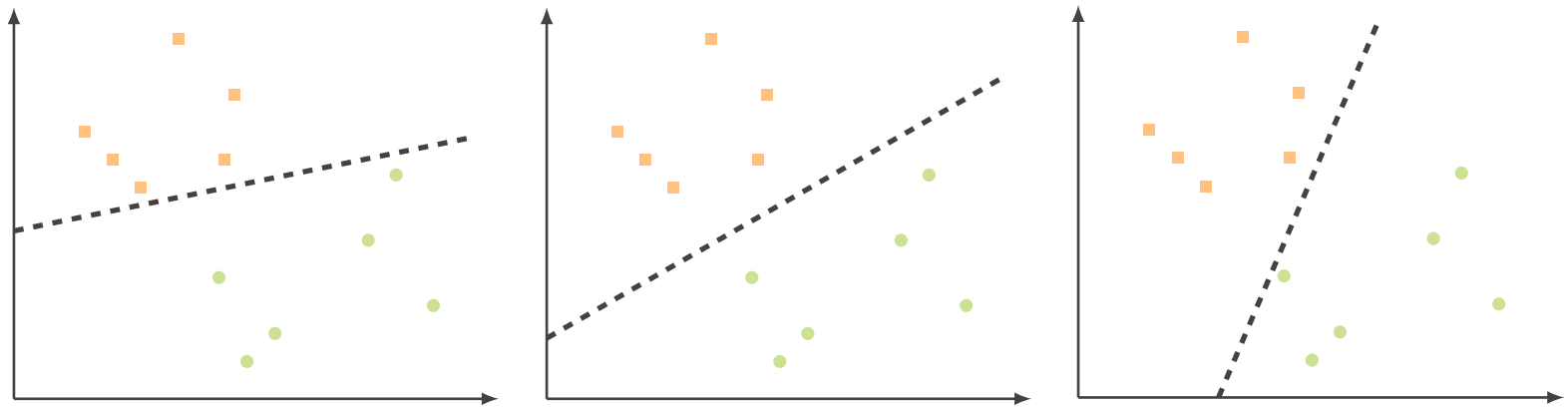
## Complexity

Training a SVM requires $O(m^3)$ operations and $O(m^2)$ memory, with $m$ being the number of training samples. For example, if you have 10,000 training samples, the training will take in the order of $10^{12}$ operations (imagine you can do 100,000 operations/second, you will need $10^8$ seconds $\equiv$ 3 years to train your SVM and $10^{12}$ memory units.

For more information on "big-O notation", see complexity of algorithms

# SVM PRINCIPLE

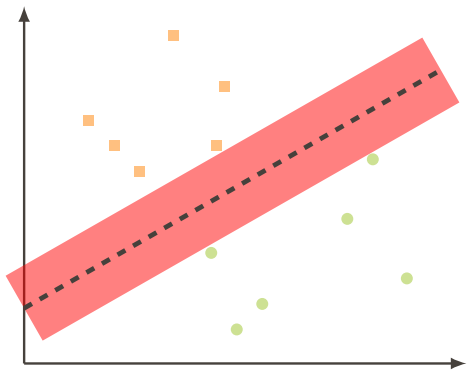
Three different classification lines. All are correct, but is there any reason why one is better than the others?



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?



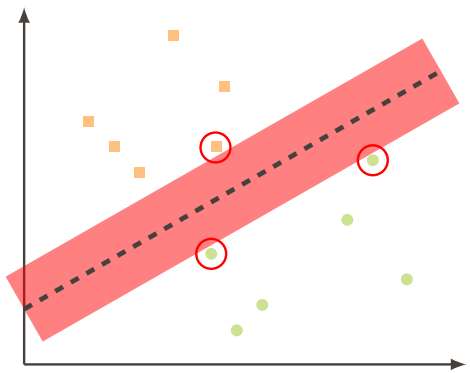
Intuitively, the line that is most distant to the training data is the best division.

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**.

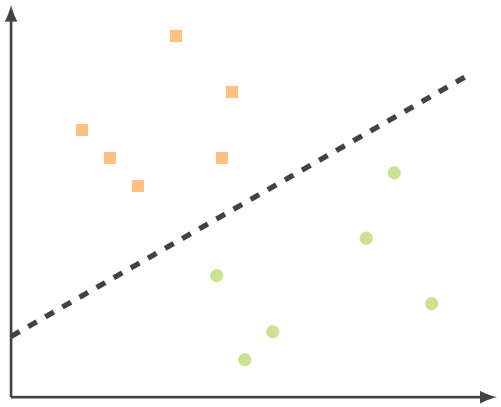
The data points nearest the classifier are called **support vectors**.



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.

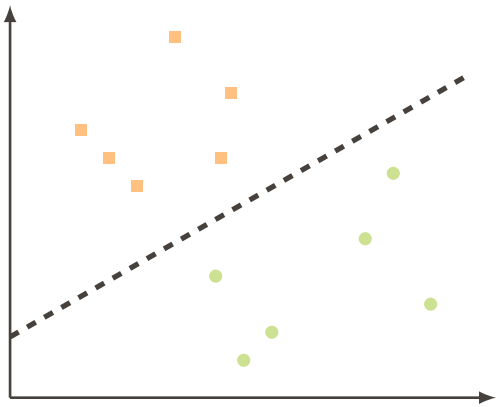
# LINEAR CLASSIFIER



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

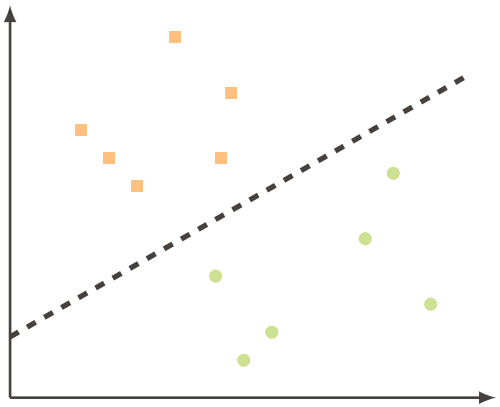


**w** has to be calculated such that:

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

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

# LINEAR CLASSIFIER



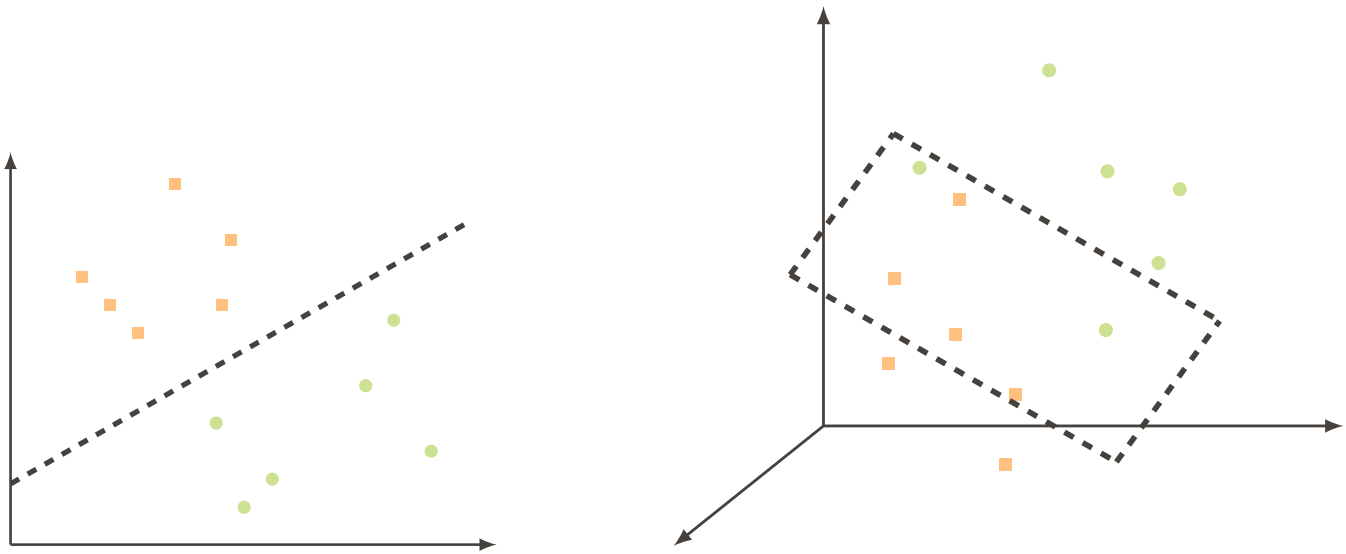
**w** has to be calculated such that:

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

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

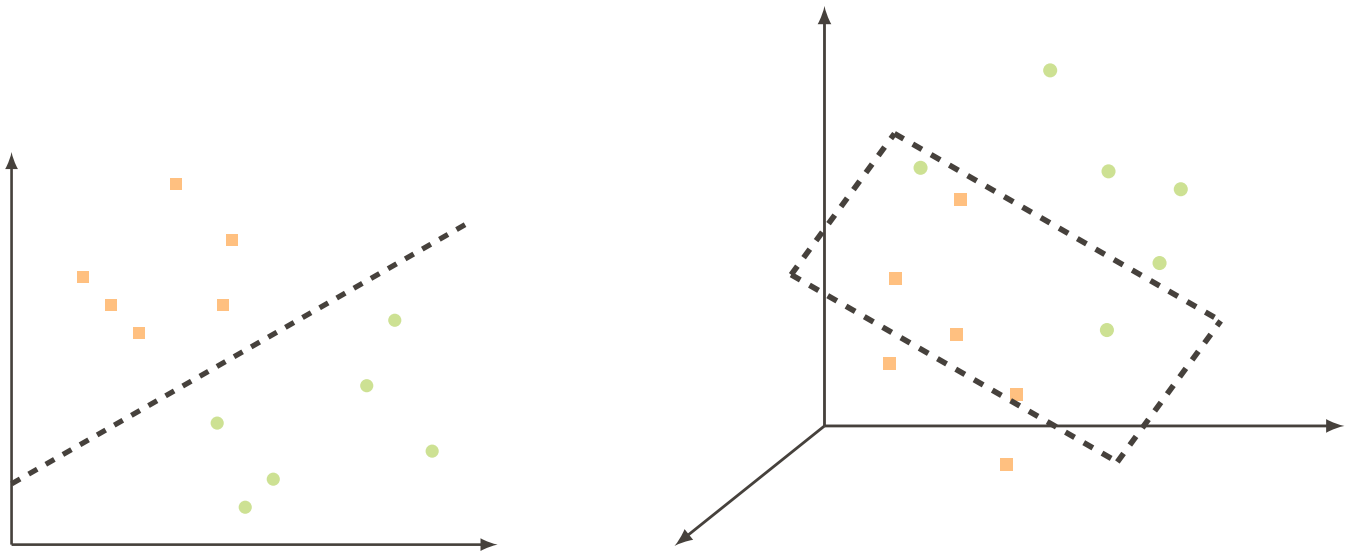
for kNNs, we have to carry along the training data; with a linear classifier, once **w** is found, we can discard the training data

# LINEAR CLASSIFIER



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

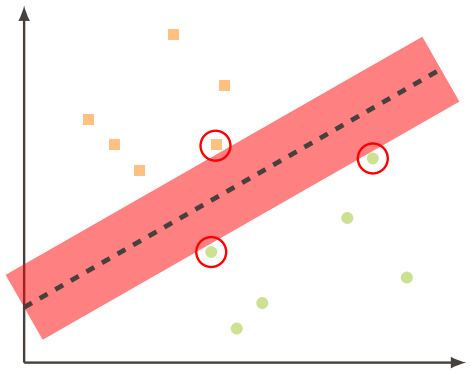
# LINEAR CLASSIFIER



- 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...

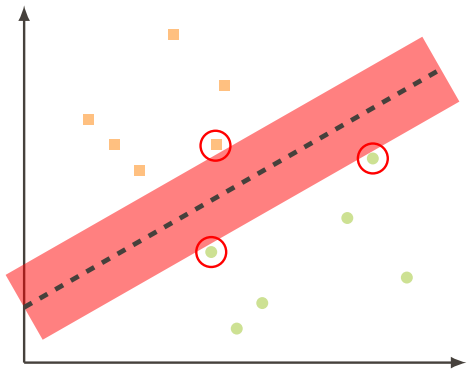
# SUPPORT VECTOR CLASSIFIER



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

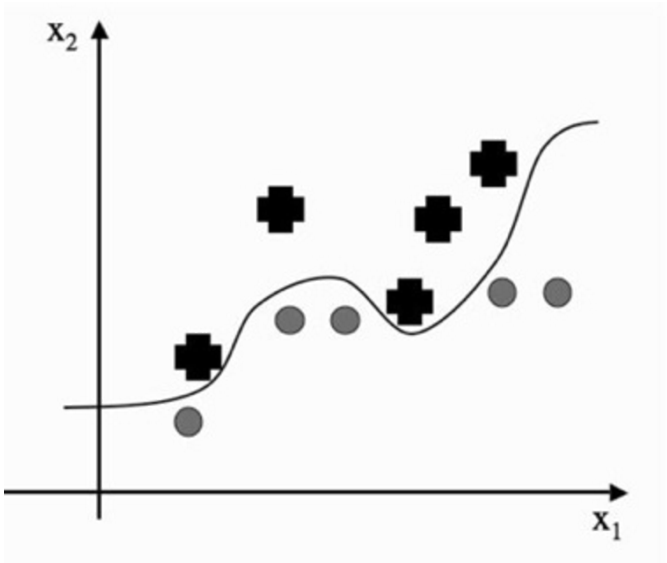
- It needs to correctly classify the training data,
- and needs the margin from the classifier to be as large as possible.

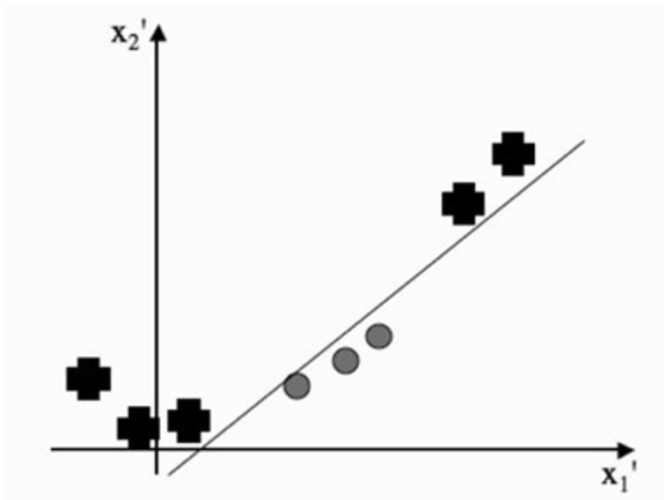
A **quadratic programming solver** is used to find the optimal values for **w** and *b*.

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



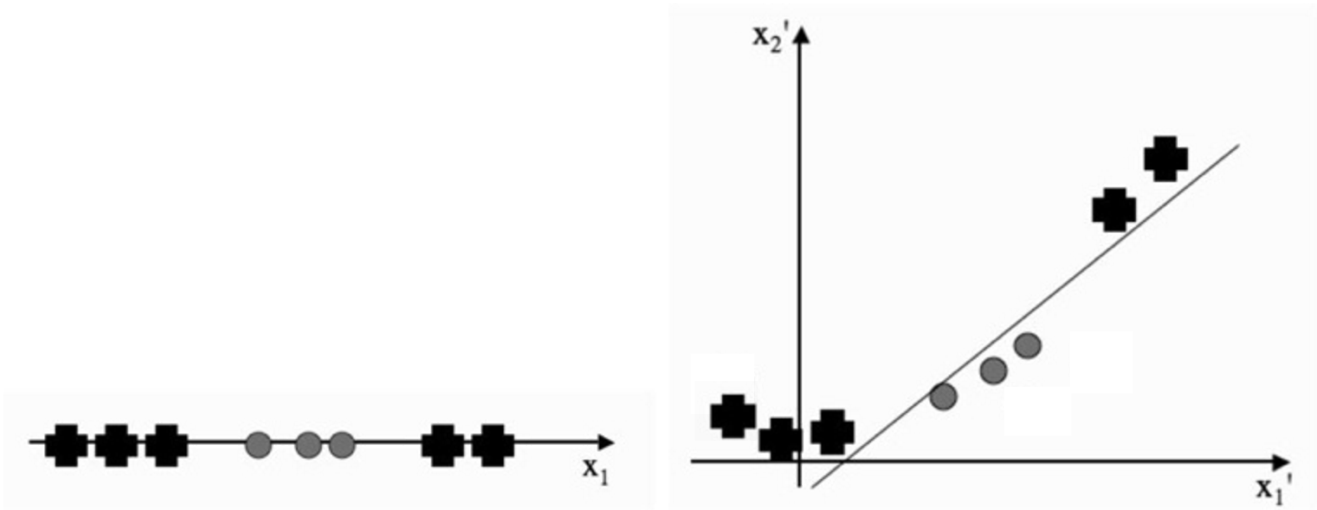



# FEATURE MAPS

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

Example:

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



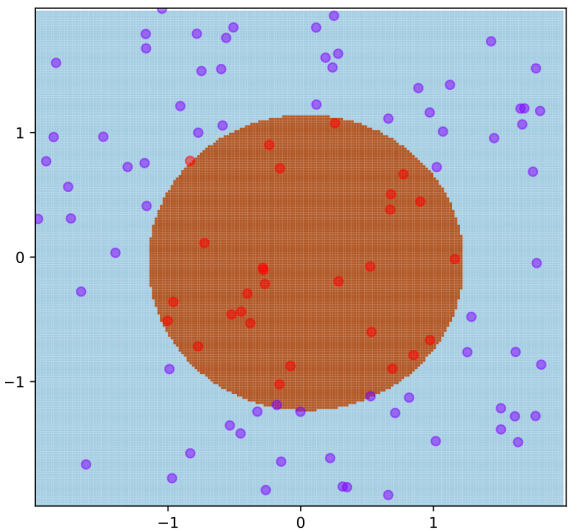

$\rightarrow$ now, linearly separable

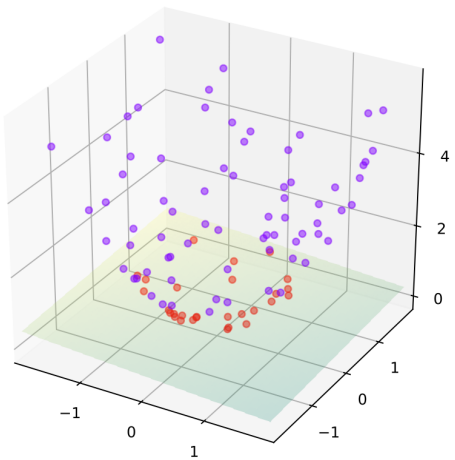
# FEATURE MAPS

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

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

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 $\kappa$ and $\delta$, and kernel:

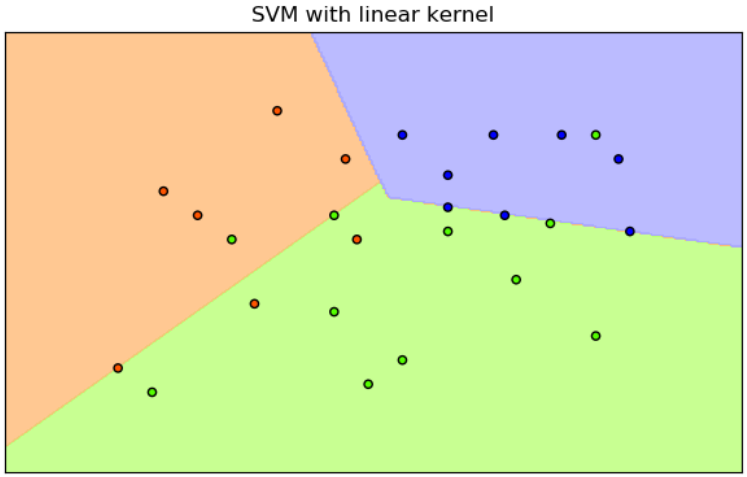
$$K(x, y) = tanh(\kappa x^T y - \delta)$$

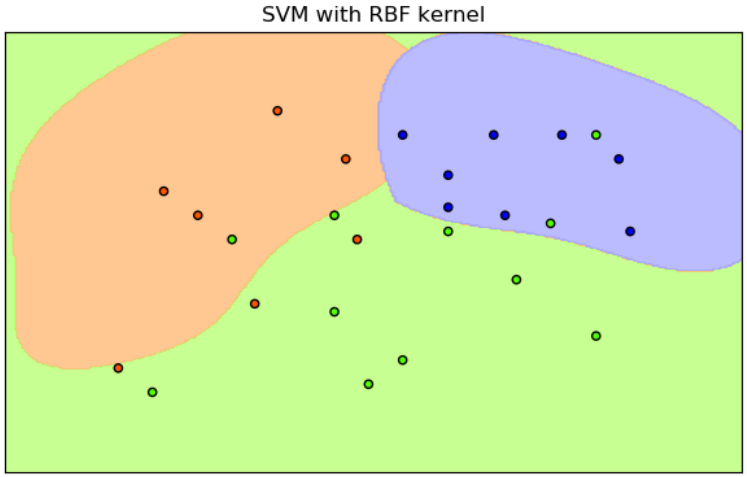
- **Radial basis function** expansions of the $x_k$ with parameter $\sigma$ and kernel:

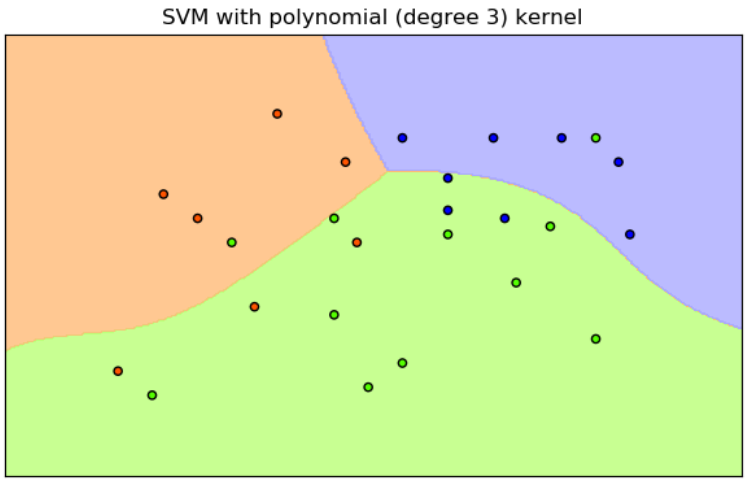
$$K(x, y) = exp(-\frac{(x - y)^2}{2\sigma^2})$$

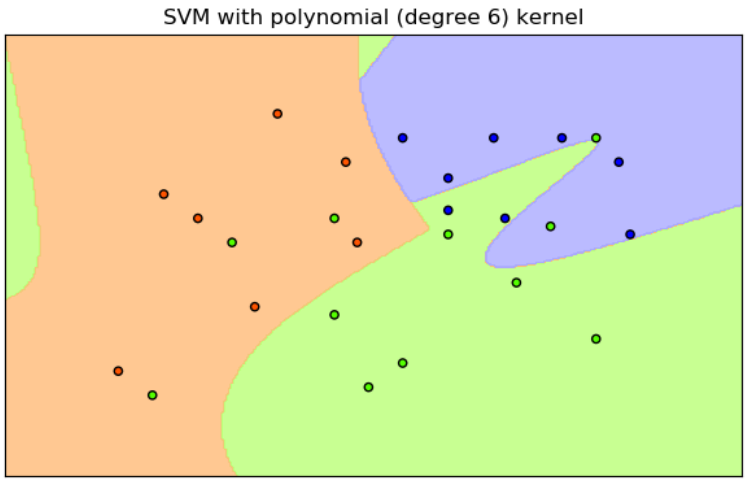
(a Gaussian kernel)

# 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
...
"""
```

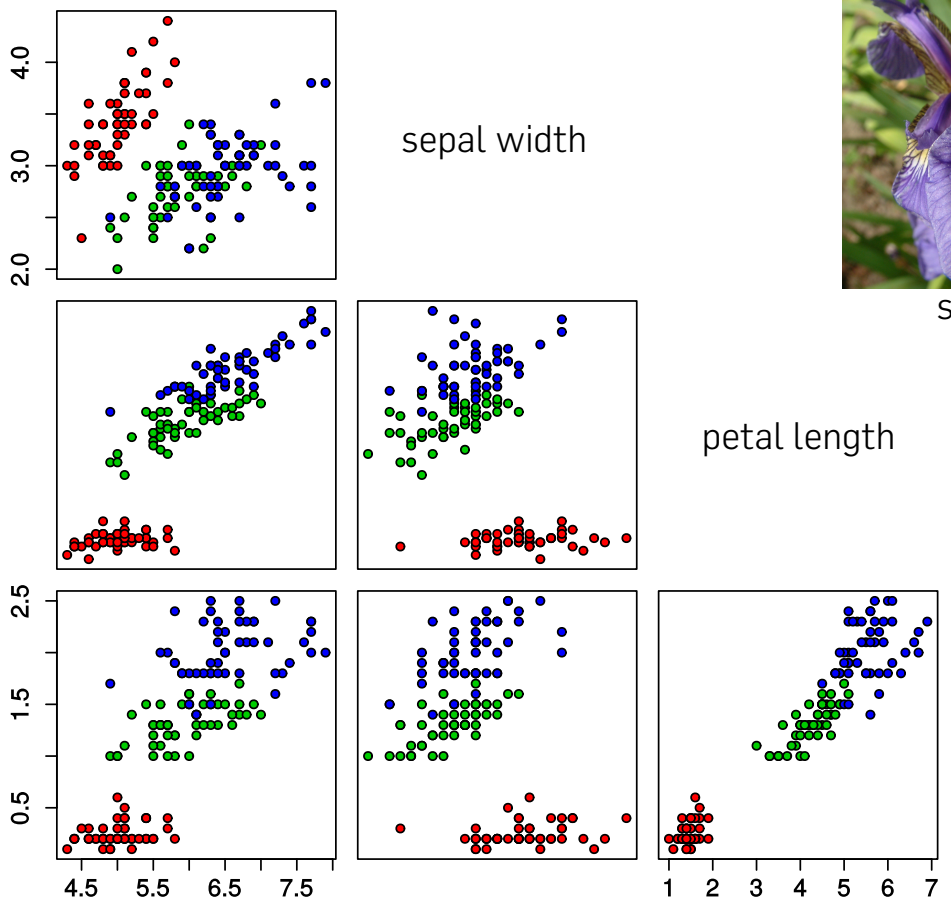
```
from numpy import genfromtxt
from sklearn import svm

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.]
```

# CHOOSING THE KERNEL

Choosing which kernel to use and the parameters in these kernels is a tricky problem. While there is some theory based on something known as the Vapnik–Chernik dimension that can be applied, most people just experiment with different values and find one that works.

# THE IRIS DATA SET

**Iris data (red: setosa, green: versicolor, blue: virginica)**




setosa

virginica

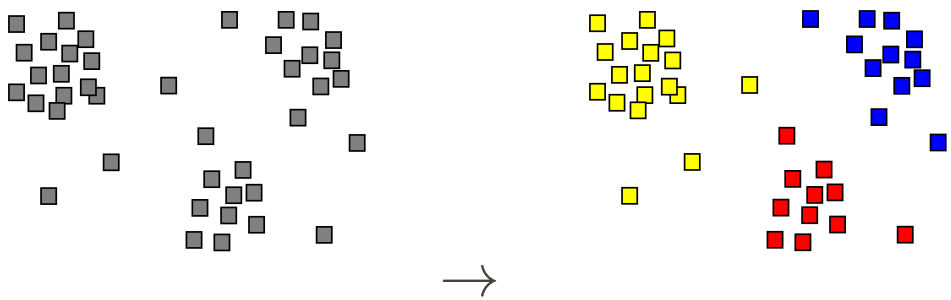

versicolor

# 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 $\rightarrow$ **unsupervised learning**.

$\rightarrow$ Used to find patterns in the data.



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

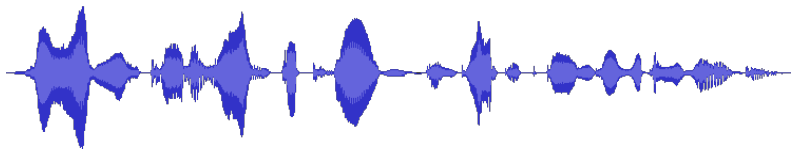
# HOW TO CLASSIFY SOCIAL SIGNAL?

# HOW TO CLASSIFY SOCIAL SIGNALS?

Raw signals will in most cases require pre-processing to extract features.

The raw social signal (audio or video) requires pre-processing to extract between 10 and over a 1000 **features**.

- A raw signals contains too much data, and cannot be fed to the classifier immediately.



- Pre-processing extracts feature data which is relevant for the information which we are after (pitch, volume/energy, duration, formant frequencies, …)
- These features then form the input for the classifier.

For more information see, for example, Liang et al. (2005) Feature analysis and extraction for audio automatic classification, Systems, Man and Cybernetics,

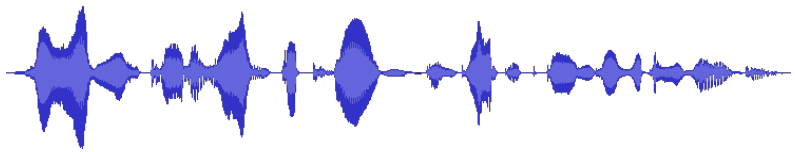
# HOW TO CLASSIFY SOCIAL SIGNALS?

Raw signals will in most cases require pre-processing to extract features.

The raw social signal (audio or video) requires pre-processing to extract between 10 and over a 1000 **features**.

- A raw signals contains too much data and cannot be fed to the classifier immediately.

An exception to this are Convolutional Neural Networks, which can deal with unprocessed data



- Pre-processing extracts feature data which is relevant for the information which we are after (pitch, volume/energy, duration, formant frequencies, …)
- These features then form the input for the classifier.

For more information see, for example, Liang et al. (2005) Feature analysis and extraction for audio automatic classification, Systems, Man and Cybernetics,

# EXAMPLE: RECOGNISING GENDER FROM SPEECH

Can we automatically recognise someone's gender from speech?

3,168 recorded voice samples, collected from male and female speakers.

- Examples from the database: male (US), female (US), male (Scotish)

The voice samples are pre-processed by acoustic analysis in R using the `seewave`, `warbleR` and `tuneR` packages, with an analysed frequency range of 0hz-280Hz (fundamental frequency of human speech).

*Source: data and more information*

# THE 20 FEATURES USED FOR GENDER CLASSIFICATION

- **meanfreq**: mean frequency (in kHz)
- **sd**: standard deviation of frequency
- **median**: median frequency (in kHz)
- **Q25**: first quantile (in kHz)
- **Q75**: third quantile (in kHz)
- **IQR**: interquantile range (in kHz)
- **skew**: skewness (see note in specprop description)
- **kurt**: kurtosis (see note in specprop description)
- **sp.ent**: spectral entropy
- **sfm**: spectral flatness
- **mode**: mode frequency
- **centroid**: frequency centroid (see specprop)
- **peakf**: peak frequency (frequency with highest energy)
- **meanfun**: average of fundamental frequency measured across acoustic signal

# THE 20 FEATURES USED FOR GENDER CLASSIFICATION

- **minfun**: minimum fundamental frequency measured across acoustic signal
- **maxfun**: maximum fundamental frequency measured across acoustic signal
- **meandom**: average of dominant frequency measured across acoustic signal
- **mindom**: minimum of dominant frequency measured across acoustic signal
- **maxdom**: maximum of dominant frequency measured across acoustic signal
- **dfrange**: range of dominant frequency measured across acoustic signal
- **modindx**: modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range

# EXAMPLE: RECOGNISING GENDER FROM SPEECH

| meanfreq | sd | median | Q25 | Q75 | IQR | skew | kurt | sp.ent | sfm | mode | centroid | meanfun | minfun | maxfun | meandom | mindom | maxdom | dfrange | modindx | label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.05978 | 0.06424 | 0.03203 | 0.01507 | 0.09019 | 0.07512 | 12.8635 | 274.403 | 0.89337 | 0.49192 | 0 | 0.05978 | 0.08428 | 0.0157 | 0.27586 | 0.00781 | 0.00781 | 0.00781 | 0 | 0 | male |
| 0.06601 | 0.06731 | 0.04023 | 0.01941 | 0.09267 | 0.07325 | 22.4233 | 634.614 | 0.89219 | 0.51372 | 0 | 0.06601 | 0.10794 | 0.01583 | 0.25 | 0.00901 | 0.00781 | 0.05469 | 0.04688 | 0.05263 | male |
| 0.07732 | 0.08383 | 0.03672 | 0.0087 | 0.13191 | 0.12321 | 30.7572 | 1024.93 | 0.84639 | 0.47891 | 0 | 0.07732 | 0.09871 | 0.01566 | 0.27119 | 0.00799 | 0.00781 | 0.01563 | 0.00781 | 0.04651 | male |
| 0.192275 | 0.060818 | 0.21913 | 0.130952 | 0.242491 | 0.111539 | 1.891999 | 6.680009 | 0.915249 | 0.461751 | 0.244465 | 0.192275 | 0.114544 | 0.016615 | 0.210526 | 0.518682 | 0.03125 | 4.164063 | 4.132813 | 0.118491 | male |
| 0.203083 | 0.058876 | 0.232857 | 0.134286 | 0.247143 | 0.112857 | 2.858921 | 12.34421 | 0.852594 | 0.320577 | 0.246429 | 0.203083 | 0.108871 | 0.023426 | 0.15534 | 0.4375 | 0.21875 | 0.734375 | 0.515625 | 0.296296 | male |
| 0.166858 | 0.076829 | 0.202062 | 0.112096 | 0.228522 | 0.116426 | 1.971154 | 7.121826 | 0.937182 | 0.624363 | 0.216014 | 0.166858 | 0.095246 | 0.016563 | 0.142857 | 0.310547 | 0.15625 | 0.734375 | 0.578125 | 0.3 | male |
| 0.187391 | 0.059659 | 0.202846 | 0.125692 | 0.235809 | 0.110116 | 1.722606 | 6.693799 | 0.923242 | 0.464031 | 0.232549 | 0.187391 | 0.098694 | 0.027972 | 0.141593 | 0.324405 | 0.164063 | 0.59375 | 0.429688 | 0.324545 | male |
| 0.194088 | 0.061379 | 0.216466 | 0.127631 | 0.246827 | 0.119197 | 1.490315 | 4.321145 | 0.88923 | 0.364435 | 0.249639 | 0.194088 | 0.10925 | 0.036782 | 0.231884 | 0.495793 | 0.117188 | 2.164063 | 2.046875 | 0.192748 | male |
| 0.185908 | 0.062359 | 0.198432 | 0.133178 | 0.242034 | 0.108856 | 1.396273 | 5.493992 | 0.934298 | 0.521624 | 0.260127 | 0.185908 | 0.11307 | 0.020434 | 0.195122 | 0.696514 | 0.140625 | 5.414063 | 5.273438 | 0.165689 | male |
| 0.178028 | 0.070548 | 0.19 | 0.127436 | 0.242821 | 0.115385 | 2.149795 | 9.1742 | 0.945836 | 0.637708 | 0.251795 | 0.178028 | 0.116113 | 0.020101 | 0.275862 | 0.983854 | 0.03125 | 5.109375 | 5.078125 | 0.225199 | male |
| 0.187952 | 0.065855 | 0.203059 | 0.132068 | 0.245099 | 0.113031 | 1.480657 | 5.018319 | 0.934454 | 0.582426 | 0.243909 | 0.187952 | 0.111375 | 0.019704 | 0.181818 | 0.519737 | 0.0625 | 2.84375 | 2.78125 | 0.17603 | male |
| 0.208232 | 0.033483 | 0.214075 | 0.186861 | 0.231538 | 0.044678 | 2.569042 | 10.79804 | 0.86938 | 0.16029 | 0.226299 | 0.208232 | 0.186654 | 0.023121 | 0.258065 | 0.79974 | 0.171875 | 3.242188 | 3.070313 | 0.229271 | female |
| 0.199387 | 0.03544 | 0.196046 | 0.180719 | 0.226471 | 0.045752 | 2.342196 | 9.294968 | 0.860738 | 0.210884 | 0.18415 | 0.199387 | 0.159956 | 0.027119 | 0.271186 | 0.898438 | 0.007813 | 5.976563 | 5.96875 | 0.189915 | female |
| 0.195679 | 0.031613 | 0.193891 | 0.181785 | 0.21166 | 0.029874 | 3.189926 | 15.15865 | 0.850763 | 0.201765 | 0.193501 | 0.195679 | 0.183362 | 0.017778 | 0.25 | 0.935397 | 0.1875 | 5.921875 | 5.734375 | 0.193079 | female |
| 0.195605 | 0.033526 | 0.197941 | 0.179728 | 0.210751 | 0.031022 | 3.079277 | 14.56234 | 0.861835 | 0.22135 | 0.197341 | 0.195605 | 0.182781 | 0.029685 | 0.266667 | 1.055226 | 0.015625 | 6.25 | 6.234375 | 0.196491 | female |
| 0.200325 | 0.033138 | 0.205588 | 0.179753 | 0.223236 | 0.043483 | 2.107451 | 7.372721 | 0.869482 | 0.179791 | 0.223418 | 0.200325 | 0.17806 | 0.037915 | 0.266667 | 1.13151 | 0.164063 | 5.609375 | 5.445313 | 0.202108 | female |
| 0.212681 | 0.042392 | 0.212152 | 0.180529 | 0.25138 | 0.070851 | 1.142382 | 3.271853 | 0.895565 | 0.198001 | 0.19654 | 0.212681 | 0.169677 | 0.017837 | 0.266667 | 1.740885 | 0.148438 | 7 | 6.851563 | 0.35467 | female |
| 0.198039 | 0.030396 | 0.198105 | 0.183464 | 0.212974 | 0.02951 | 2.118279 | 7.139244 | 0.857263 | 0.177914 | 0.189412 | 0.198039 | 0.188897 | 0.025932 | 0.242424 | 0.508878 | 0.109375 | 1.507813 | 1.398438 | 0.324904 | female |
| 0.218552 | 0.037574 | 0.220555 | 0.200416 | 0.246274 | 0.045858 | 2.4775 | 11.06604 | 0.877562 | 0.188399 | 0.220555 | 0.218552 | 0.182308 | 0.020725 | 0.275862 | 0.474609 | 0.007813 | 1.492188 | 1.484375 | 0.199624 | female |
| 0.198203 | 0.031488 | 0.195094 | 0.182032 | 0.218269 | 0.036238 | 2.643353 | 12.04094 | 0.862682 | 0.174607 | 0.183085 | 0.198203 | 0.186066 | 0.07619 | 0.238806 | 0.714154 | 0.171875 | 6.171875 | 6 | 0.113542 | female |
| 0.202647 | 0.031964 | 0.198973 | 0.184434 | 0.223804 | 0.03937 | 2.44728 | 10.35293 | 0.864479 | 0.165662 | 0.185904 | 0.202647 | 0.184609 | 0.021769 | 0.25 | 1.107759 | 0.070313 | 6.140625 | 6.070313 | 0.19701 | female |
| 0.217759 | 0.031261 | 0.223285 | 0.1991 | 0.237762 | 0.038662 | 2.038032 | 6.67446 | 0.861819 | 0.15492 | 0.226691 | 0.217759 | 0.193159 | 0.017335 | 0.271186 | 1.109086 | 0.007813 | 5.914063 | 5.90625 | 0.177407 | female |
| 0.191456 | 0.030422 | 0.19173 | 0.172434 | 0.212874 | 0.04044 | 2.109024 | 7.296761 | 0.85787 | 0.175168 | 0.172023 | 0.191456 | 0.179518 | 0.028269 | 0.271186 | 0.642188 | 0.171875 | 3.429688 | 3.257813 | 0.174889 | female |

# EXAMPLE: RECOGNISING GENDER FROM SPEECH

Performance

- kNN (k = 7): 97.8% classified correctly.
- SVM: 97.5% classified correctly.

Recognising gender from speech is easy and robust.

All classification algorithms can deal with this problem.

# CONCLUSION

Social signal processing is extracting relevant information from the social environment.

Some work relatively well

- Face **detection**, voice activity detection, gender classification, ...

Some work, but need improvement

- Gaze detection, basic emotion recognition, face **recognition**, speech recognition, ...

But still many open problems remaining

- Complex real-word affect and emotion recognition (e.g. embarrassment, pride).
- Speech recognition for atypical speakers (children, elderly), multi-party interaction, ...

# That's all for today, folks!

Questions:
Portland Square B316 or **severin.lemaignan@plymouth.ac.uk**

Slides:
github.com/severin-lemaignan/lecture-hri-social-signal-processing