MIDDLE EAST TECHNICAL UNIVERSITY

DI504 Foundations of Deep Learning

Features, Classification and Fundamentals of Machine Learning

Welcome again!

This is DI504, Foundations of Deep Learning

- This week, we will start our discussion with the so-called “Feature Spaces”.
- And we will continue with some machine learning fundamentals.

Features!

This is DI504, Foundations of Deep Learning

- This week, we will start our discussion with the so-called “Feature Spaces”.
- Let’s start with a toy example: Let’s separate apples and pears.

Features!

This is DI504, Foundations of Deep Learning

- This week, we will start our discussion with the so-called “Feature Spaces”.
- Let’s start with a toy example: Let’s separate apples and pears.
  But how to define an apple and a pear
  - Color : Red-orange-yellow-green ?
  - Shape : Spherical – oval ?
-

Features!

This is DI504, Foundations of Deep Learning

- This week, we will start our discussion with the so-called “Feature Spaces”.
- Let’s start with a toy example: Let’s separate apples and pears.
  But how to define an apple and a pear
  - Color : Red-orange-yellow-green ?
  - Shape : Spherical – oval ?
- Let’s use these two measures only.
  When we use two measures, we actually define apples and pears in a 2D world.

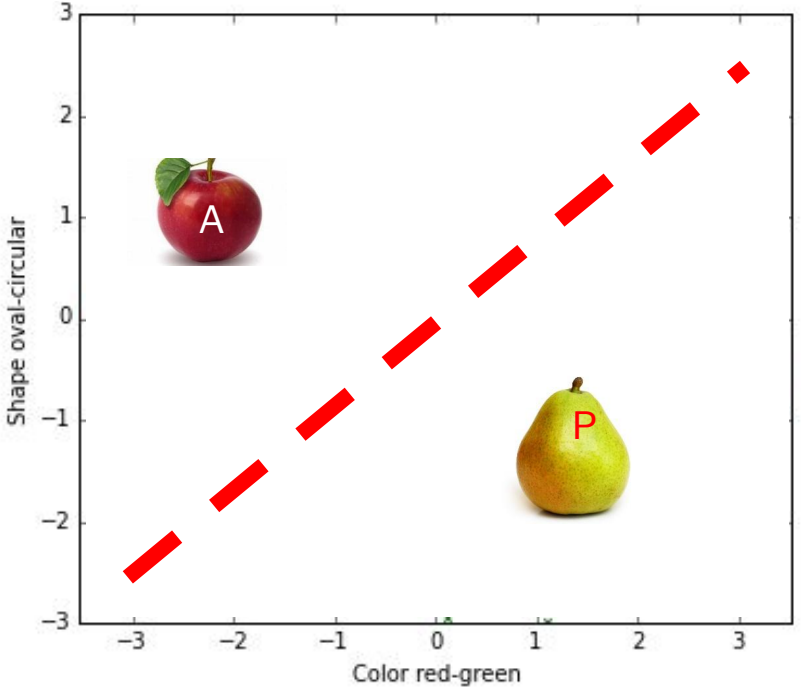
Sensors

I will be using two sensors, that output numerical values.

- Numerical values help me create “distances”
  - Color:
    - Red-orange-yellow-green:
      - a dimension from -3 to +3.
      - -3 is green
      - +3 is red
  - Shape:
    - oval – spherical:
      - a dimension from -3 to +3
      - -3 is oval
      - +3 spherical

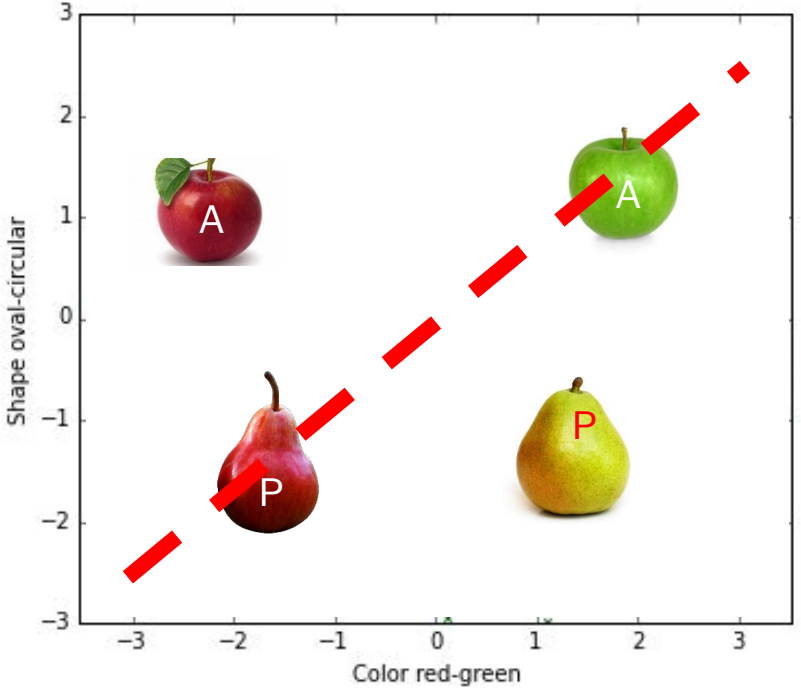
Features Spaces and «Distance»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



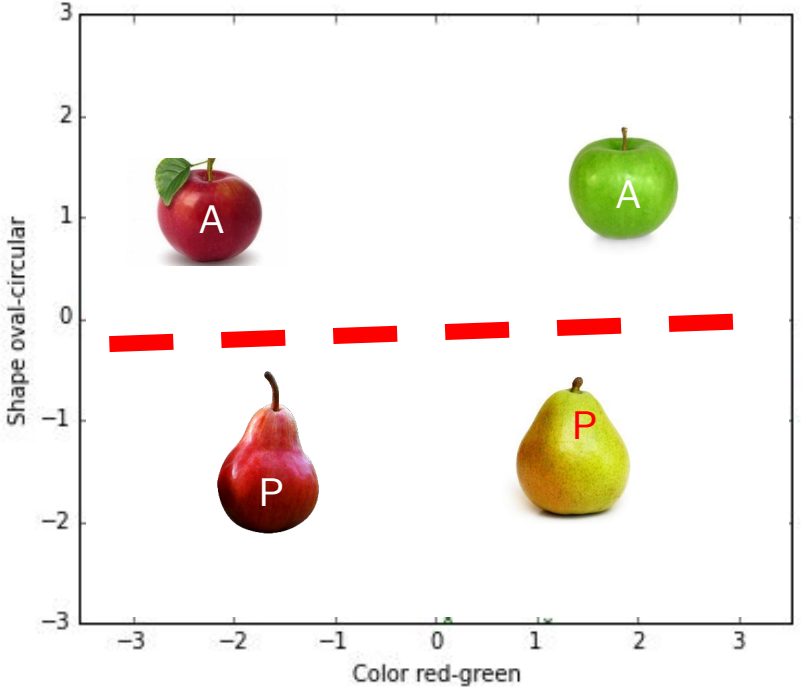
Features Spaces and «Decision»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



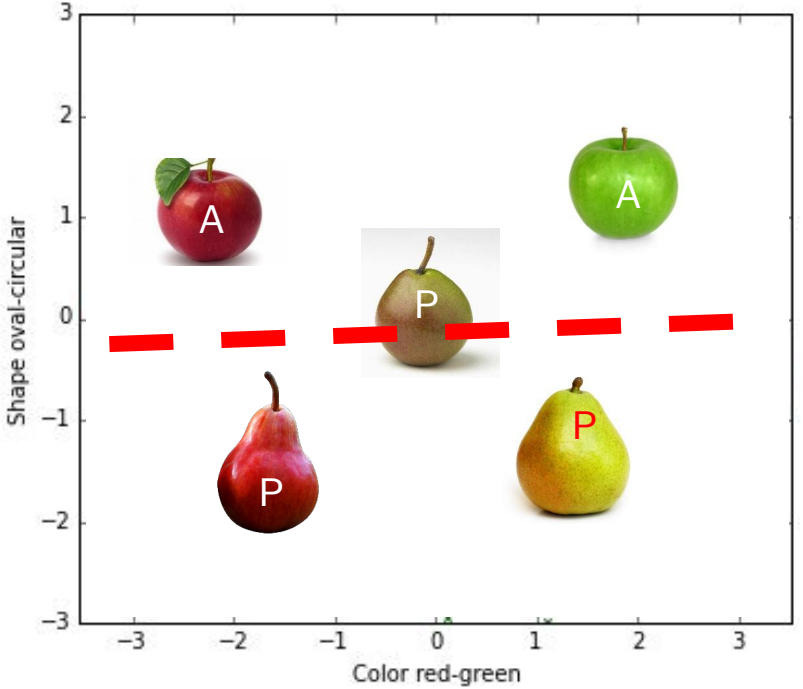
Features Spaces and «Distance»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



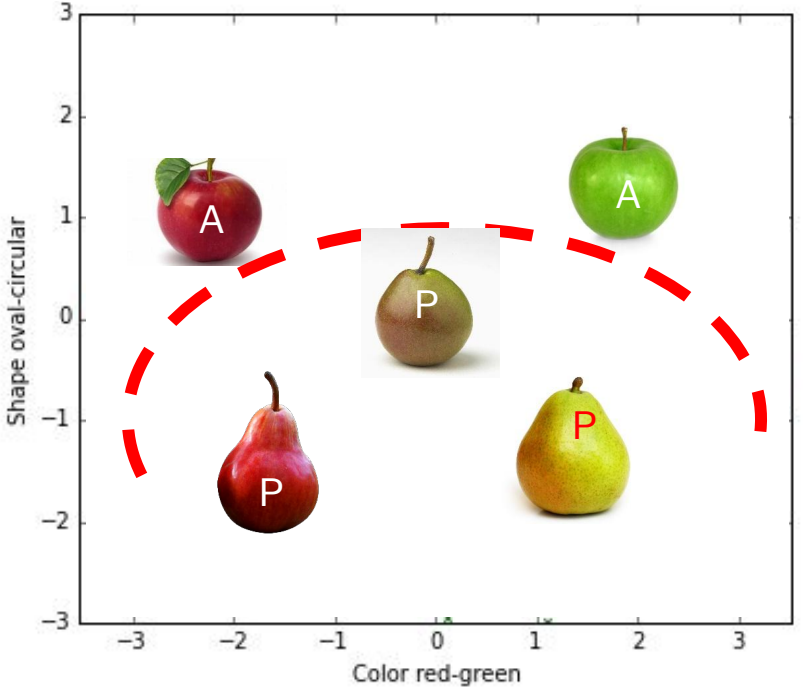
Features Spaces and «Distance»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



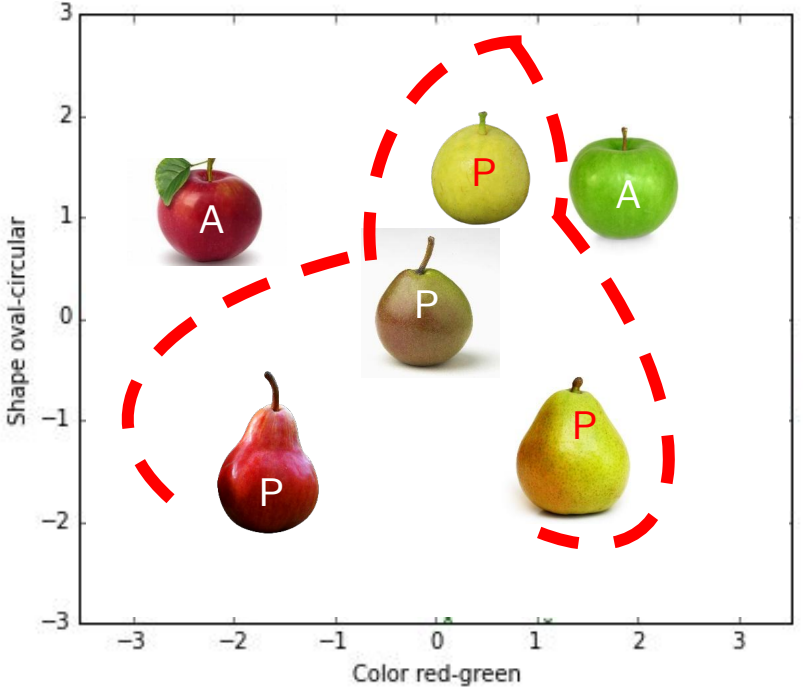
Features Spaces and «Distance»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



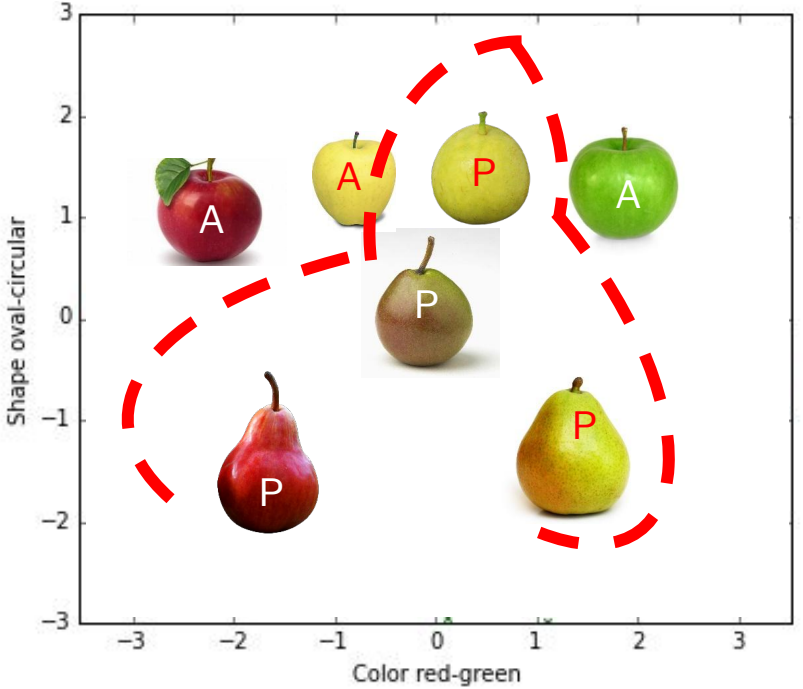
Features Spaces and «Distance»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



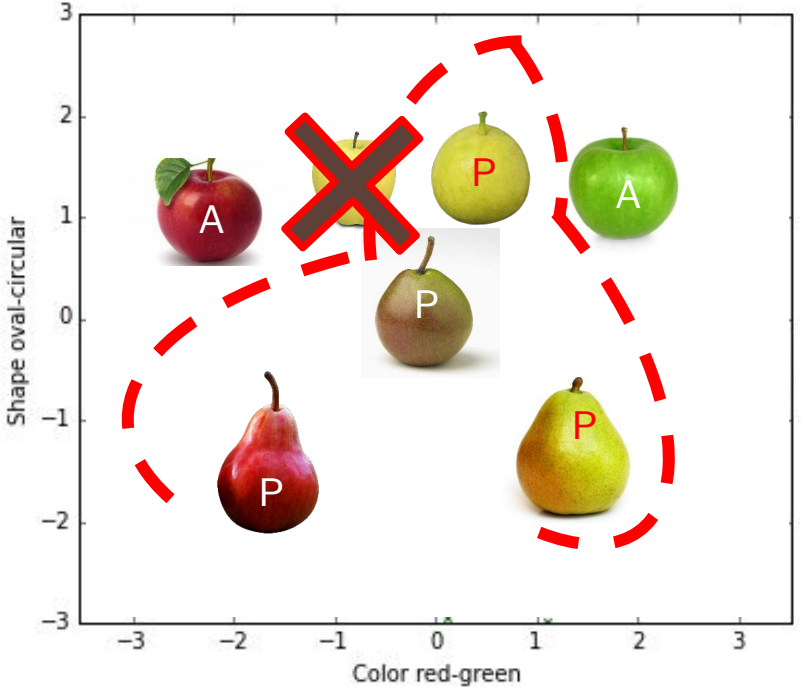
Features Spaces and «Distance»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



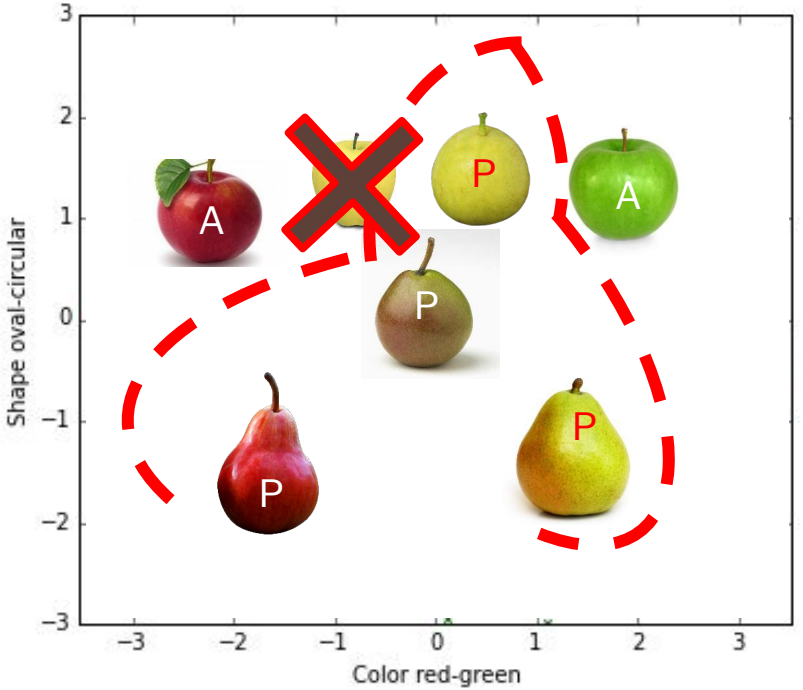
Features Spaces and «Distance»

- Color:
  - Red-orange-yellow-green:
    - a dimension from -3 to +3.
    - -3 is green
    - +3 is red
- Shape:
  - oval – spherical:
    - a dimension from -3 to +3
    - -3 is oval
    - +3 spherical



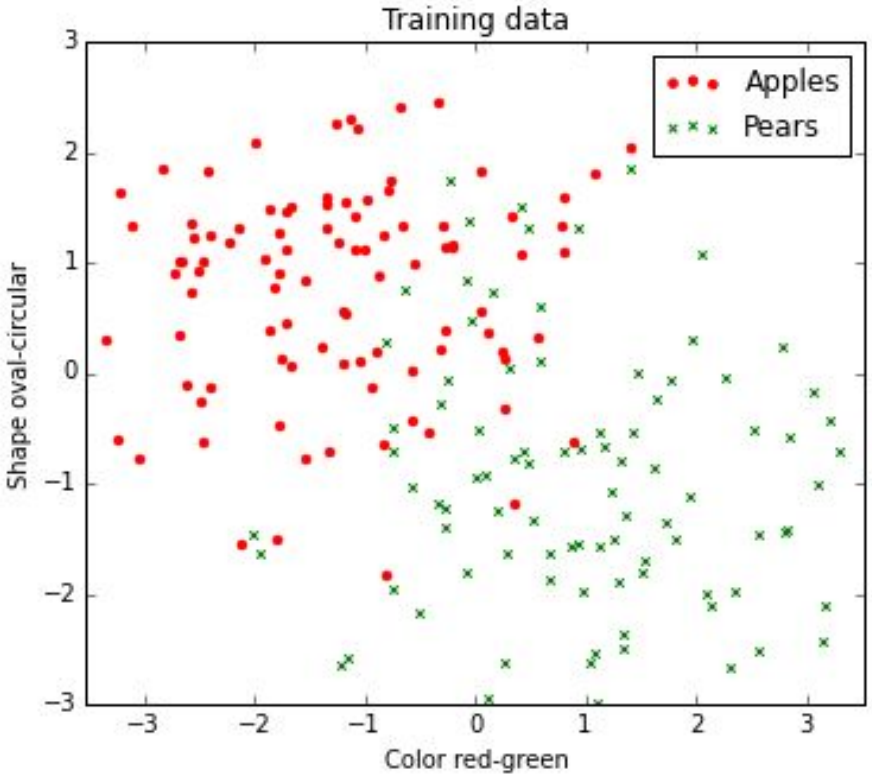
Features Spaces and «Distance»

- At some point you may conclude that the «features» you use to «decide» whether it is an apple or a pear is not enough.
  - Or you may conclude that they are! (depends on the case)
- This is called the feature space.
- A decision is a separation line/surface/hyper-surface in an N dimensional feature space.



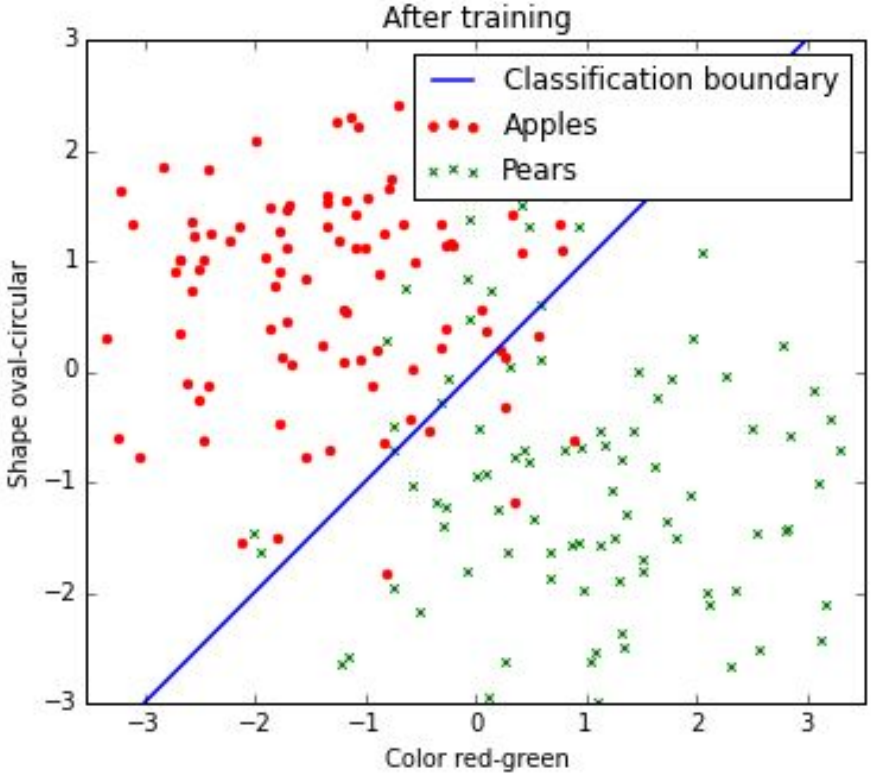
Features Spaces and «Distance»

- Let’s look at real data:
- Somebody, thanks to them, collected the color and shape values for a number of apples and pears and represented them on our 2D feature space:



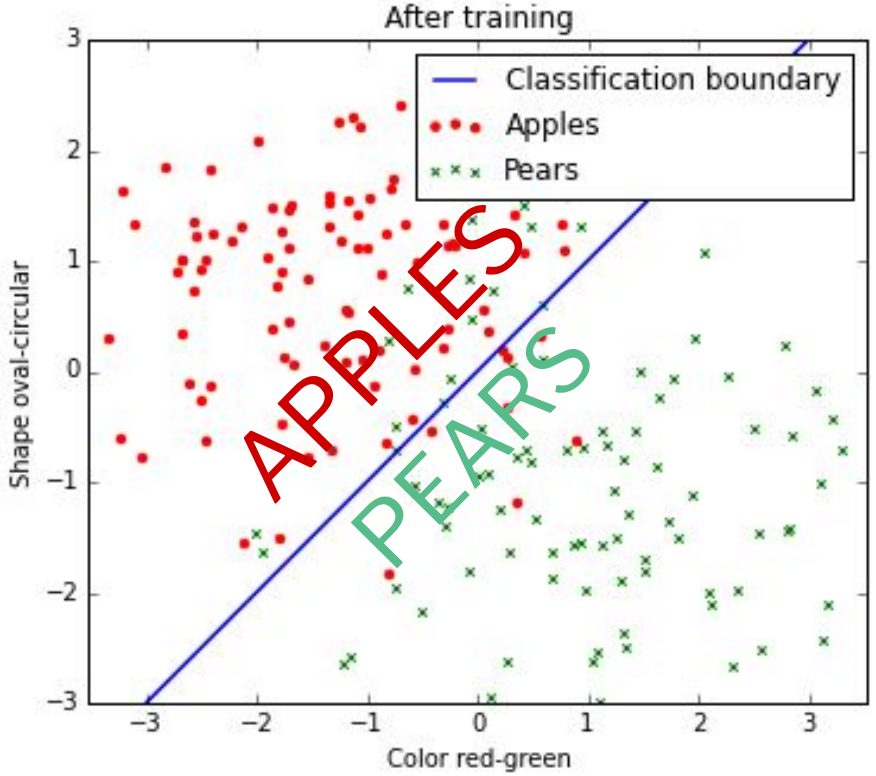
Features Spaces and «Distance»

- Let’s find a decision «line» from the data.
  - Which is x=y line. Simple.
- There are many errors?



Features Spaces and «Distance»

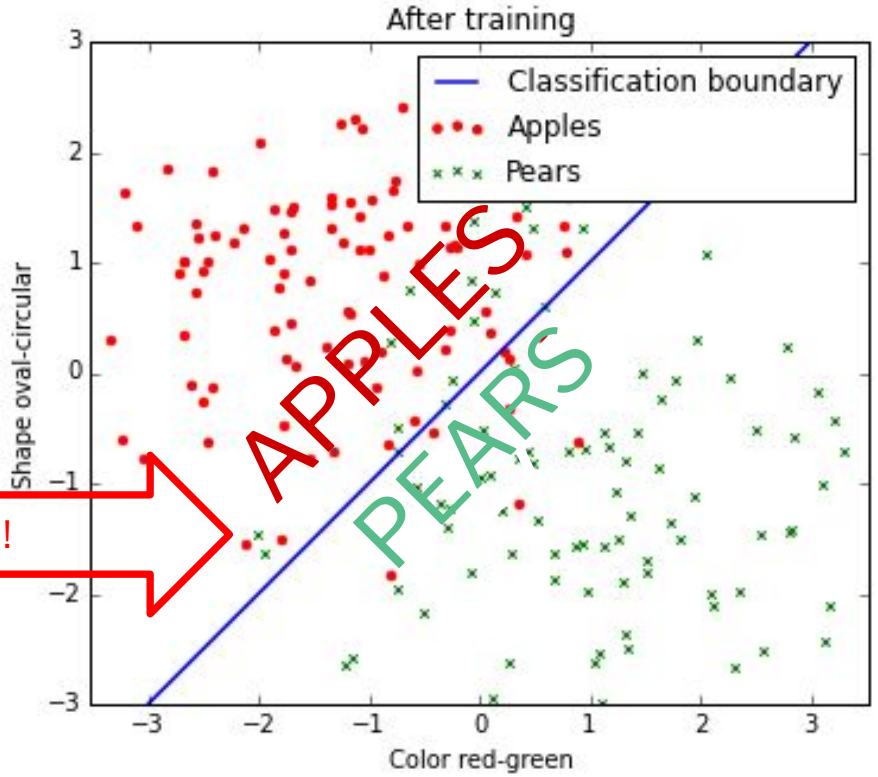
- Let’s find a decision «line» from the data.
  - Which is x=y line. Simple.
- There are many errors?



Features Spaces and «Distance»

- Let’s find a decision «line» from the data.
  - Which is x=y line. Simple.
- There are many errors?

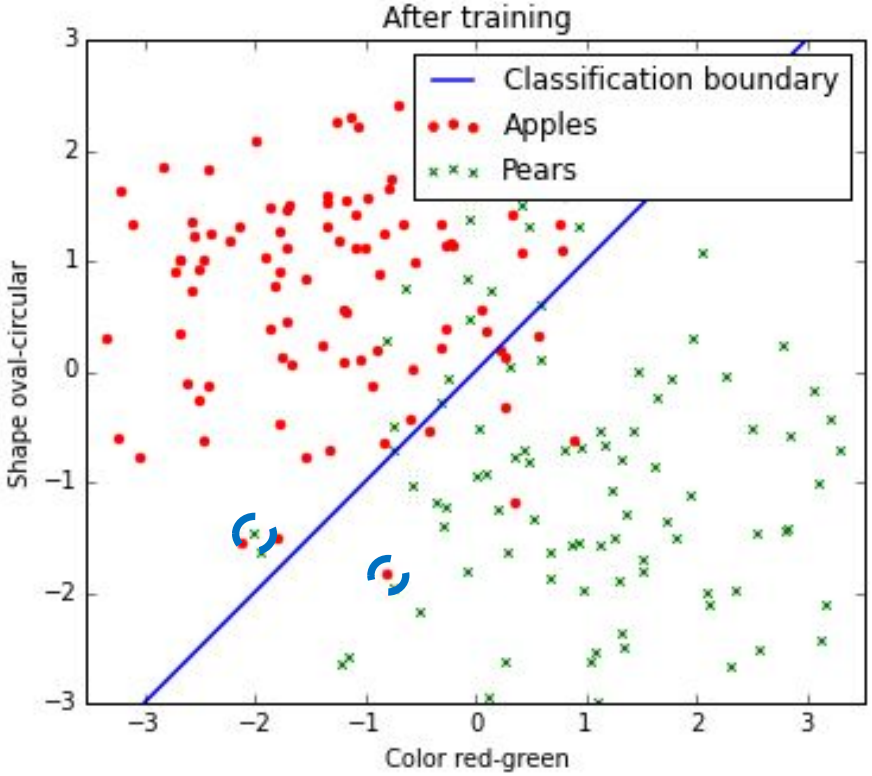
WRONG DECISION !



- This is a reddish, less oval pear!

Features Spaces and «Distance»

- Why our decisions are wrong?
  - Because our features are inadequate!
  - Because our discriminative surface is inadequate!
- But as humans we never mistake an apple to a pear, how do we do it?
  - What kind of features we use?
- We use very complex features, which are called abstract features. We as humans have a «schema» of everything in our semantic memory.
- We can define an abstract «appleness» and «pearness».



Abstract Features

- What kind of features we use?
- We can define an abstract «appleness» and «pearness» in our minds.
- Actually these semantically meaningful definitions (abstract features) is the manifestation of our sophisticated intelligence
  (such as art.)

Abstract Features

- What do you see in this image
  - A man, a woman, a girl, a horse…
  - A family ? (how!)
- Because we have an abstract understanding of ever interaction in a signal we collect.
- We convert simple signals in to complex relations and create complex features.

Abstract Features

- How to create these abstract features mathematically?
- This is called:
  - *The semantic gap!*
- The solution is:
  - “Deep Learning”

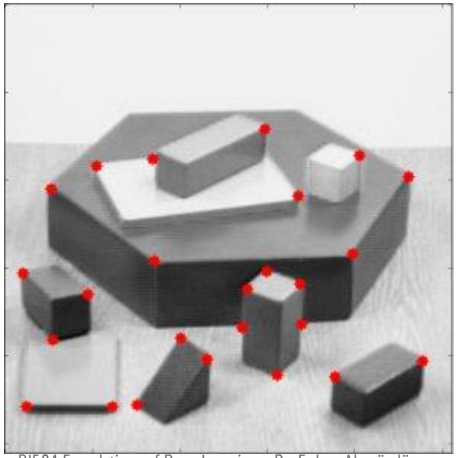
Features

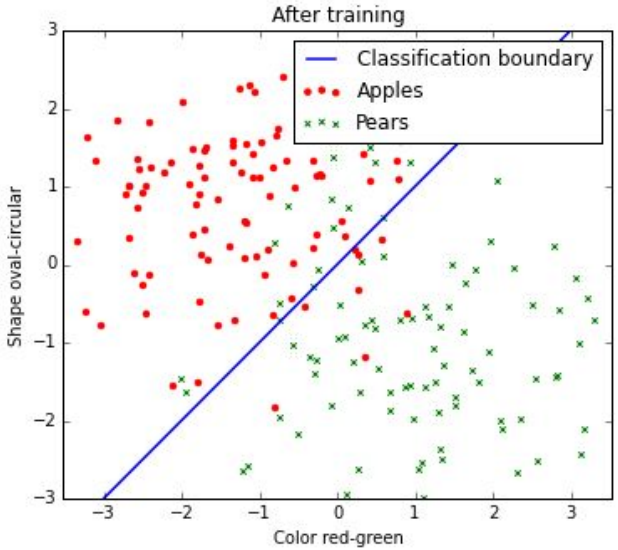
- What kinds of mathematical features can we obtain from signals?
  - Images:
    - edges, corners,
    - Histograms, gradients
    - More ?
  - Sound
    - FFT coefficients
    - Spectrograms, Cepstrum Coefficients
    - More?
- How to define More features?

Features Evolution

- Classical features were mathematically basic features, like edges, corners, blobs, etc for images.
- For other signals (sound etc) they were similarly simple (FFT coefs, etc.)

Before 2000s

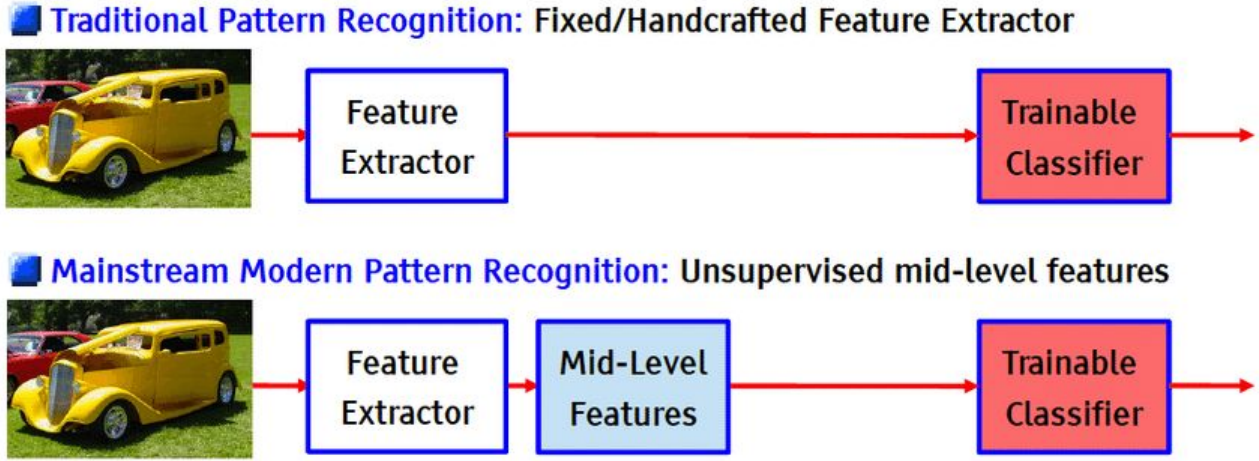




Features Evolution

- Then came the hand-crafted features ages. Scientist worked on «hand-design» of best features for an image, sound clip, a signal, etc.
- These hand-crafted features were mathematically complex but still semantically meaningless and not abstract.

Before 2000s

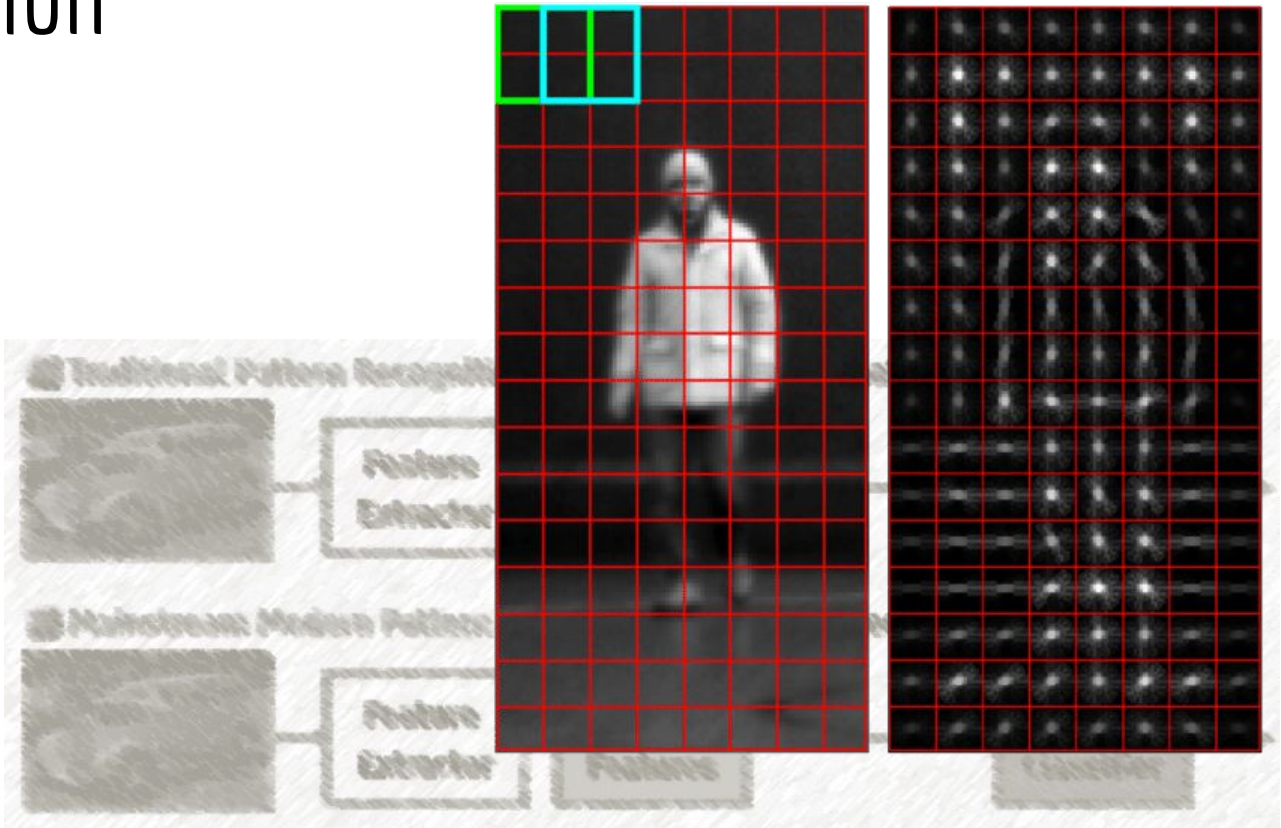


Features Evolution

- Histogram of gradients (2006) is a good example.

Before 2000s

2000-2010



Features Evolution

Before 2000s

Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor

Feature Extractor → Trainable Classifier

2000-2010

Mainstream Modern Pattern Recognition: Unsupervised mid-level features

Feature Extractor → Mid-Level Features → Trainable Classifier

After 2010

Deep Learning: Representations are hierarchical and trained

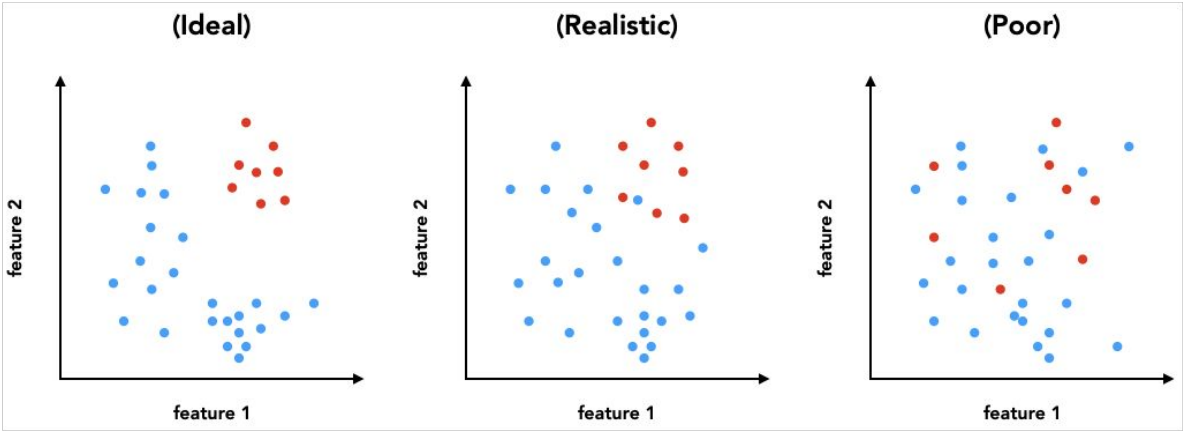
Low-Level Features → Mid-Level Features → High-Level Features → Trainable Classifier

Features Evolution

- With deep learning and convolutional neural networks, age of hand-crafted features ended.
- Now Deep Learning learns which features are necessary and uses them for that purpose.
- So all features (low level, abstract) are «learned».
- HOW ?!?!
  - That's the purpose of this course ☺.

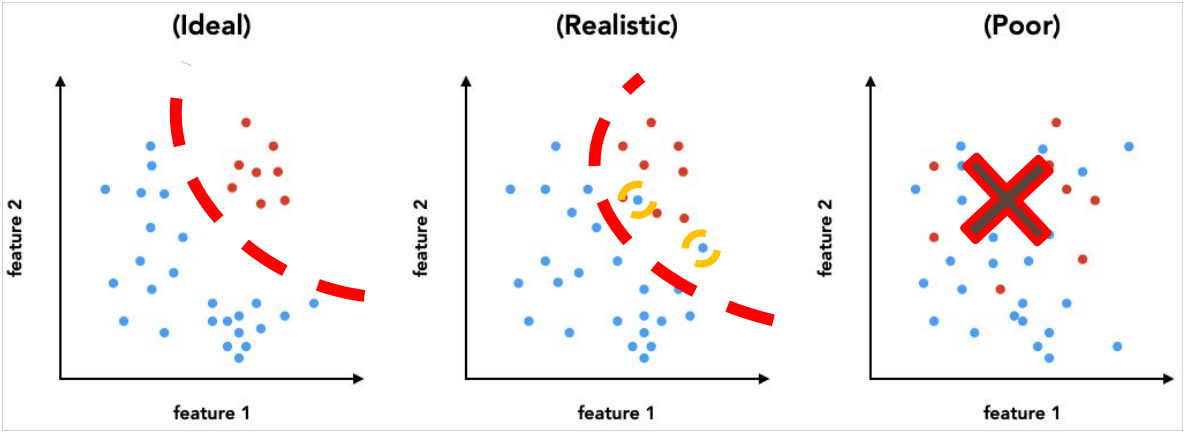
Feature Spaces and «Distance»

- So we figured out that, in order to learn a concept, we need «good features» for that problem.
- A feature may serve a concept (intelligent decision), whereas it may not another one:



Feature Spaces and «Distance»

- So we figured out that, in order to learn a concept, we need «good features» for that problem.
- A feature may serve a concept (intelligent decision), whereas it may not another one:

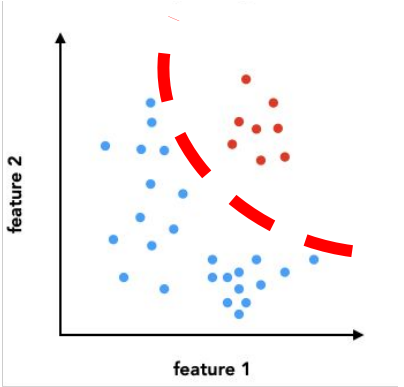


Feature Spaces and «Distance»

- To find these discriminative «hyper-surfaces» we need a definition of a «distance»
- In 2-D world, we can define distances in different ways:
  - Euclidean distance (l_2 norm)
    - $[(x_B - x_A)^T \cdot (x_B - x_A)]^{½}$
  - Manhattan distance (l_1 norm)
  - l_N norm
  - Mahalanobis Distance (covariance normalized l_2 norm)
    - $[(x_B - x_A)^T \cdot \boldsymbol{C}^{-1} \cdot (x_B - x_A)]^{½}$
  - And many other…

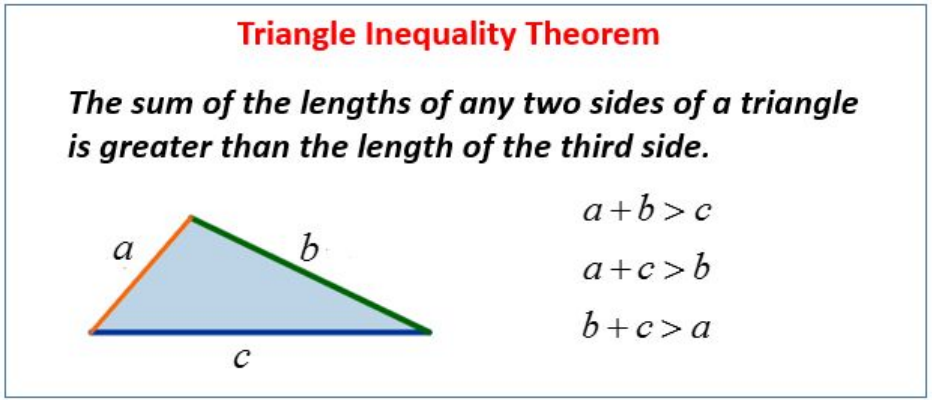
Feature Spaces and «Distance»

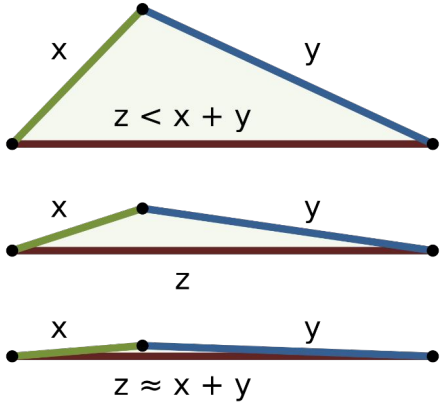
- These metric all follow the «**triangle inqeuality**»
  - Euclidean distance (l_2 norm)
    - $[(x_B - x_A)^T \cdot (x_B - x_A)]^{½}$
  - Manhattan distance (l_1 norm)
  - l_N norm
  - Mahalanobis Distance (covariance normalized l_2 norm)
    - $[(x_B - x_A)^T \cdot \boldsymbol{C}^{-1} \cdot (x_B - x_A)]^{½}$
  - And many other…



Feature Spaces and «Distance»

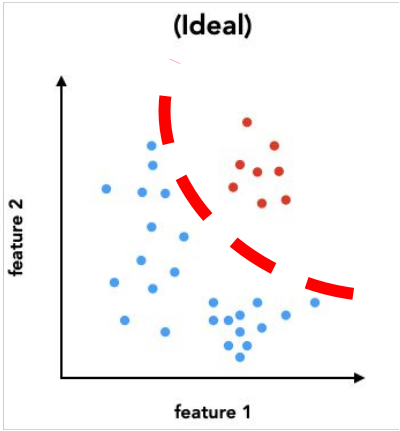
- What is «**triangle inqeuality**» ?





Feature Spaces and «Distance»

- Triangle inequality sounds trivial and natural.
- Actually this natural distance measure is how we find the discriminative surfaces



Feature Spaces and «Distance»

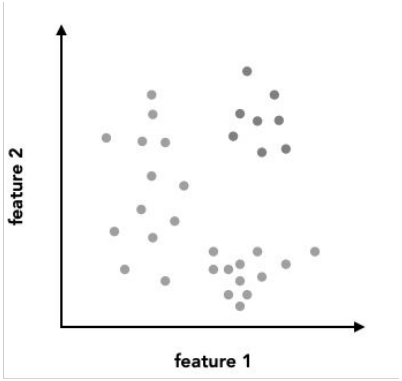
- Triangle inequality sound trivial and natural.
- Actually this natural distance measure is how we find the discriminative surfaces
  - The distance
- Does all feature spaces follow the triangle inequality?
  - Let’s think of a distance called «buddyness»
    - Buddyness: how close we are as friends:
      - Erdem is very close to Milla (0.001 buddyness distance)
      - Erdem is very close to Monica (0.0011 buddyness distance)
      - Milla and Monica are not very close (3 buddyness distance - they are jealous!)
    - No triangle inequality!

Feature Spaces and «Distance»

- Within abstract (semantic) feature spaces, intelligent systems (like AI) usually define distances that do not follow triangle inequality.
- Consequently, most of the time, the hyper-surfaces we define are not linear
  - Hyper-**planes** will not work
- The trademark of deep learning is to handle this non-linear relationship.
- Because
  - They create abstract features.
  - They create non-linear relations between these abstract features.

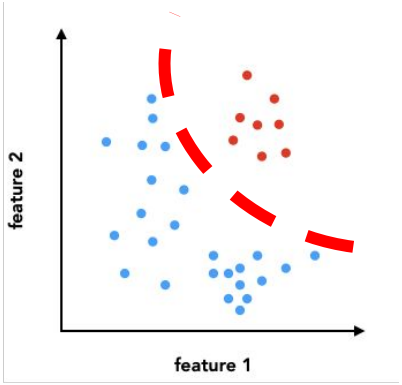
K-nearest neighbors

- So, within a features space, using a predetermined distance measure, we know which sample is closer to which.



K-nearest neighbors

- So, within a features space, using a predetermined distance measure, we know which sample is closer to which.
- In order to make a decision, you need some labels and a decision (discriminative) surface



K-nearest neighbors

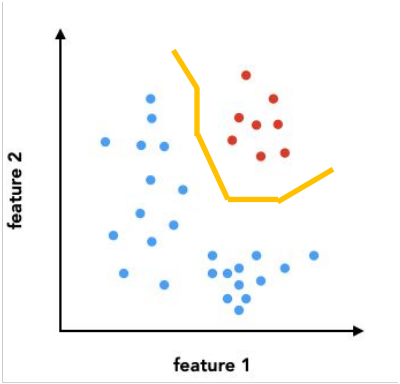
- K-nearest neighbors method is a very simple way to define a discriminative surface.
- In *k-NN classification*, the output is a class membership.
- An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors
  - *k* is a positive integer, typically small.
  - If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

K-nearest neighbors

- An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors
  - *k* is a positive integer, typically small.
  - If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

→

- L2-norm
- k=1 nearest



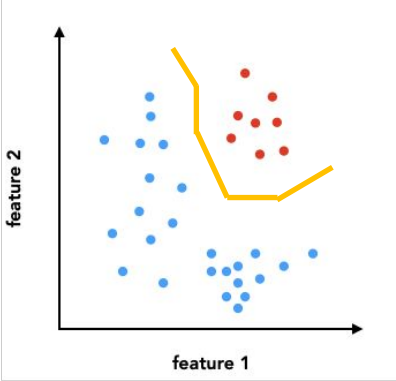
K-nearest neighbors

- An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors
  - *k* is a positive integer, typically small.
  - If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

- L2-norm
- k=1 nearest

- What is the best value of **k** to use?
- What is the best **distance** to use?

These are **hyperparameters**: choices about the algorithm that *we set rather than learn*.



How to set the hyper-parameters?

You can:

1. Choose hyperparameters that work best on the data?
   - Not a good idea, always works perfectly on training data, usually poor on new cases you haven't seen.

2. Split data into **train** and **test**, choose hyperparameters that work best on test data
   - Still not a good idea, how will algorithm perform on new data? You do not know.

How to set the hyper-parameters?

You can:

1. Split data into train, val, and test; choose hyperparameters on validation and evaluate on test sets. Decide the hyper-parameters on validation data. Finally test your success on Test data
   - Yeah, that's the way to do it!

- This is called **cross-validation**. It is a technique to assess how the results of a statistical analysis (such as a machine learning problem) will generalize to an independent data set.

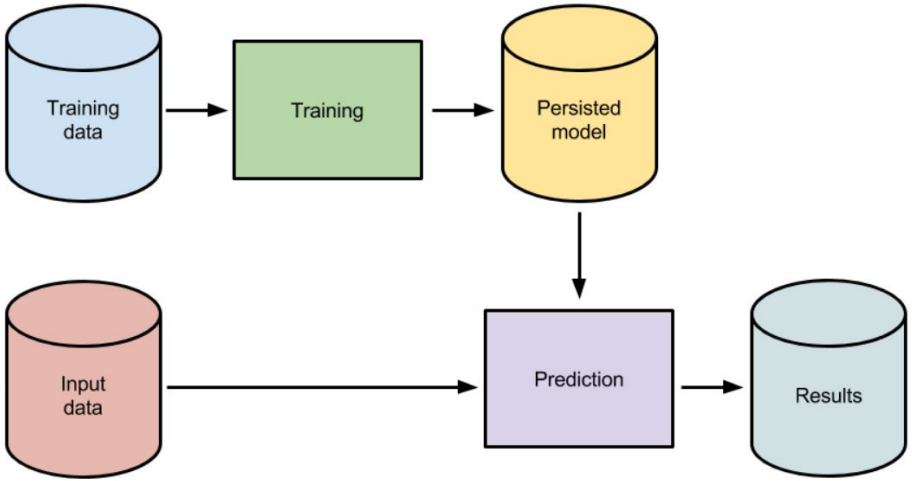
(back to) K-nearest neighbors

- K-nn is a very simple method to find a decision surface.
- Its main drawback is that basic "majority voting" classification occurs when the class distribution is skewed.
  - Examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the *k* nearest neighbors due to their large number.
  - One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its *k* nearest neighbors. (weightes k-nearest)
  - Let's look at a simple k-nn demo:

http://vision.stanford.edu/teaching/cs231n-demos/knn/ **Let's try it!**

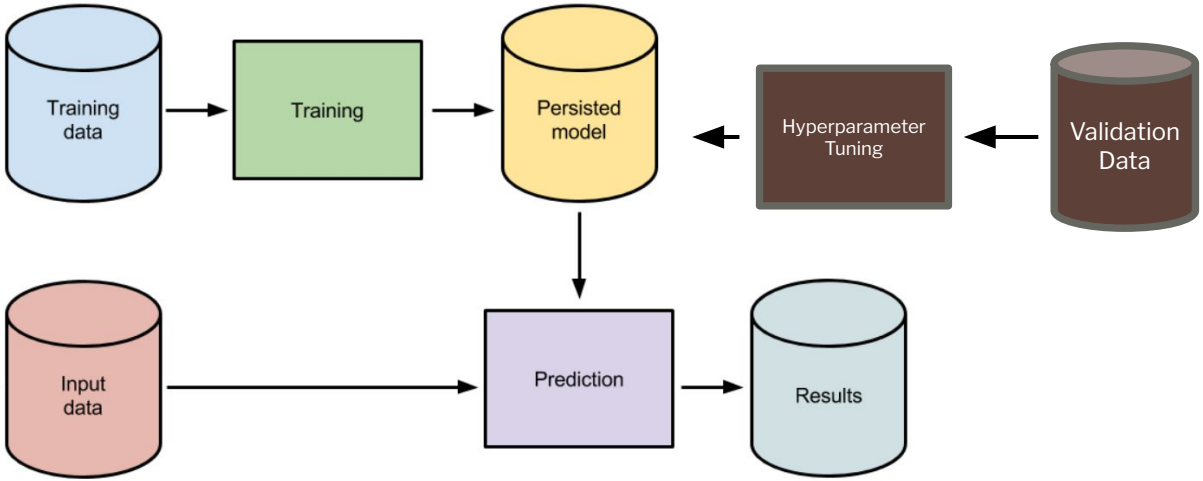
Machine Learning Pipeline

- *The simplest machine learning pipeline looks like this*



Machine Learning Pipeline

- *The simplest machine learning pipeline looks like this*



How to set the hyper-parameters?

You can:

1. Split data into train, val, and test; choose hyperparameters on validation and evaluate on test sets. Decide the hyper-parameters on validation data. Finally test your success on Test data
   - Yeah, that's the way to do it!

- This is called **cross-validation**. It is a technique to assess how the results of a statistical analysis (such as a machine learning problem) will generalize to an independent data set.

Machine Learning Pipeline Steps:

1. Define a problem
2. Collect Data
3. Split Data into Training/Validation/Test Sets
4. Create a model Using the Training Data
5. Tune the parameters of the model using the Validation Data
6. Measure your Performance (i.e. Results) using the Test Data

Machine Learning Pipeline Steps:

1. Define a problem
    - Is this a cat sound?

Machine Learning Pipeline Steps:

1. Define a problem
   - Is this a cat sound?
2. Collect Data
   - Collect sound clips that include a cat «meauuwing», and all other types of sound (dog barking, horse neighing, Serdar Ortaç singing etc.)
   - You have two groups of sounds: «cat» sounds and «other»

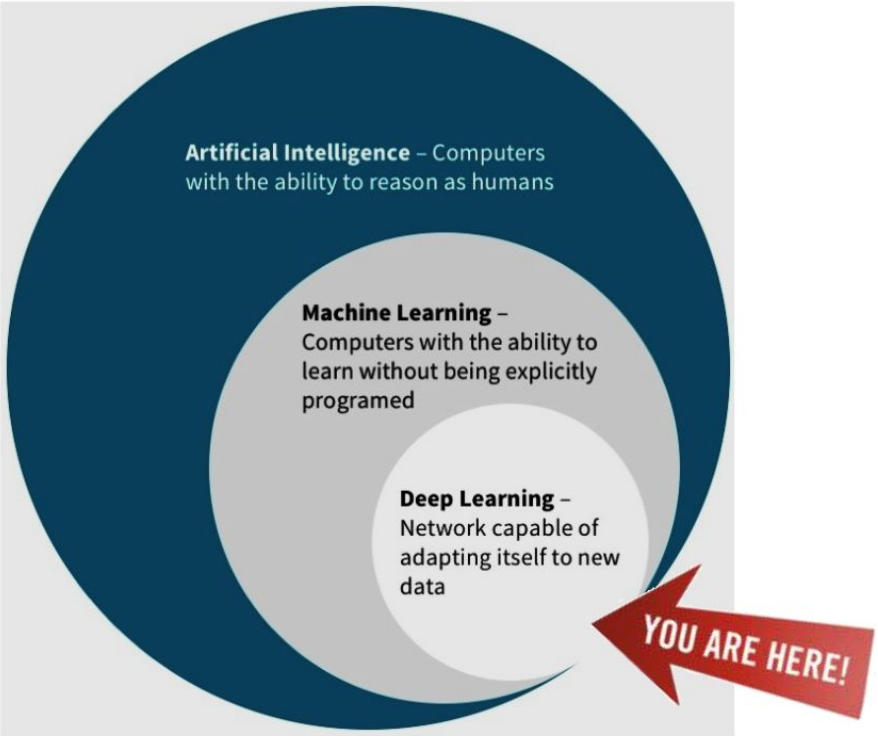
Machine Learning Pipeline Steps:

1. Define a problem
2. Collect Data
   - Collect sound clips that include a cat «meauuwing», and all other types of sound (dog barking, horse neighing, Serdar Ortaç singing etc.)
   - You have two sets of sounds cat «sounds» and «other»
3. Split Data into Training/Validation/Test Sets
   - Now split two sets (cat and other) into three groups.
   - Determining the size of these Training/Validation/Test Sets is a delicate business. We will talk about it later.

Machine Learning Pipeline Steps:

3. Split Data into Training/Validation/Test Sets
   - Now split two sets (cat and other) into three groups.
   - Determining the size of these Training/Validation/Test Sets is a delicate business. We will talk about it later.
4. Create a model Using the Training Data
   - Now you delve into real machine learning business. Use a machine learning method to create a «model» training data.
   - In this course our models will be deep learning-based.
   - There are many other types of machine learning models DLs are just some of them.

The Big Picture



DL models:

- *image classification*
- *image segmentation*
- *speech-to-text*
- *Natural language processing*
- *Sound segmentation*
- *...*

Machine Learning Pipeline Steps:

4. Create a model Using the Training Data
   - Now you delve into real machine learning business. Use a machine learning method to create a «model» Training data.
   - In this course our model will be Convolutional Neural Networks, we will learn how to do it.
   - There are many other types of machine learning models CNNs are just one of them
5. Tune the parameters of the model using the Validation Data
   - Your model will have some specific parameters. To find them use the Validation set
6. Measure your Performance (i.e. Results) using the Test Data

Machine Learning Pipeline Steps:

4. Create a model Using the Training Data
5. Tune the parameters of the model using the Validation Data
   - Your model will have some specific parameters. To find them use the Validation set
6. Measure your Performance (i.e. Results) using the Test Data
   - If you are satisfied with your parameters (the previous step), get your results using your Test Data.
   - e.g. Results:

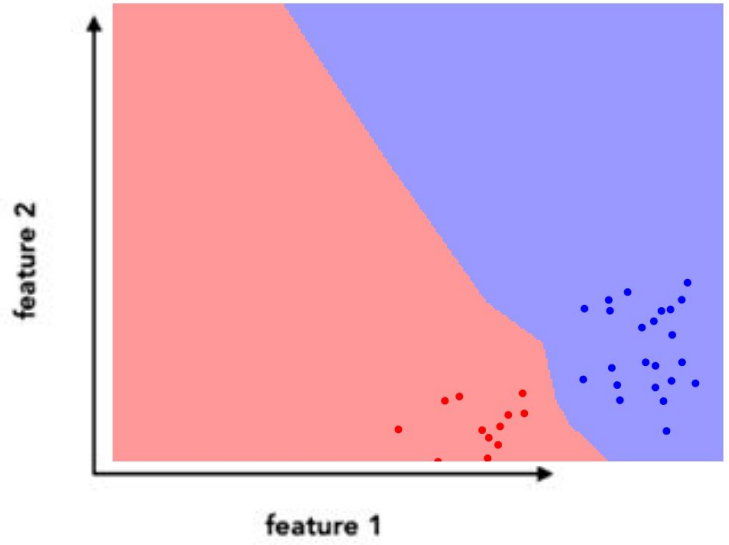
Machine Learning Pipeline Steps:

4. Create a model Using the Training Data
5. Tune the parameters of the model using the Validation Data
   - Your model will have some specific parameters. To find them use the Validation set
6. Measure your Performance (i.e. Results) using the Test Data
   - If you are satisfied with your parameters (the previous step), get your results using your Test Data.
   - e.g. Results:
     - Cat sound detection with 87% accuracy
     - 8% false alarms with Serdar Ortaç clips.

Linear Classifier

- We will now create our first mathematil model, the linear classifier.
- Actually last week we had a classifiers, the k-nearest-neighbor algorithm.

  - K-nn worked by comparing a given test sample to all samples in the training set.
  - This method has two main drawbacks:
    - *The classifier must remember all of the training data and store it for future comparisons with the test data. This is space inefficient because datasets may easily be gigabytes in size.*
    - *Classifying a test image is expensive since it requires a comparison to all training images.*



Linear Classifier

- What we want is a mathematical model, in which all the training information is efficiently embedded within.
  - E.g. You all can detect faces. But you do not remember all the faces you have seen in your entire live.
  - You must have a way to code them efficiently.
- This is also what all DL models do.
- But also a “Linear Classifier (LC)” can do this simply. Let’s see how:

Linear Classifier

- LC will have two major components:
  - a **score function** that maps the raw data to class scores,
  - and a **loss function** that quantifies the agreement between the predicted scores and the ground truth labels.
- We will then cast this as an optimization problem in which we will minimize the loss function with respect to the parameters of the score function.

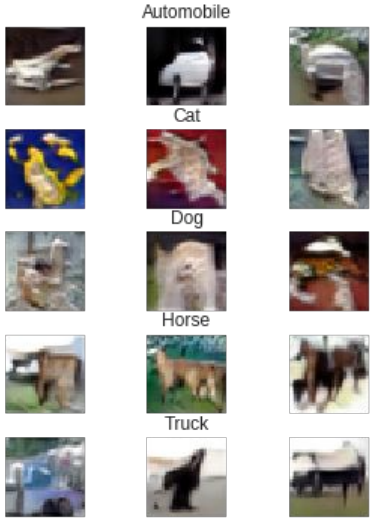
Linear Classifier - The score function

- Let's define the problem first
  - We have a signal of D dimensions:
  - We have N training samples:
  - Each sample x_i is associated with a label y_i
  - There are K distinct categories

$$x_i \in R^D$$
$$i = 1 \dots N$$

$$y_i \in 1 \dots K$$

Linear Classifier – The score function



- Let's define the problem first
  - We have a signal of D dimensions: $x_i \in R^D$
  - We have N training samples:
  - Each sample x_i is associated with a label y_i: $y_i \in 1 \ldots K$
  - There are K distinct categories

- For example, in CIFAR-10 image set, there is a training set of **N** = 50,000 images, each with **D** = 32 x 32 x 3 = 3072 pixels (dimensions), and **K** = 10, since there are 10 distinct classes (dog, cat, car, etc).
  - Each pixel is a dimension ?!?

Linear Classifier - The score function

- Our purpose is to define function $f : R^D \mapsto R^K$ that maps the raw image pixels to class scores.
  - f is called the score function.
- **Linear classifier** is one of the the simplest possible functions, a linear mapping, between the input signal and the output label:

$$f(x_i, W, b) = Wx_i + b$$

- x_i here is a column vector. If your signal was a 32x32x3 color image, then you would have its pixels flattened out to a single column vector of shape [D x 1], where d = 32x32x3 = 3072.

Linear Classifier - The score function

- The parameters in W are often called the **weights**, and b is called the **bias** vector.
- Let's write the Equation on the board in detail
  - single matrix multiplication $W \cdot x_i$ is effectively evaluating K separate classifiers in parallel (one for each class), where each classifier is a row of W.
  - Our goal will be to set these in such way that the computed scores match the ground truth labels across the whole training set

$$f(x_i, W, b) = Wx_i + b$$

Linear Classifier - The score function

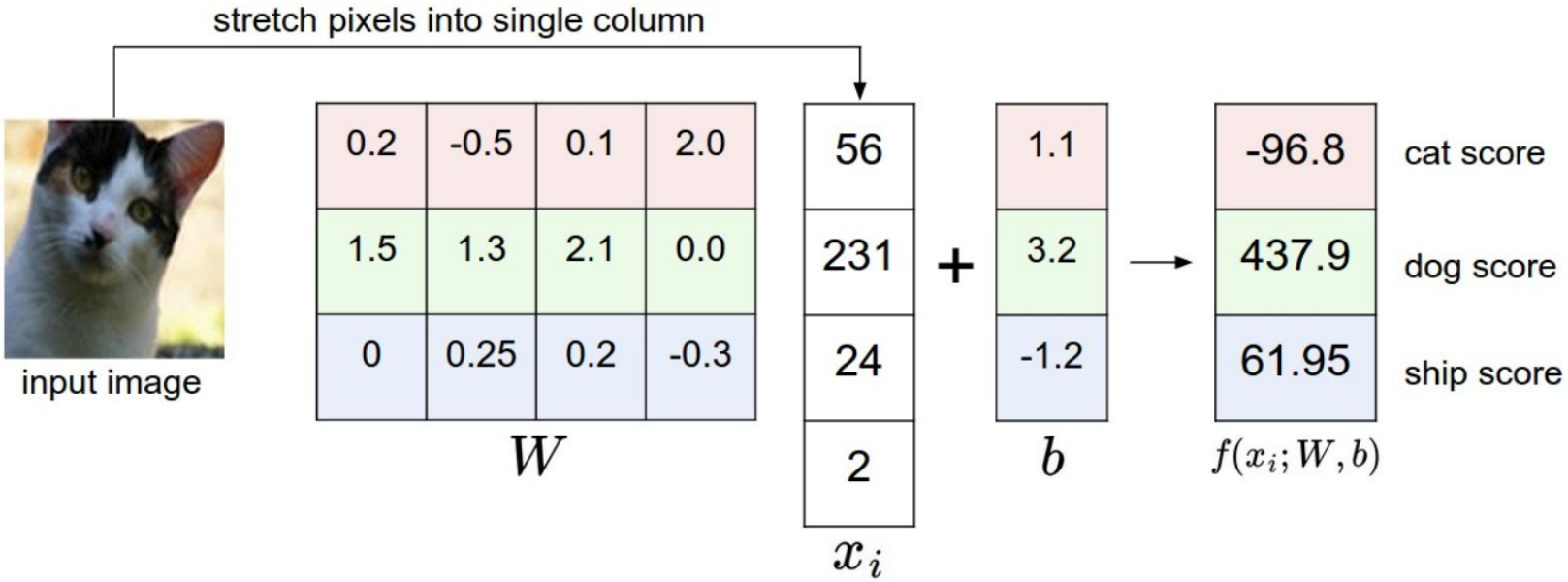
$$f(x_i, W, b) = Wx_i + b$$

$$f(x, W, b) = \begin{bmatrix} w_{1,1} & \cdots & w_{1,N} \\ \vdots & \ddots & \vdots \\ w_{D,1} & \cdots & w_{D,N} \end{bmatrix}_{KxN} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{Nx1} + \begin{bmatrix} b_1 \\ \vdots \\ b_D \end{bmatrix}_{Dx1}$$

single matrix multiplication $W \cdot x_i$ is effectively evaluating K separate classifiers in parallel (one for each class), where each classifier is a row of W

Linear Classifier - CIFAR-10 example

- How it works on a real example:



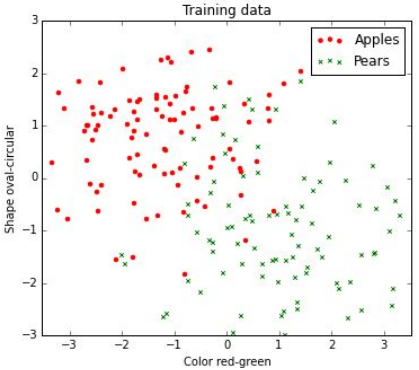
Linear Classifier

- Or a simpler example is previous week's apple/peer challenge!
- D = 2, there are two dimensions, shape and colour
- K = 2, there are two labels, apples and pears

$$f(x, W, b) = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp} \\ x_{col} \end{bmatrix}_{2x1} + \begin{bmatrix} x_{app} \\ x_{per} \end{bmatrix}_{2x1}$$

Linear Classifier

- Or a simpler example is previous week's apple/peer challenge!
- D = 2, there are two dimensions, shape and colour
- K = 2, there are two labels, apple and peer



$$f(x, W, b) = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp} \\ x_{col} \end{bmatrix}_{2x1} + \begin{bmatrix} b_{app} \\ b_{per} \end{bmatrix}_{2x1}$$

- And we have many (S number of) samples

$$f(x, W, b, s) = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp,1} & \cdots & x_{shp,S} \\ x_{col,1} & \cdots & x_{col,S} \end{bmatrix}_{2xS} + \begin{bmatrix} b_{app} & \cdots & b_{app} \\ b_{col} & \cdots & b_{col} \end{bmatrix}_{2xS}$$

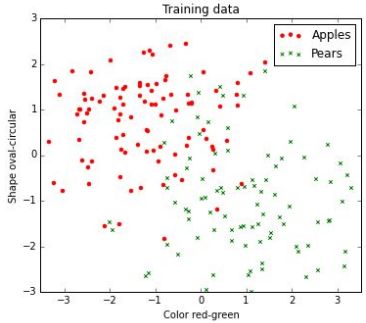
Linear Classifier

- And we have many (S number of) samples

$$f(x, W, b, s) = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \end{bmatrix}_{2xS} + \begin{bmatrix} b_{app} & \dots & b_{app} \\ b_{col} & \dots & b_{col} \end{bmatrix}_{2xS}$$

- We label the equation for a training set:

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \end{bmatrix}_{2xS} + \begin{bmatrix} b_{app} & \dots & b_{app} \\ b_{col} & \dots & b_{col} \end{bmatrix}_{2xS}$$



Linear Classifier

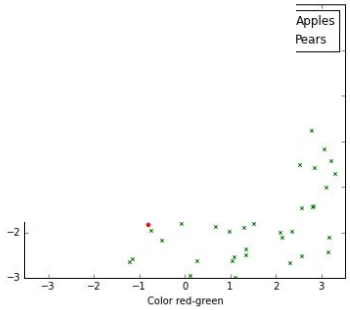
- And we have many (S number of) samples

$$f(x, W, b, s) = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \end{bmatrix}_{2xS} + \begin{bmatrix} b_{app} & \dots & b_{app} \\ b_{col} & \dots & b_{col} \end{bmatrix}_{2xS}$$

- We label the equation for a training set:

Appleness measure

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \end{bmatrix}_{2xS} + \begin{bmatrix} b_{app} & \dots & b_{app} \\ b_{col} & \dots & b_{col} \end{bmatrix}_{2xS}$$



Linear Classifier

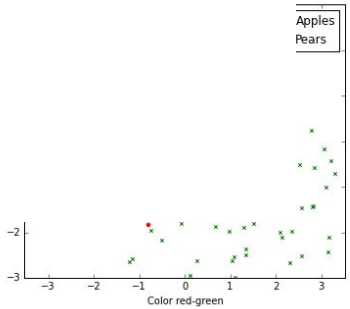
- And we have many (S number of) samples

$$f(x, W, b, s) = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \end{bmatrix}_{2xS} + \begin{bmatrix} b_{app} & \dots & b_{app} \\ b_{col} & \dots & b_{col} \end{bmatrix}_{2xS}$$

- We label the equation for a training set:

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} \\ w_{per,shp} & w_{per,col} \end{bmatrix}_{2x2} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \end{bmatrix}_{2xS} + \begin{bmatrix} b_{app} & \dots & b_{app} \\ b_{col} & \dots & b_{col} \end{bmatrix}_{2xS}$$

Pearness measure



Linear Classifier

- Then we combine W with bias

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}_{3xS}$$

This is collected data

Linear Classifier

- Then we combine W with bias

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}_{3xS}$$

There are the labels you manually designate

These are collected data

Linear Classifier

- What is what?

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}_{3xS}$$

There are the labels you manually designate

You want to find these parameters!

These are collected data

Linear Classifier

- How to solve this:

- Psuedo-inversing?

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}_{3xS}$$

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}_{3xS}$$

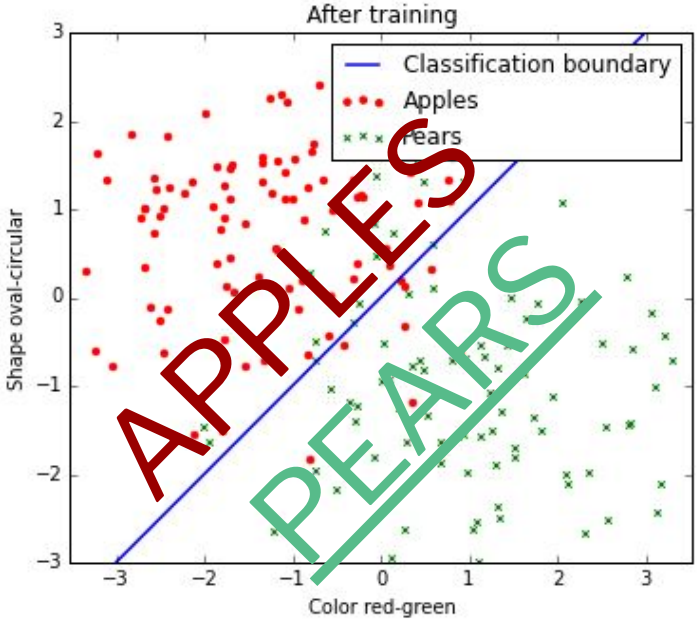
$$\mathbf{A^{+} = (A^{T} \cdot A)^{-1} \cdot A^{T}}$$

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}^{+}_{3xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3}$$

Linear Classifier

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}^{+}_{3xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3}$$

- By pseudo-inversing you will obtain a linear surface that discriminates (if it can) apples and pears.
- Psuedo-inversing is actually a type of optimization (least square optimization)
- There are other types of optimization techniques.
- Some will work better for other types of model (non-linear)



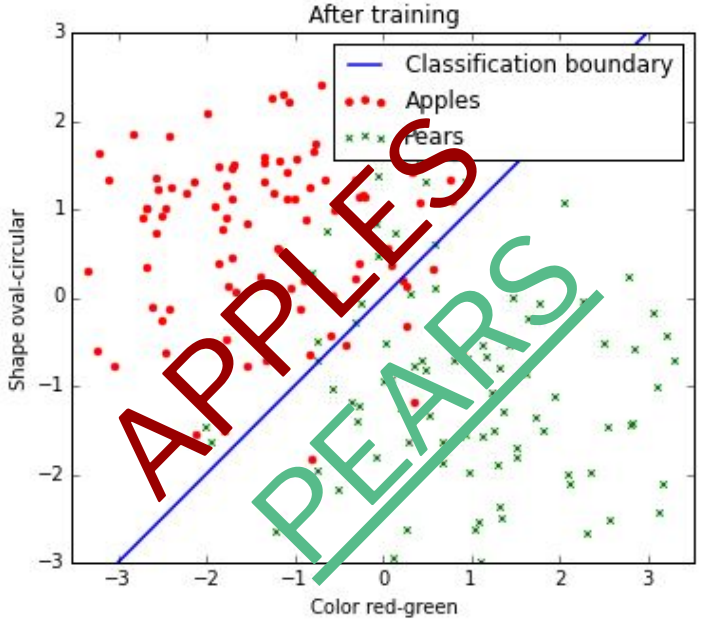
Linear Classifier vs. K-nn

- The training data is used to learn the parameters W and b. Once the learning is complete we can discard the entire training set and only keep the learned parameters.
  - This was not the case for K-nn. K-nn had to check all samples in the dataset.
- A new test image can be simply forwarded through the LC score function and classified based on the computed scores.
- Linear Classifier involves a single matrix multiplication and addition, which is significantly faster than comparing a test image to all training images.
- LC is a machine learning model!
- In this course, we will generalize this idea to nonlinear classifiers using DL models.

Linear Classifier

$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}^{+}_{3xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3}$$

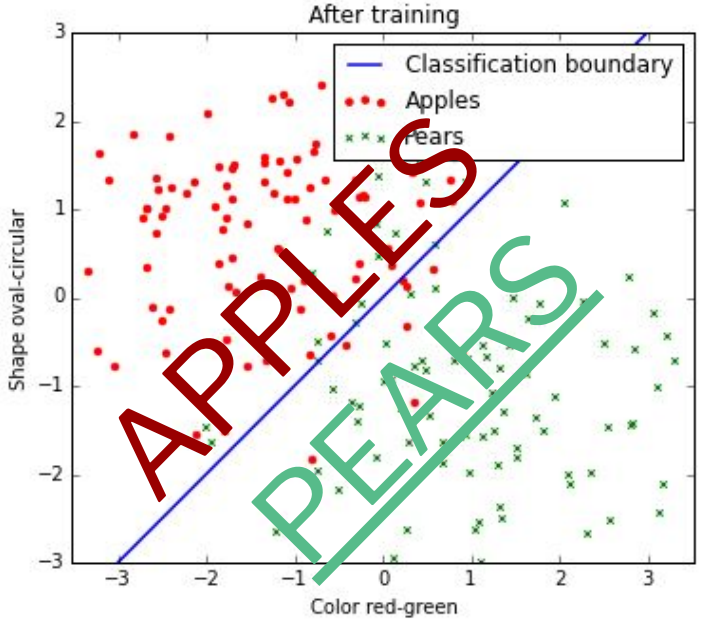
- Trick Question?
  - There was something wrong with this figure. What was it?



Linear Classifier

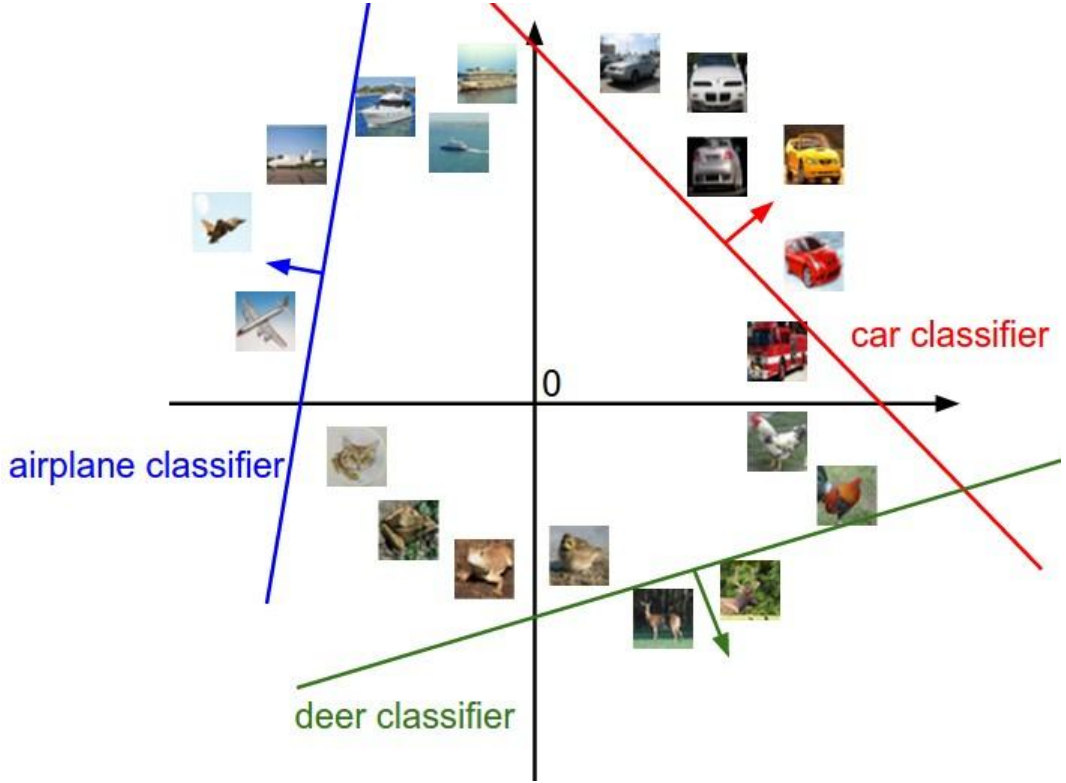
$$\begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \end{bmatrix}_{2xS} \cdot \begin{bmatrix} x_{shp,1} & \dots & x_{shp,S} \\ x_{col,1} & \dots & x_{col,S} \\ 1 & \dots & 1 \end{bmatrix}^{+}_{3xS} = \begin{bmatrix} w_{app,shp} & w_{app,col} & b_{app} \\ w_{per,shp} & w_{per,col} & b_{app} \end{bmatrix}_{2x3}$$

- Trick Question?
  - There was something wrong with this figure. What was it?
  - There is only one «line». However the equation should have given us two!



Linear Classifier - CIFAR-10

- We cannot visualize 3072-dimensional spaces, but if we imagine squashing all those dimensions into only two dimensions, then we can try to visualize what the classifier might be doing:

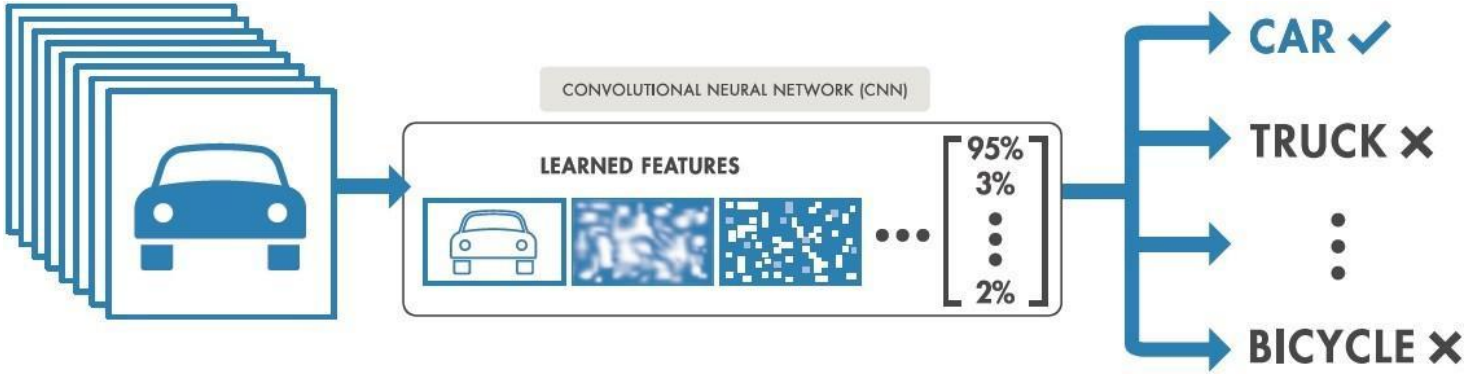


How to evaluate a classifier?

- Evaluating a decision system is a title of statistics.
- There are various methods.
- We will make a brief summary
- http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/31PattRecog/13ClassifierPerformance.pdf
  - Highly recommended.

Classification

- K-nn algorithm was a very simple way to **classify** an n-class feature space.
- Most deep learning architectures are used to classify signals (images, sounds etc)



Other Deep Learning Problems

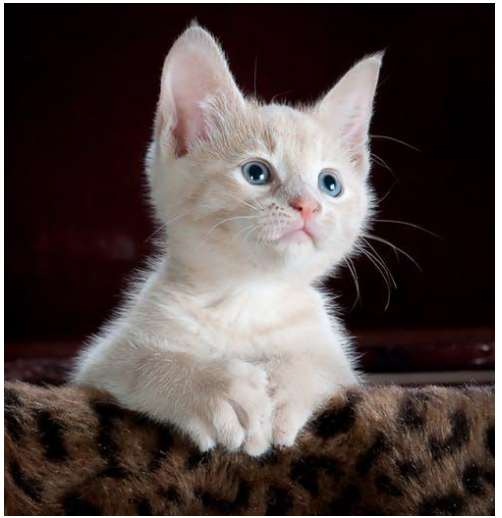
- **Detection**: Designate the existence of a «type of signal» (i.e. Object) in a given signal.
- **Classification/Recognition**: Given a signal with an object, find out what that object is. In other words, classify it in a class from a set of predefined categories.
- **Localization** : Find where the region-of-interest is, within a signal.
- **Segmentation**: Classify every signal sample (pixel, sound clip) in a signal to a class according to its context.
- **Regression**: Reconstruct a function with a given an input and an output signal.
- **Generation**: Create an original signal, given a dataset of similar signals.

Detection

- Find the existence of an object/event in a signal (image, sound, etc.)

Is this a cat image?

- **YES** (or NO)

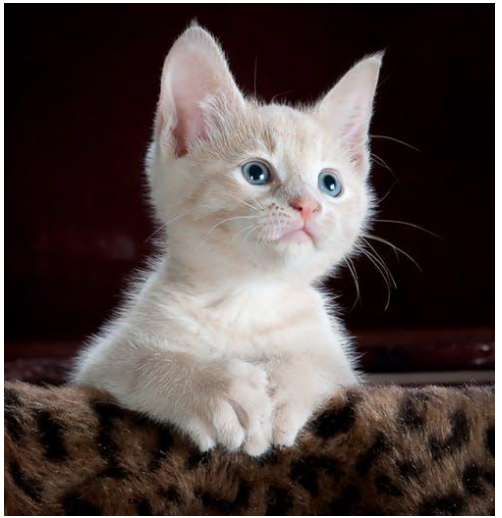


Detection as Classification

- Given a signal with an object, find out what that object is. In other words, classify it in a class from a set of predefined categories

What kind of an image is this?

- It is **CAT** image

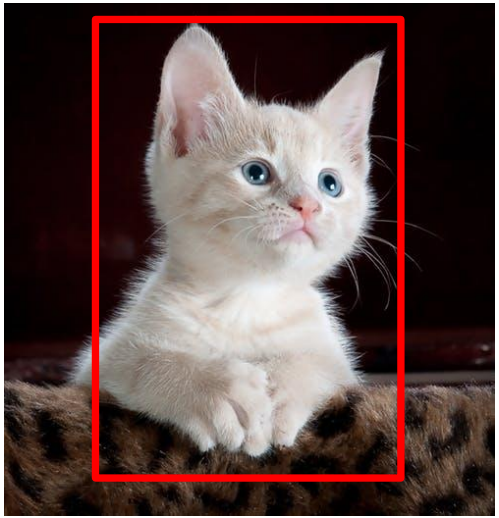


Localization

- Find where the region-of-interest is, within a signal

Where is the cat?

- In that **box**



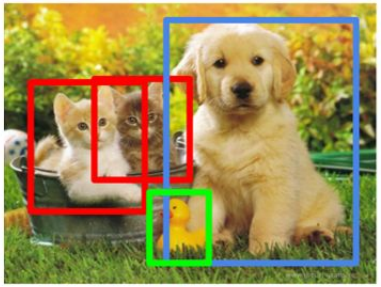
Detection+Classification+Localization

- These three problems are combined under the title «object detection».

Classification

CAT

Object Detection



CAT, DOG, DUCK

Segmentation

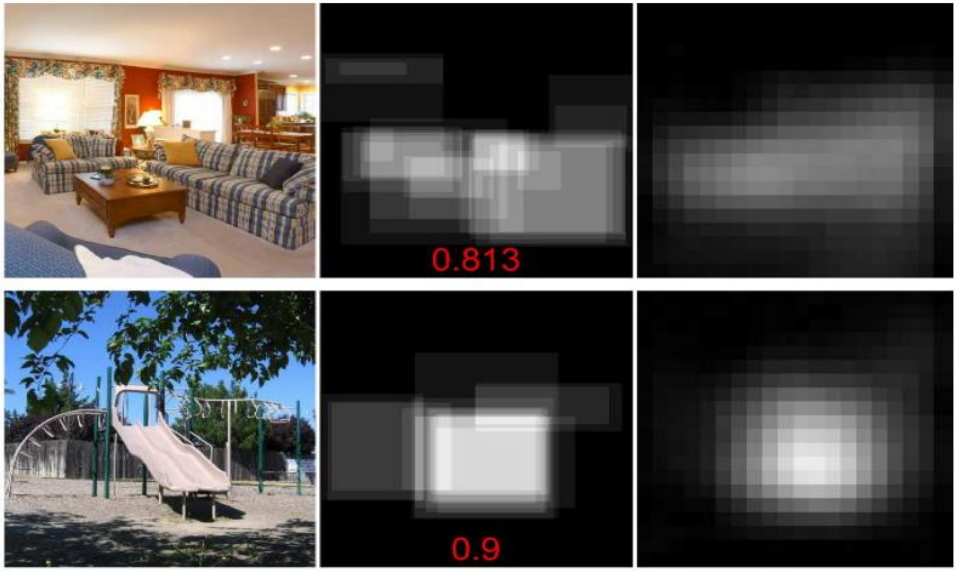
- Classify every signal sample (pixel, sound clip) in a signal to a class according to its context.

Exactly where is the cat?

- On those pixels only □

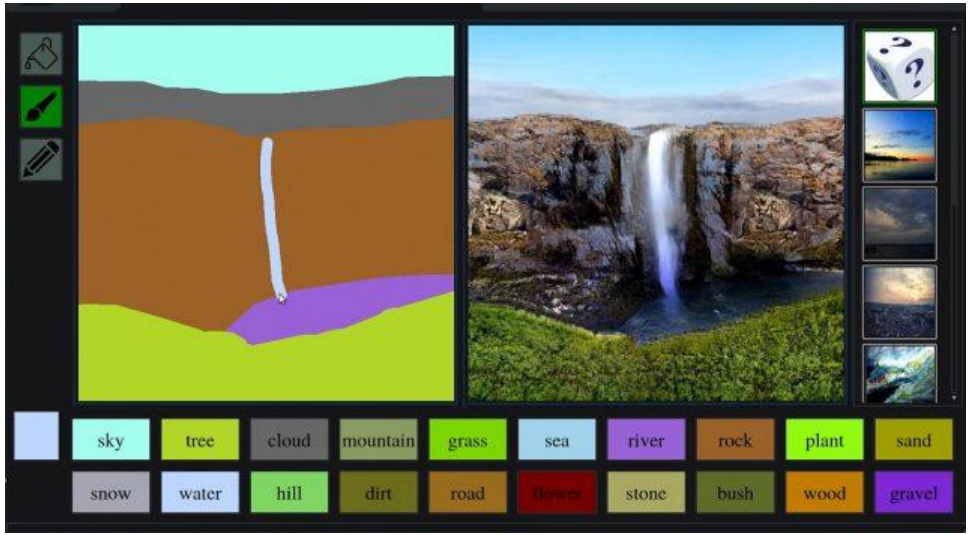
Regression

- Reconstruct a function with a given an input and an output signal



Generation

- Create an original signal, given a dataset of similar signals.



Generation

- Create an original signal, given a dataset of similar signals.
  - Or if you really need cat images:

What to do until next week?

- Choose your programming environment and set it up!
  Last Warning!
- Think of a project idea. Preferably related to your thesis.
  - You don’t know how to do it yet, even if it can be done or not. Do some research. Did people do it before? What could they achieve? Maybe you will implement a modification on the problem? Start thinking about it.