7.1 Discriminant functions

In the previous chapter, we explored how to extract a feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ to describe an image object or *region*. In this chapter, our goal is **to classify the object as belonging to one of M object classes or categories**, according with said feature vector $\mathbf{x}$. In other words, we pursuit the design of a mapping between features and categories $\mathbf{x}^n \to C$ where $C \in \{C_1, \ldots, C_M\}$ (see Fig 1.).

Scenario: A computer vision system in charge of classifying the objects within kitchens. The considered categories are: $C_1 = \text{fridge}$, $C_2 = \text{bowl}$, $C_3 = \text{microwave}$, $C_4 = \text{oven}$, $C_5 = \text{toaster}$, $C_6 = \text{spoon}$.

Results of object classification:

$$x^1 = [x_1^1, x_2^1, \ldots, x_n^1] \to C = C_4$$
$$x^2 = [x_1^2, x_2^2, \ldots, x_n^2] \to C = C_1$$
$$x^3 = [x_1^3, x_2^3, \ldots, x_n^3] \to C = C_3$$
$$x^4 = [x_1^4, x_2^4, \ldots, x_n^4] \to C = C_5$$

Fig 1. Example of object classification within the COCO dataset.

For that, the space defined by the features in $\mathbf{x}$, also referred as **feature space**, is divided into prediction regions. The image below shows an example of those regions in a scenario with categories $C = \{C_1, C_2, C_3\}$, so $M = 3$, and features $\mathbf{x} = [x_1, x_2]$:
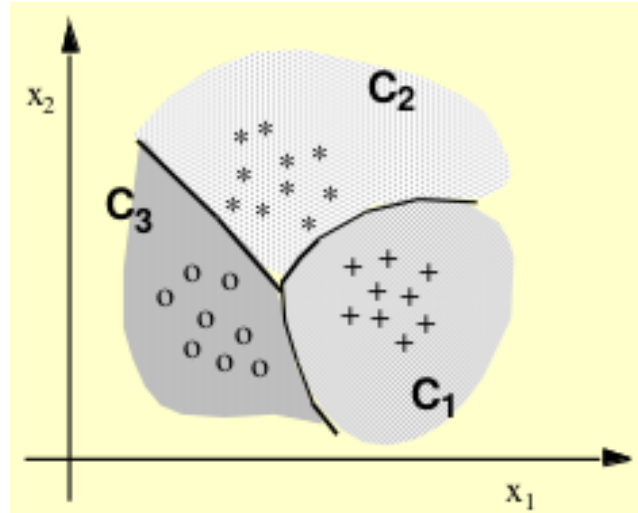

Fig 2. Example of a feature space with three regions.

Two main steps are needed in order to build an object recognition system and work with it:

- A **training (design) phase**, where sample vectors of known objects are used to learn the classifier (supervised learning).
- A **prediction (online) phase**, where the image objects are classified as belonging to one of the classes based on the learned prediction models.

Such prediction models come in two flavors:

- **Statistical classifiers:**
  - It is assumed that the feature vectors $\mathbf{x}$ of the classes C follow a **statistical distribution**.
  - The parameters of such distribution need to be **learned from known objects**.
  - Examples: Naïve Bayessian Classifier, Logistic Regression, Conditional Random Fields.
- **Non-statistical classifiers:**
  - No assumption is made on the statistical distribution of the feature vector.
  - The coefficient of deterministic discriminant functions are learned.
  - Examples: Support Vector Machine, Perceptron, AdaBoost.

This notebook focuses on non-statistical classifiers. Concretely it covers:

- Linear discriminant functions (section 7.1.1).
- The concept of separability (section 7.1.2).

- Generalized discriminant functions ([section 7.1.3](#)).

Problem context - Traffic sign recognition

Before self-driving vehicles can truly operate autonomously, they would need to identify traffic signs like "speed limit", "school ahead" or "turn ahead". This problem is called **Traffic Sign Recognition (TSR)** and is faced by companies like Tesla or Google (Waymo), and now by *AliquindoiCars*, a startup located in the Andalusia Technology Park in Málaga.

TSR is part of the features collectively called *Advanced Driver Assistance Systems (ADAS)*. It uses image processing techniques to detect the traffic signs. The detection methods used can be generally divided into color based, shape based, or those using low level features.

Given your growing experience in computer vision, *AliquindoiCars* contacted you asking for a TSR technique to be integrated into a self-driving car.

```python
In [28]:  import numpy as np
          import cv2
          import matplotlib.pyplot as plt
          import matplotlib
          matplotlib.rcParams['figure.figsize'] = (8.0, 8.0)

          images_path = './images/'
```

The data

Since most traffic signs in Spain (and in the rest of the world) have a rectangular, circular or triangular shape, they provided you 3 folders containing 20 images each;

- `./images/circles/{0-19}.png`
- `./images/squares/{0-19}.png`
- `./images/triangles/{0-19}.png`

These folders contain segmented and binarized images of traffic signs captured by a camera in a car. **Our task is to design a recognition system able to distinguish each type of shape**.

Let's take a look at some of these images:

In [29]:
```python
# Load images
img_circle = cv2.imread(images_path + "circles/0.png", 0)
img_square = cv2.imread(images_path + "squares/0.png", 0)
img_triangle = cv2.imread(images_path + "triangles/0.png", 0)

# Show them!
plt.subplot(131)
plt.imshow(img_circle,cmap='gray');
plt.title('Circular shape')

plt.subplot(132)
plt.imshow(img_square,cmap='gray');
plt.title('Rectangular shape')

plt.subplot(133)
plt.imshow(img_triangle,cmap='gray');
plt.title('Triangular shape');
```
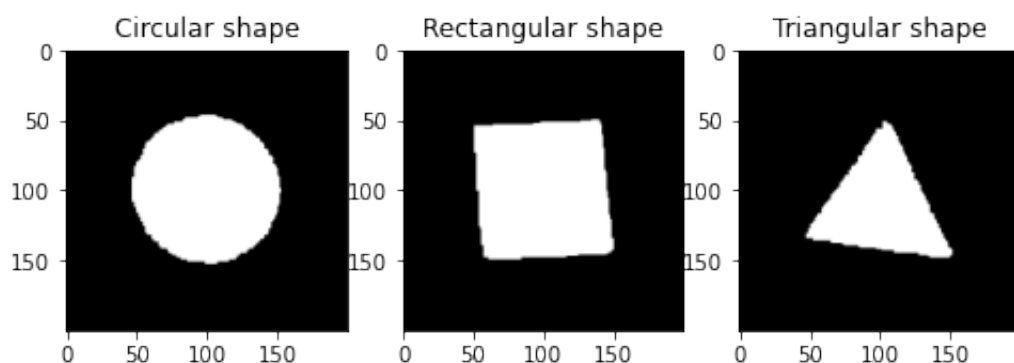
Computing feature vectors

Prior to describe the methods that will be used to perform recognition, let's build the feature vectors they are going to use. For this, and given your experience, you are going to use the Hu Moments, a 7-size feature vector that looks like
$\mathbf{x} = [v_1, v_2, v_3, v_4, v_5, v_6, v_7]$.

ASSIGNMENT 1: Computing the features

Your first task is to obtain a $20x7$ matrix for each type of shape, where rows index images and columns Hu moments. In this way, the 5th row in the matrix characterizing rectangular signs would contain the Hu moments computed from the 5th image with that shape.

In order to get a first-hand impression of how these features are distributed in the feature space for the different shapes, plot the results using `plt.scatter()` (https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.scatter.html) (show only the firsts 2 Hu moments).

```python
In [30]:  # Assignment 1
          def image_moments(region):
              """ Compute moments of the external contour in a binary image.

                  Args:
                      region: Binary image

                  Returns:
                      moments: dictionary containing all moments of the regio
              """

              # Get external contour
              contours,_ = cv2.findContours(region,cv2.RETR_EXTERNAL ,cv2.CHA
              cnt = contours[0]

              # Compute moments
              moments = cv2.moments(cnt)

              return moments
```

```python
In [31]:  # Compute Hu moments for circle shape
          hu_circles = np.zeros((20,7))

          for i in range(20):
              img = cv2.imread(images_path + "circles/" + str(i) + ".png", 0)
              moments = image_moments(img)
              hu_circles[i,:] = cv2.HuMoments(moments).flatten()

          # Compute Hu moments for square shape
```

```python
hu_squares = np.zeros((20,7))

for i in range(20):
    img = cv2.imread(images_path + "squares/" + str(i) + ".png", 0)
    moments = image_moments(img)
    hu_squares[i,:] = cv2.HuMoments(moments).flatten()

# Compute Hu moments hor triangle shape
hu_triangles = np.zeros((20,7))

for i in range(20):
    img = cv2.imread(images_path + "triangles/" + str(i) + ".png", 0
    moments = image_moments(img)
    hu_triangles[i,:] = cv2.HuMoments(moments).flatten()

# Define plot axis
plt.axis([0.15, 0.2, -0.0005, 0.0015])

# Plot firsts two Hu moments
plt.xlabel("First Hu moment")
plt.ylabel("Second Hu moment")
plt.scatter(hu_triangles[:,0],hu_triangles[:,1],marker="^")
plt.scatter(hu_circles[:,0],hu_circles[:,1], marker="o")
plt.scatter(hu_squares[:,0],hu_squares[:,1], marker="s");
```
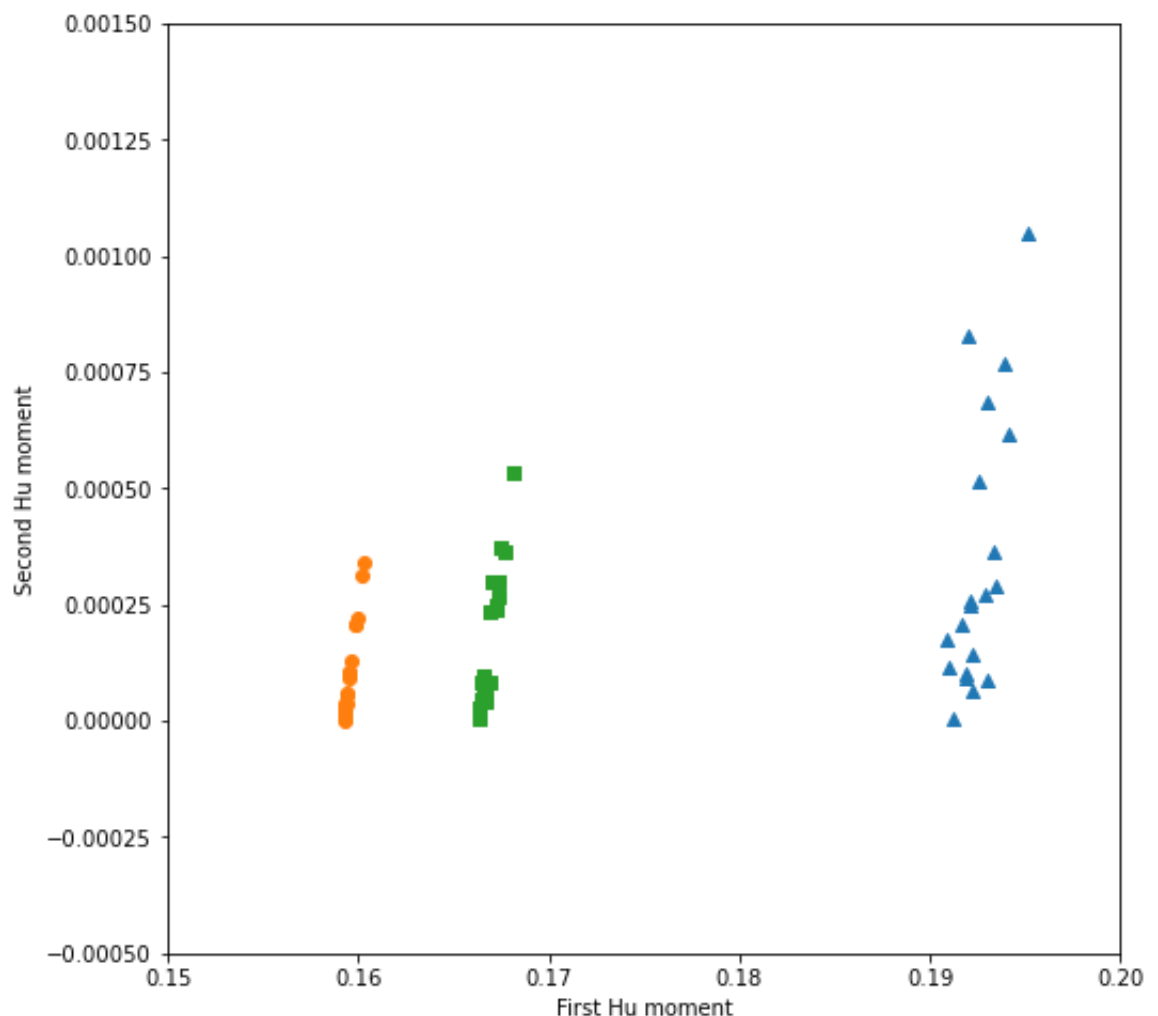
Interlude

You must save the computed Hu moments of the segmented data, since we are going to need them in the next notebook. For saving numPy matrices, you can use `np.save()` [(https://docs.scipy.org/doc/numpy/reference/generated/numpy.save.html)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.save.html) (data is saved at `./data/` by default).

In [34]:
```python
np.save("./data/hu_circles.npy",hu_circles)
np.save("./data/hu_triangles.npy",hu_triangles)
np.save("./data/hu_squares.npy",hu_squares)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/var/folders/yb/867_w5x91wn8hf0qx58cxt_c0000gn/T/ipykernel_6896/3180410788.py in <module>
----> 1 np.save("./data/hu_circles.npy",hu_circles)
      2 np.save("./data/hu_triangles.npy",hu_triangles)
      3 np.save("./data/hu_squares.npy",hu_squares)

<__array_function__ internals> in save(*args, **kwargs)

~/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/npyio.py in save(file, arr, allow_pickle, fix_imports)
    523         if not file.endswith('.npy'):
    524             file = file + '.npy'
--> 525         file_ctx = open(file, "wb")
    526
    527     with file_ctx as fid:

FileNotFoundError: [Errno 2] No such file or directory: './data/hu_circles.npy'
```

7.1.1 Linear discriminant functions

Taking a look at the scatter plot resultant from the previous assignment, we can see how the three types of signs could be segmented using 2 lines. For example:
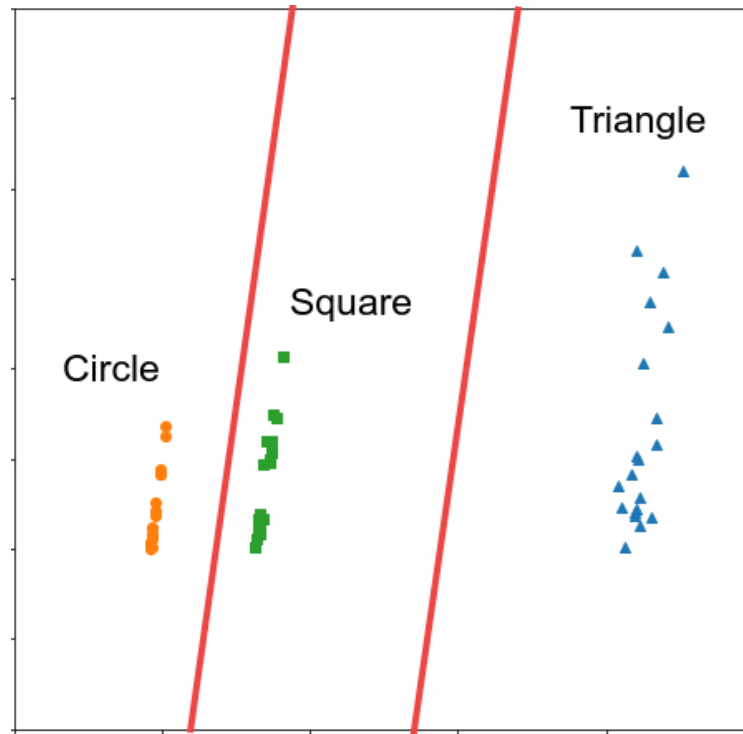
Fig 3. First two moments of the binarized traffic signs and a possible division into regions.

And that's what **linear discriminant functions** do!

Let's start by describing how such lines could be represented. As we know, a way to define a 2D line is:

$$0 = Ax + By + C$$

Using another notation:

$$0 = w_1 x_1 + w_2 x_2 + w_3$$

Note that we can generalize this 2D line to any dimension hyperplane:

$$0 = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_{n+1} 1 = \sum_{i=1}^{n+1} w_i x_i = w^T \cdot x$$

In this way, we can divide the feature space for any dimension (e.g. n = 7 for Hu moments).

Working with two classes

If we are considering just two classes $C = \{C_1, C_2\}$ (e.g. an object in a kitchen could be a spoon or a fork), we can define a linear discriminant function as:

$$d(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} \quad \text{(dot product of both vectors)}$$

Formally, the vector w is called **weight vector** and we will see a way to compute it automatically from a set of data called **training data**.

In this way, when we have a new object characterized by $\mathbf{x}'$ to classify, $d(\mathbf{x}')$ is computed and:

- If $d(\mathbf{x}') \geq 0$ the new object is assigned C_1, and
- if $d(\mathbf{x}') < 0$ the object is assigned to C_2.

We say that the linear discriminant function $d(\mathbf{x}')$ works as a **decision boundary** between both classes. The following figure illustrates this (blue dots represent characterized objects belonging to C_1, while red ones those belonging to C_2).
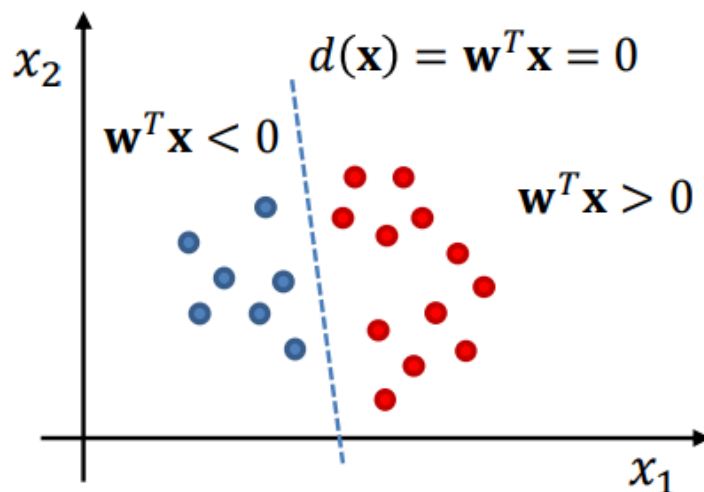

Fig 3. A decision boundary defined by a decision function that divides the feature space into two regions.

ASSIGNMENT 2: Classifying samples

Let's play a bit with a toy example of a linear discriminant function. Consider that the learned weights during the training phase of the classifier are $\mathbf{w} = [1, 0, -3]$, so the linear discriminant function looks like $d(\mathbf{x}) = 1x_1 + 0x_2 - 3$.

Your task is to:

- classify some 2D data using the previous discriminant function (for that, use a combination of `np.dot()` [(https://numpy.org/doc/stable/reference/generated/numpy.dot.html)](https://numpy.org/doc/stable/reference/generated/numpy.dot.html) and `np.sign()` [(https://numpy.org/doc/stable/reference/generated/numpy.sign.html)](https://numpy.org/doc/stable/reference/generated/numpy.sign.html)), and
- plot them with different colors depending on their assigned class using `plt.scatter()` [(https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.scatter.html)](https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.scatter.html) and its `c` argument.

Tip: [how to color a data point using its class (https://stackoverflow.com/questions/43579626/pandas-plot-with-positive-values-one-color-and-negative-values-another)](https://stackoverflow.com/questions/43579626/pandas-plot-with-positive-values-one-color-and-negative-values-another).

In [37]:
```python
# Define weight vector
w = np.array([[1],[0],[-3]])

# Define data matrix
data = np.array([[2,2,1],[1,2,1],[2,1,1],[5,4,1],[4,5,1],[4,4,1]])

# Compute class for each data
classification = np.dot(data,w)
class_ = np.sign(classification)

# Plot classified data
plt.scatter(data[:,0],data[:,1],c=class_.flatten(), cmap="bwr")

# Plot discriminant line
x = np.linspace(3, 3, 1000)
y = np.linspace(0, 5, 1000)
plt.plot(x,y);
```
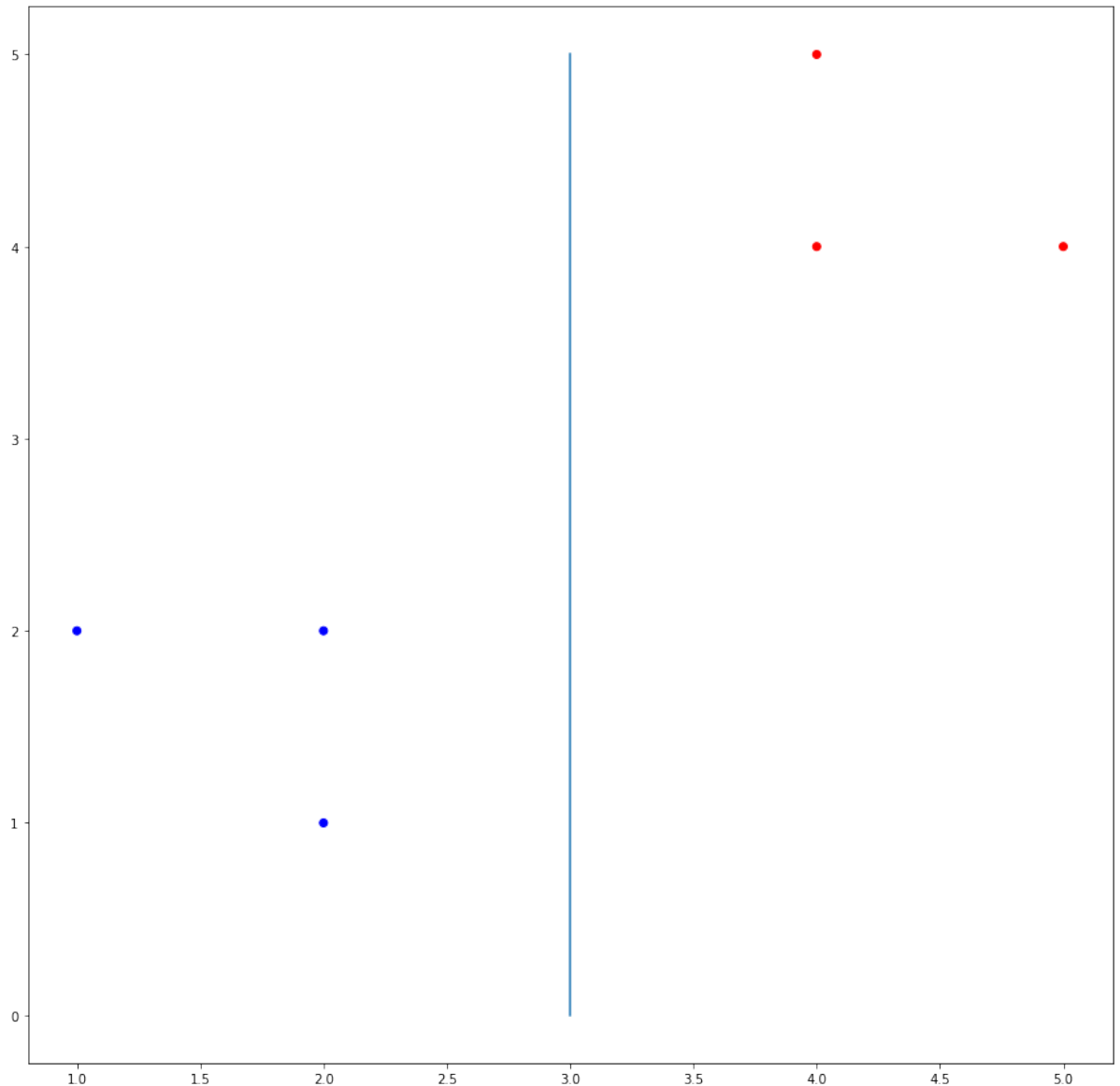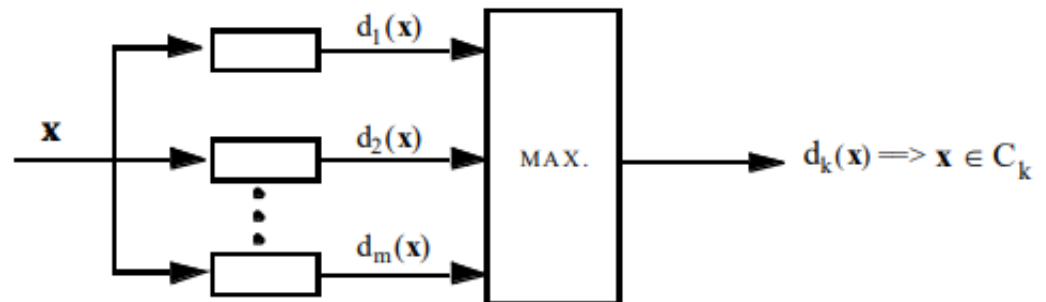
Scenarios with N-classes

The previous idea can be generalized to an arbitrary number of classes, defining in that case hyper-planes in a n-dimensional space:

$$d_i(\mathbf{x}) = w_1^i x_1 + w_2^i x_2 + \ldots + w_n^i x_n + w_{n+1}^i = \mathbf{w^i}^T \cdot \mathbf{x}$$

In this case, the object is assigned to the category which discriminant function returns the highest value. That is:

$$\text{IF } d_i(\mathbf{x}) > d_j(\mathbf{x}) \ \forall i \neq j \ \text{ THEN } \mathbf{x} \in C_i$$

When more than 2 classes are considered, **the discriminant functions are no longer the decision boundaries between classes**. However, the decision boundary between classes C_i and C_j can be computed as:

$$d_{ij}(\mathbf{x}) = d_i(\mathbf{x}) - d_j(\mathbf{x}) = (\mathbf{w}_i^T - \mathbf{x}_j^T) \cdot \mathbf{x} = \mathbf{w}_{ij}^T \cdot \mathbf{x}$$

The following image shows an example with 3 classes (notice that it also exists a decision boundary between classes 1 and 3, $d_{13}(\mathbf{x})$, but it has been omitted for clarity).
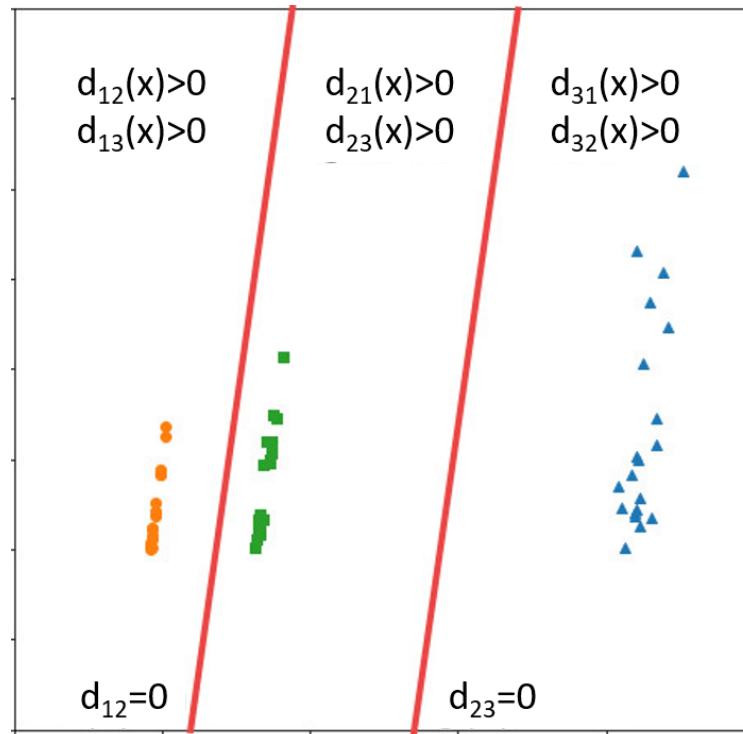
$$d_{12}(x)>0 \quad d_{21}(x)>0 \quad d_{31}(x)>0$$
$$d_{13}(x)>0 \quad d_{23}(x)>0 \quad d_{32}(x)>0$$

$$d_{12}=0 \qquad d_{23}=0$$

Fig 4. Decision boundaries between 3 categories.

7.1.2 Separability

An important concept regarding object recognition is such of **separability**. In this way, two sets of points defining two classes may be separable or not in a given dimension (e.g. x_1, x_2, etc.).

Two sets are **linearly separable** if it exists at least one line in the plane that leaves all the points belonging to one class on one side of the line and all those belonging to the second class on the other side. This idea immediately generalizes to higher-dimensional Euclidean spaces if the line is replaced by a hyperplane.

Some illustrative examples of this:
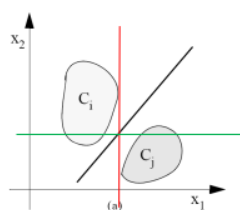
- Classes separable with just x_1
- Classes DO NOT separable with x_2
- x_1 more discriminative that x_2
- In x_1-x_2 classes are separables

- As you can see, those classes are separable. Also, if only dimension x_1 is used, we can separate the classes as well. This is not recomended because there isn't a clear separation using only x_1, and x_2 is also **discriminative**, that is, it also provides valuable information for separating both classes.
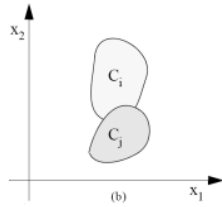
- Classes are not separable in x_1-x_2
- The feature x_1 provides nathing! → NO discriminantive at all
- Can be separable introducing an additional feature x_3

- In this example the classes are not separable as they overlap in both dimensions. Additionally, we can see that x_1 is not discriminative as it doesn't provide any information for separating those classes. The addition of a third dimension/feature could fix this!
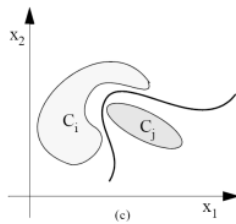
- Classes are not separable with x_2
- Clases NOT-linearly separables in x_1-x_2
- Can be separated with a Not-linear discriminant function

- This is an example of a non-linear separable problem. The classes can't be separated using a line, but a **generalized discriminant function** could do it. We will see this later in this notebook.

Back to sign shape recognition

Back to our traffic sign shape recognition problem, **we are trying to separate** the usual shapes of traffic signs using the Hu moments. As a first approximation to this problem, we plotted the first and second Hu moments (x_1 and x_2) in order to analyze them. You should have obtained something like this:
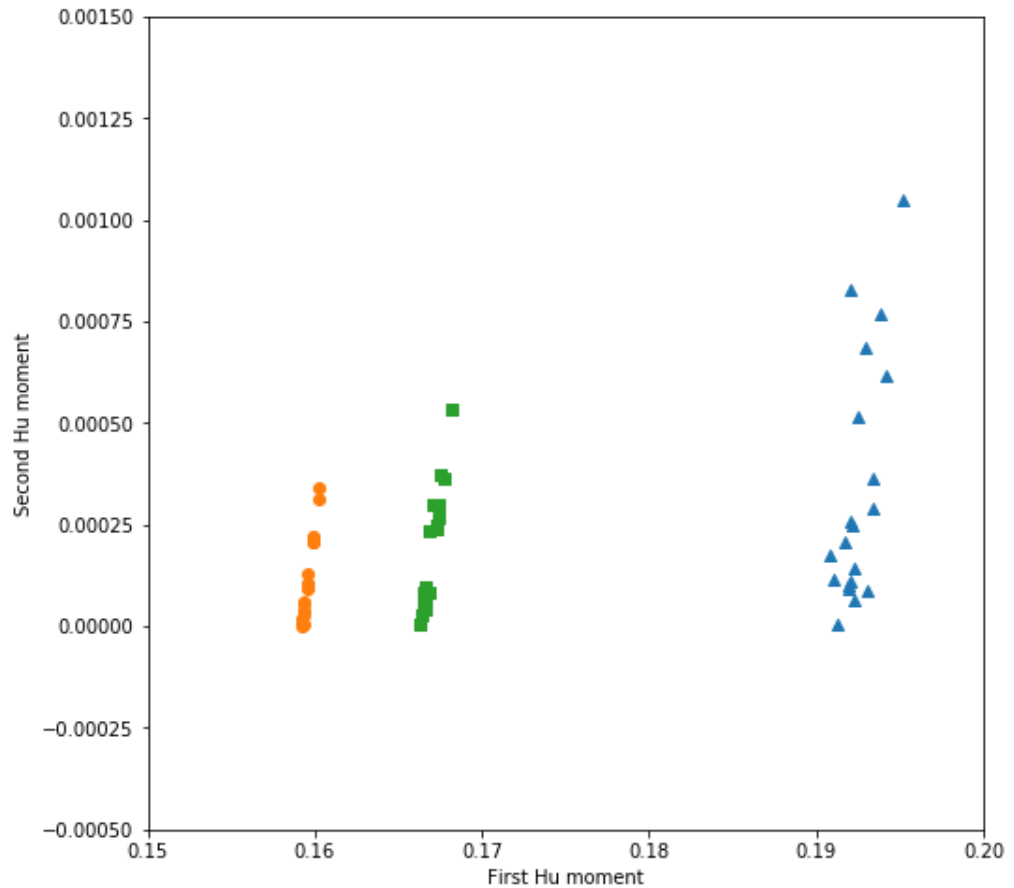

Fig 5. Result of plotting the two first Hu moments computed from the images in our train data.

Thinking about it (1)

Now you are in a good position to answer these questions:

- Is this problem linearly separable?

  Yes, a straigth line would separate the groups clearly.
- Are the classes separable using only the first Hu moment (x_1)?

  Yes, we could see it by the groups distribution.
- Are the classes separable using only the second Hu moment (x_2)?

  Clearly no, we could not distinguish.
- Which dimension is more discriminative?

  The discriminative is the first Hu Moment.
- Would be *AliquindoiCars* guys happy if we propose a solution for shape classification based on linear discriminant functions?

  Yes, linear functions separete the objects perfectly.

7.1.3 Generalized discriminant function

As we saw in the previous section, some classes are not separable using a linear discriminant function. In those cases, we may need a **linear basis function model**, also called **generalized discriminant function**, which permit us to use non-linear functions to separate classes (notice that, despite of this, the problem may still not be separable).

The idea behind these discriminant functions is to transform the $\mathbf{x}$ space into a $\mathbf{x'} = \mathbf{f(x)}$ space ($dim(x) < dim(x')$):

$$d(x) = w_1 \; f_1(x) + w_2 \; f_2(x) + \ldots + w_k \; f_k(x) + w_{k+1} = \sum_{i=1}^{k+1} w_i \; \underbrace{f_i(x)}_{\text{New space}} = \mathbf{w}^T \cdot \mathbf{f(x)}$$

$$d(x) = w_1 \; x'_1 + w_2 \; x'_2 + \ldots + w_k \; x'_k + w_{k+1} = \sum_{i=1}^{k+1} w_i \; x'_i = w^T \cdot x' = d'(\mathbf{x'})$$

Thereby:

$$\left. \begin{array}{c} \mathbf{x'} = \mathbf{f(x)} = [f_1(\mathbf{x}) \quad f_2(\mathbf{x}) \quad \cdots \quad f_k(\mathbf{x}) \quad 1]^T \\ \mathbf{w} = [w_1 \quad w_2 \quad \cdots \quad w_k \quad w_{k+1}]^T \end{array} \right\} \; d(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} = \mathbf{w}^T \cdot \mathbf{f(x)}$$

being $\mathbf{f(x)}$ the basis functions!

ASSIGNMENT 3: Transforming between spaces

Let's practice this transformation between spaces. **What to do?** You have to complete the method `transform_space()`, which transforms a set of 2D points into a new space defined by:

$$x' = [x_1^2, \ x_1 x_2, \ x_2^2, \ x_1, \ x_2, \ 1]$$

The input set of points is a matrix with shape $(n\_data, 2)$, and the output is another matrix with shape $(n\_data, 6)$, being n_data the number of points in the set.

In [ ]:
```python
# Assignment 3
def transform_space(data_points):
    """ Compute moments of the external contour in a binary image.

        Args:
            data_points: Input matrix containing a set of 2D points

        Returns:
            fx: set of points transformed to a quadratic space
    """

    n_data = data_points.shape[0]

    fx = np.zeros((n_data,6))

    for i in range(n_data):

        x1 = data_points[i,0]
        x2 = data_points[i,1]

        fx[i,:] = [x1**2, x1*x2, x2**2, x1, x2, 1]

    return fx
```

You can use next code to **test if the results are correct**:

In [36]:
```python
data = np.array([[2,2],[1,2],[2,1],[5,4],[4,5],[4,4]])

x = transform_space(data)

print(x)
```
```
[[ 4.  4.  4.  2.  2.  1.]
 [ 1.  2.  4.  1.  2.  1.]
 [ 4.  2.  1.  2.  1.  1.]
 [25. 20. 16.  5.  4.  1.]
 [16. 20. 25.  4.  5.  1.]
 [16. 16. 16.  4.  4.  1.]]
```

```
[[ 4.  4.  4.  2.  2.  1.]
 [ 1.  2.  4.  1.  2.  1.]
 [ 4.  2.  1.  2.  1.  1.]
 [25. 20. 16.  5.  4.  1.]
 [16. 20. 25.  4.  5.  1.]
 [16. 16. 16.  4.  4.  1.]]
```

ASSIGNMENT 4: Classifying in the transformed space

Finally, we will see a toy example of a non-linear discriminant function in action. Concretely, we will consider a quadratic function, $d(x) = x_1^2 - 5x_2$ (so $k = 5$), in a two classes problem with $\mathbf{x} = [x_1 \ \ x_2]$. Thereby, the vector of features in the new space is:

$$\mathbf{x}' = [x_1', \dots, x_6'] = [x_1^2 \ \ x_1 x_2 \ \ x_2^2 \ \ x_1 \ \ x_2 \ \ 1]^T$$

so:

- $x_1^2 = f_1(\mathbf{x}) = x_1'$
- $x_1 x_2 = f_2(\mathbf{x}) = x_2'$
- and so on.

and the vector of weights turns out to be:

$$\mathbf{w} = [1, 0, 0, 0, -5, 0] \ .$$

As for the new discriminant function we get:

$$d(\mathbf{x}) = w_1 \cdot x_1' + w_5 \cdot x_5'$$

Your task is to classify some 2D data using the previous discriminant function and plot them coloring each point with a color depending on its class. For that, you can rely on the `np.dot()` (https://numpy.org/doc/stable/reference/generated/numpy.dot.html), `np.sign()` (https://numpy.org/doc/stable/reference/generated/numpy.sign.html)), and `plt.scatter()` (https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.scatter.html) functions previously used in the second assignment.

Tip: Remember to transform the data space before applying the dot product.
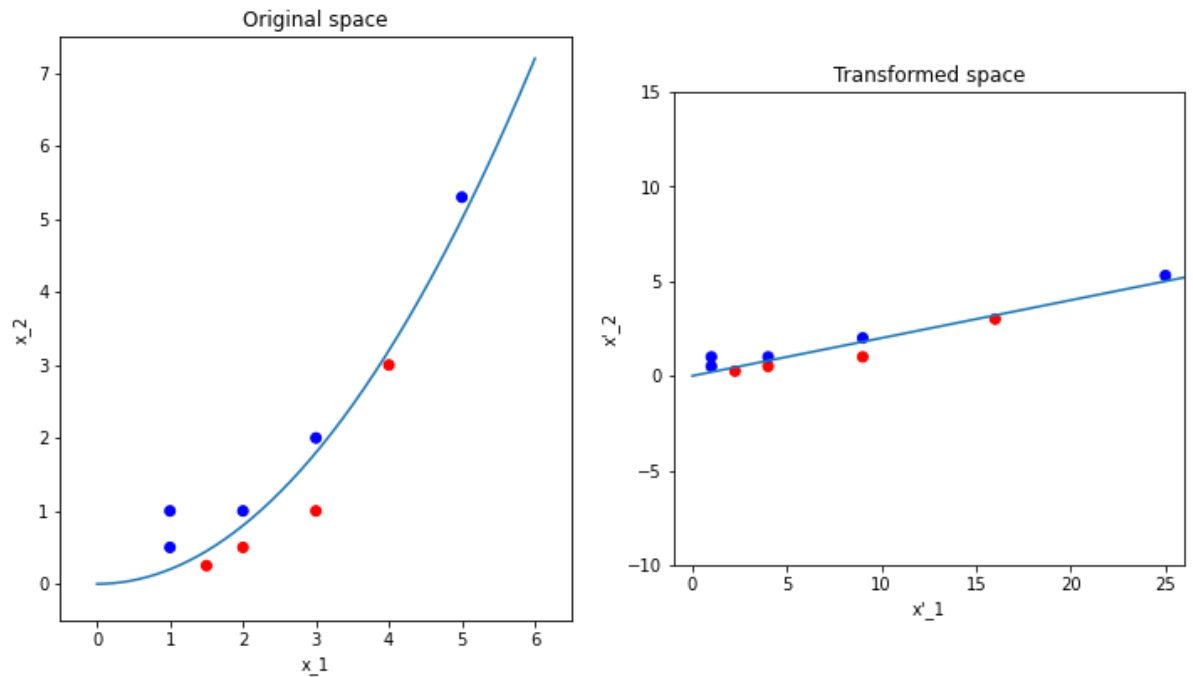
Expected results:

Fig 6. Classification of a number of samples in both, the original space and the transformed one.

In [35]:
```python
# Assignment 4

# Define weight vector
w = np.array([[1],[0],[0],[0],[-5],[0]])

# Define data matrix
data = np.array([[2,0.5],[1.5,0.25],[3,1],[1,1],[1,0.5],[2,1],[5,5.

# Trasform from the x space to a f(x) space
transformed_data = transform_space(data)

# Compute class for each data
classification = np.dot(transformed_data,w)
class_ = np.sign(classification)

# Plot classified data
matplotlib.rcParams['figure.figsize'] = (15.0, 15.0)
ax = plt.subplot(121)
ax.set_aspect('equal')
plt.title('Original space')
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.axis([-0.5, 6.5, -0.5, 7.5])
plt.scatter(data[:,0],data[:,1],c=class_.flatten(), cmap="bwr")

# Plot discriminant line
x = np.linspace(0, 6, 1000)
y = (w[0]*(x**2) + w[5])/-w[4]
plt.plot(x,y);
```

```python
# Or equivalently plotting the data in the transformed space,
# where the descision boundary defined by the discrimiant function
ax = plt.subplot(122)
ax.set_aspect('equal')
plt.title('Transformed space')
plt.xlabel("x'_1")
plt.ylabel("x'_2")
plt.axis([-1, 26, -10, 15])
plt.scatter(transformed_data[:,0],transformed_data[:,4],c=class_.fl

# Plot discrimant line
x = np.linspace(0, 27, 1000)
y = (w[0]*(x) + w[5])/-w[4]
plt.plot(x,y);
```
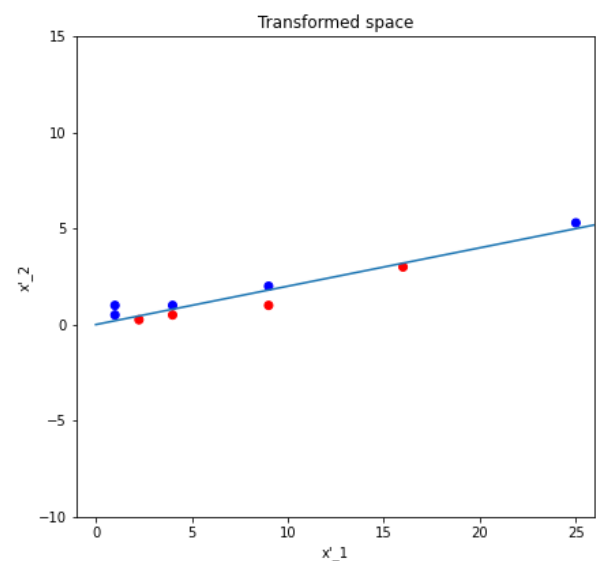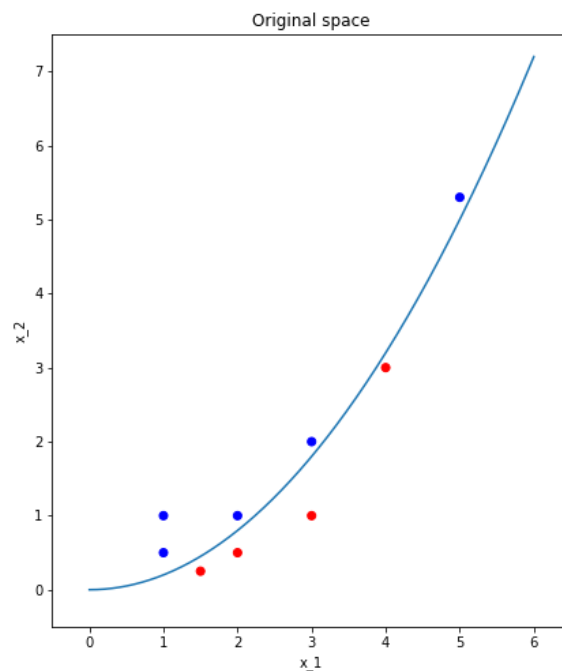
Conclusion

Well done! This was a mostly theoretical notebook, but it described the basis of object classification. You have learned so far:

- how linear and non-linear discriminant functions work,
- to classify objects in a scenario with an arbitrary number of belonging classes using multiple discriminant functions, and
- to check if a feature space allows for the definition of discriminative classifiers looking at the separability between classes.

In the following notebooks in this chapter we will see how to automatically retrieve the weight vector from a set o training data. Exciting, isn't?