Computer Vision

Histogram of Oriented Gradients
NSM
Image Classification

Fátima Leal

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UNIVERSIDADE PORTUCALENSE

What we have learnt

- Hough Transform
- Morphological operations

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Content

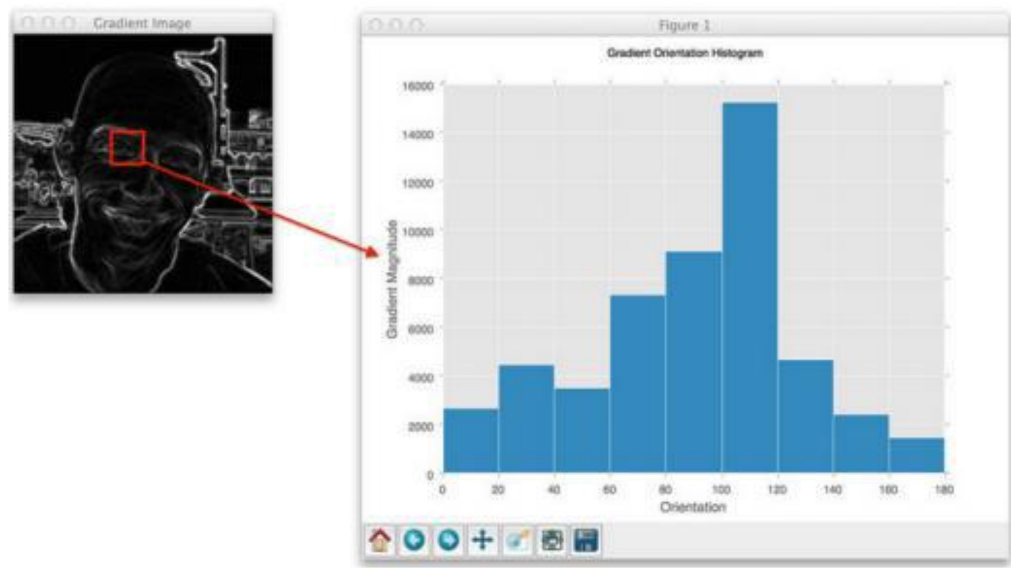
- Feature Descriptor
- Histogram of Oriented Gradients
- Non Maxima Supression
- Image Classification

Feature Descriptor

- Representation of an image which extracts useful information and eliminating unnecessary information.
- Converts an image of size width x height x 3 (channels) to a feature vector / array of length n.
- The feature vector produced by these algorithms when employed into an image classification algorithms produce good results.

Histogram of Oriented Gradients

- HOG is a feature descriptor for object detection
- HOG captures local intensity gradients and edge directions
- The appearance of the object can be modelled by the distribution of intensity gradients inside rectangular regions of an image



DEPARTAMENTO **CIÊNCIA E TECNOLOGIA**

Histogram of Oriented Gradients

- Pros
  - Very powerful descriptor.
  - Excellent at representing local appearance.
  - Extremely useful for representing structural objects that do not demonstrate substantial variation in form *(i.e.* buildings, people walking the street, bicycles leaning against a wall).
  - Very accurate for object classification.

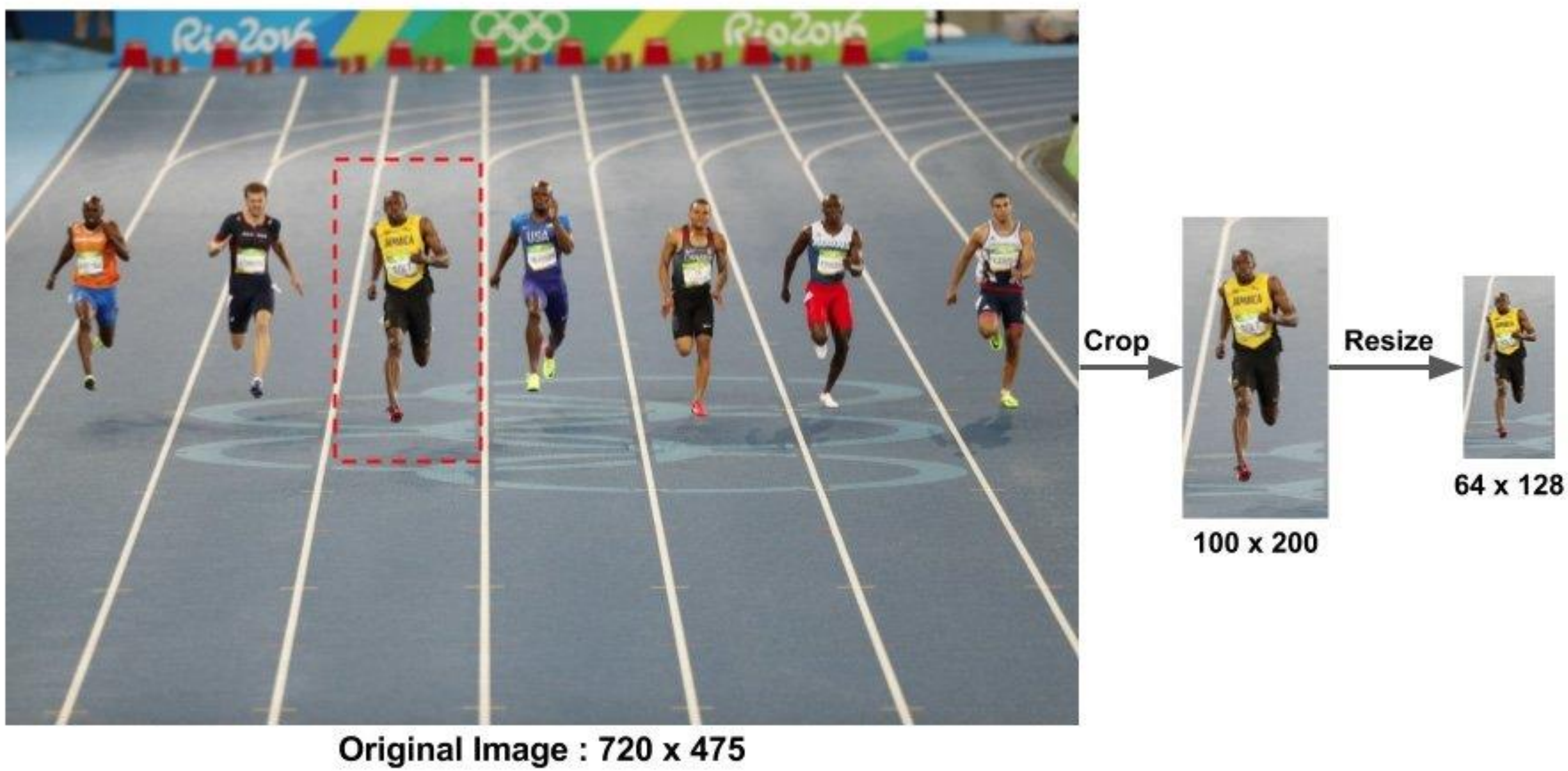
- CONS
  - Can result in very large feature vectors, leading to large storage costs and computationally expensive feature vector comparisons.
  - Often non-trivial to tune the orientations, pixels per cell, and cells per block parameters.
  - Not the slowest descriptor to compute, but also nowhere near the fastest.
  - If the object to be described exhibits substantial structural variation (i.e. the rotation/orientation of the object is consistently different), then the standard vanilla implementation of HOG will not perform well.

Histogram of Oriented Gradients

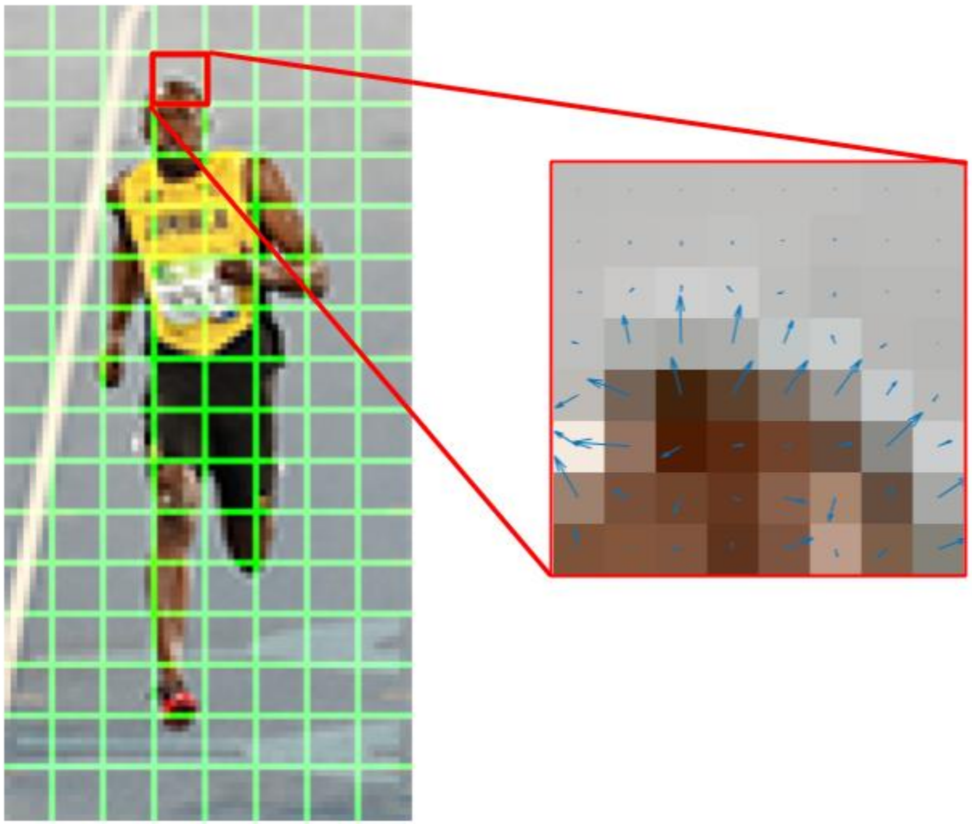
```
import cv2
import imutils

image = cv2.imread("image.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
hog = cv2.HOGDescriptor()
img = imutils.resize(image, width=xxxxx, height=yyyyy)
cv2.imshow("img", img)
cv2.waitKey(0)
descriptor = hog.compute(img)
print(len(descriptor))
```

Histogram of Oriented Gradients



Histogram of Oriented Gradients



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

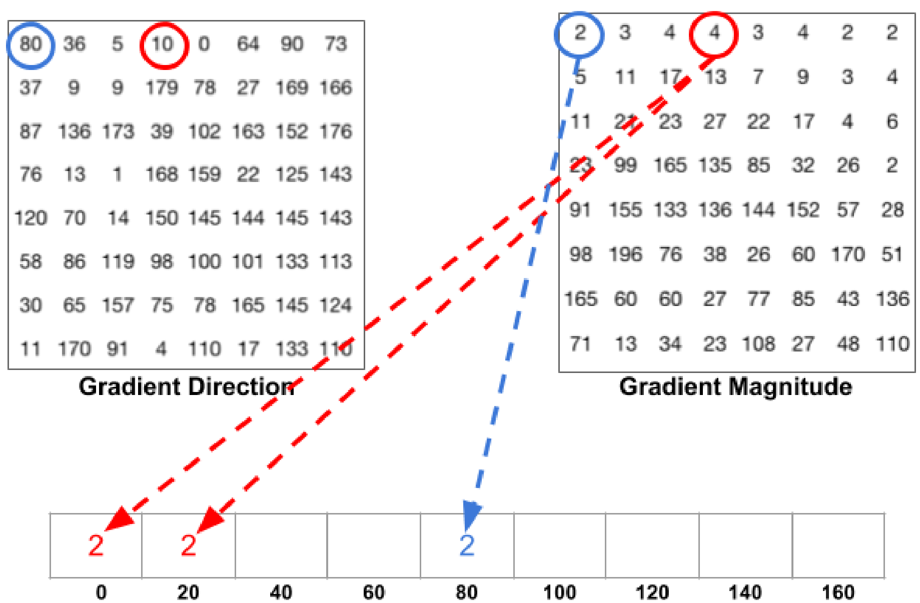
Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Histogram of Oriented Gradients

- the arrow indicate the direction of gradient and its length the magnitude
- The direction of arrows points to the direction of change in intensity
- The magnitude shows how big the difference is.



Histogram of Gradients

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

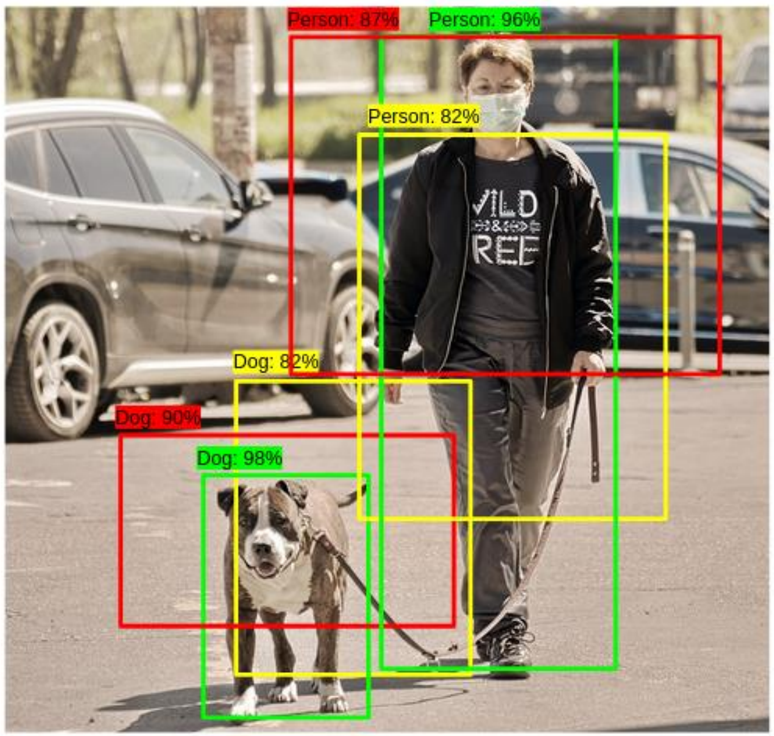
Histogram of Oriented Gradients

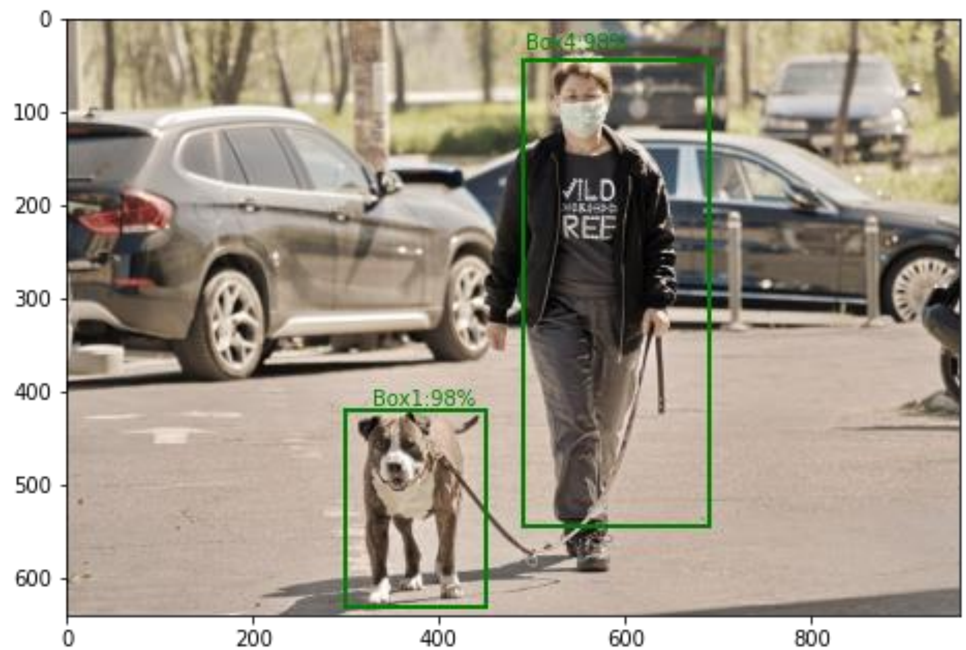
https://www.youtube.com/watch?v=itmV7druy9Y

Non Maxima Suppression

- When you are detecting objects in images you will detect multiple bounding boxes surrounding the object in the image.
- This is a very common “problem” when utilizing object detection methods.
- To fix this situation you’ll need to apply Non-Maximum Suppression (NMS), also called Non-Maxima Suppression.
- The NMS selects the box with highest Intersection Over Union (IoU)

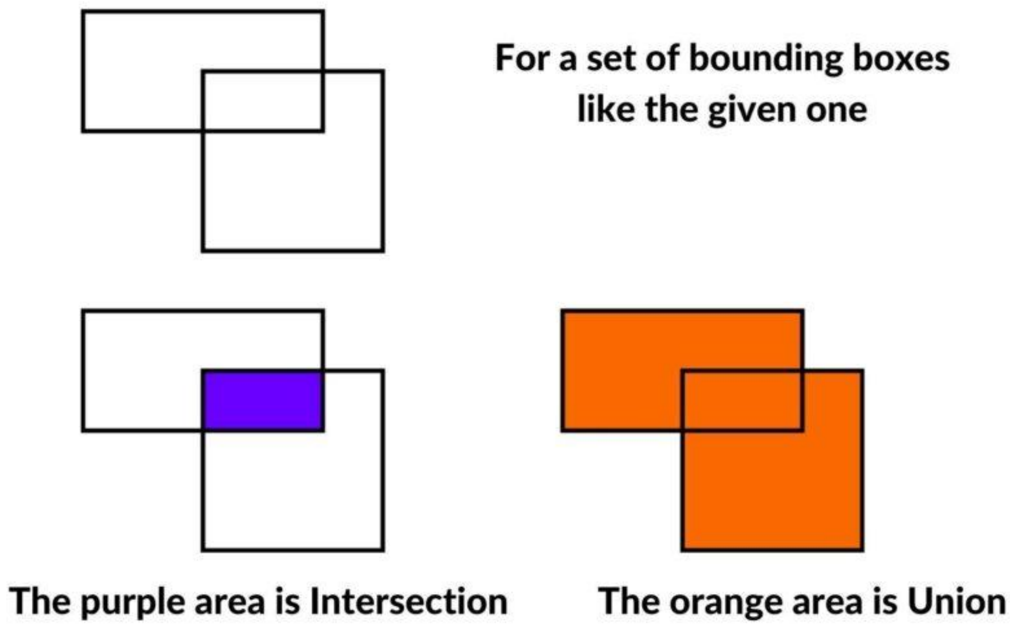
Non Maxima Suppression





Non Maxima Suppression

Intersection over Union (IoU)



DEPARTAMENTO CIÊNCIA E TECNOLOGIA

IMAGE CLASSIFICATION

Pattern Recognition

- **Representation** of an object category.
- **Learning** - model a classifier using training data.
- **Recognition -** classification

- The learning process can be:
  - **Supervised**: method where there is a teacher who says which decision is wrong or correct.
  - **Unsupervised** method where there is not a teacher. Learning consists by forming natural groups of the inputs by a specific learning methodology which by exchanging different costs or patterns leads to different groups.

Pattern Recognition

- How can we build a model?
  - Select a dataset and create the training and test sets
  - Select and extract a set of features to each image
  - Choose a classifier and adapt the parameters of the model using the training set
  - Compute the error using the test set

Pattern Recognition

- The learning strategy can be:
  - The **one-versus-all** strategy is compound by several of binary classifiers, one for each class.

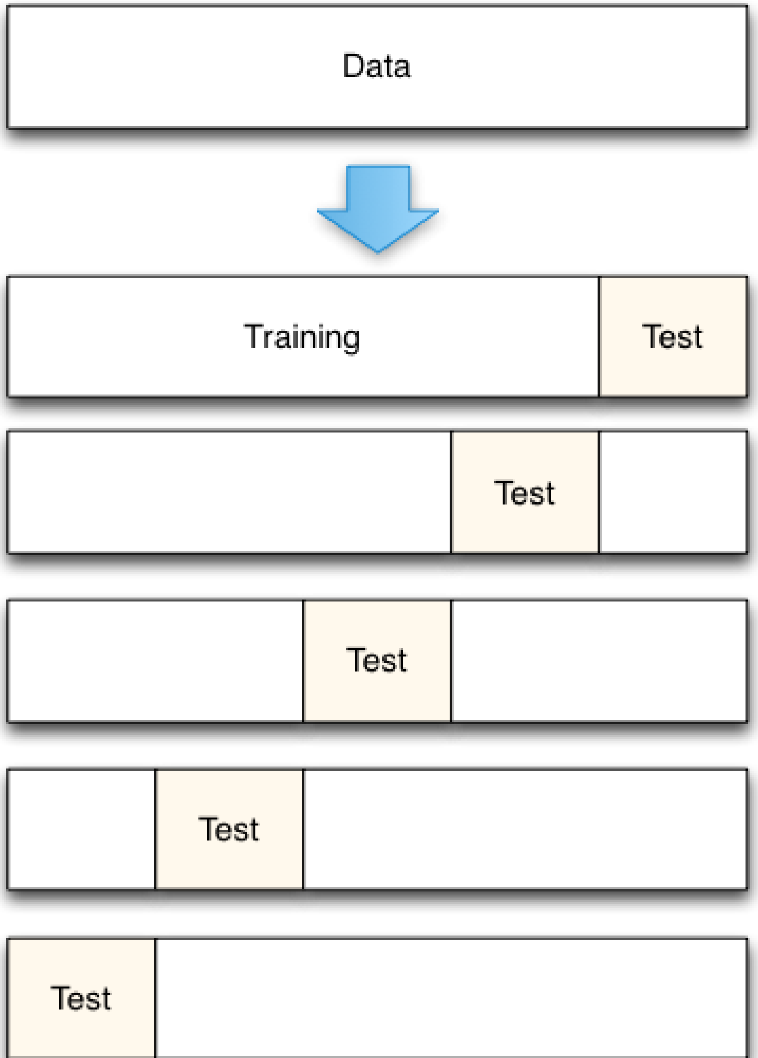
  For example, given a multi-class classification problem with examples for each class ‘red,’ ‘blue,’ and ‘green‘. This could be divided into three binary classification datasets as follows:
  - Binary Classification Problem 1: red vs [blue, green]
  - Binary Classification Problem 2: blue vs [red, green]
  - Binary Classification Problem 3: green vs [red, blue]

Pattern Recognition

- The **one-versus-one** strategy splits a multi-class classification dataset into binary classification problems

- Binary Classification Problem 1: red vs. blue
- Binary Classification Problem 2: red vs. green
- Binary Classification Problem 3: red vs. yellow
- Binary Classification Problem 4: blue vs. green
- Binary Classification Problem 5: blue vs. yellow
- Binary Classification Problem 6: green vs. yellow

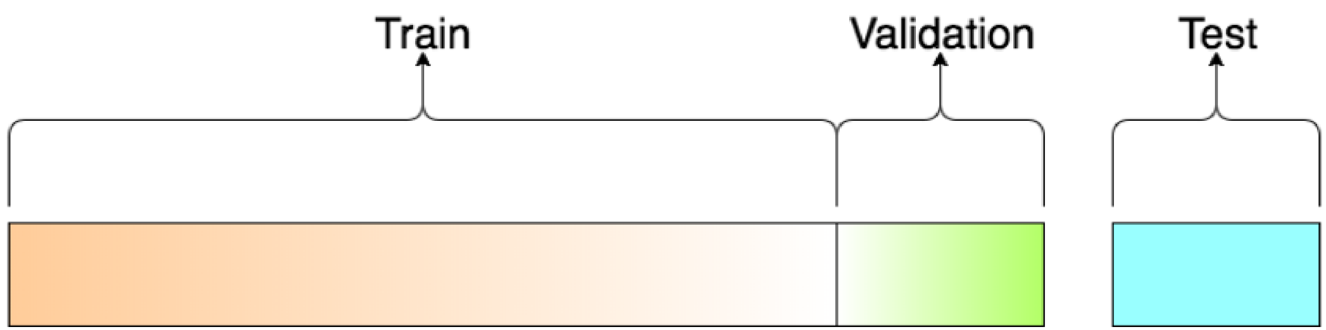
Pattern Recognition



Machine Learning

Machine Learning

- **Training Dataset**: Examines the data and creates the model
- **Validation Dataset**: Learn from model mistakes
- **Test Dataset**: Making a conclusion on how well the model performs.



A visualization of the splits

Machine Learning

x		y
1	2	0
3	4	1
5	6	1
7	8	0
9	10	1
11	12	0
13	14	0
15	16	1
17	18	1
19	20	0
21	22	1
23	24	0

x		y	
17	18	1	Training set
5	6	1	Training set
23	24	0	Training set
1	2	0	Training set
3	4	1	Training set
11	12	0	Training set
15	16	1	Training set
21	22	1	Training set
7	8	0	Test set
9	10	1	Test set
13	14	0	Test set
19	20	0	Test set

Machine Learning Model

ML Results

x		y
7	8	1
9	10	1
13	14	1
19	20	0

ML Results

x		y
7	8	1
9	10	1
13	14	1
19	20	0

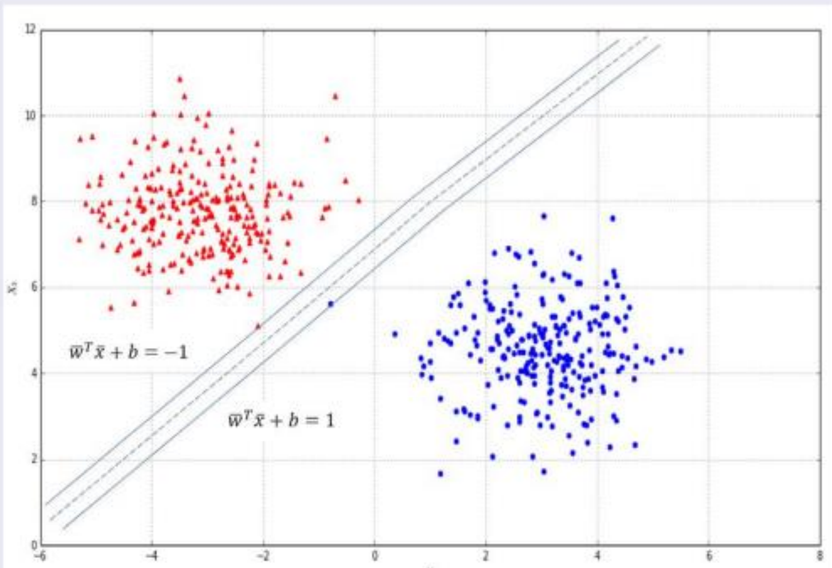
Support Vector Machine

- Let's consider a dataset of feature vectors we want to classify: $X = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_n\}$, where $\bar{x}_i \in \mathbf{R}^m$
- For simplicity, we assume it as a binary classification (in all the other cases, it's possible to use automatically the one-versus-all strategy) and we set our class labels as -1 and 1: $Y = \{y_1, y_2, ..., y_n\}$, where $y_n \in \{-1, 1\}$
- Our goal is to find the best separating hyperplane, for which the equation is:$\bar{w}^T\bar{x} + b = 0$, where $\bar{w} = W_1 ... W_m)$ and $\bar{x} = x_1 ... x_m)$
- In this way, our classifier can be written as: $\bar{y} = f(\bar{(}x)) = sgn(\bar{w}^T\bar{x} + b)$

UPT DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

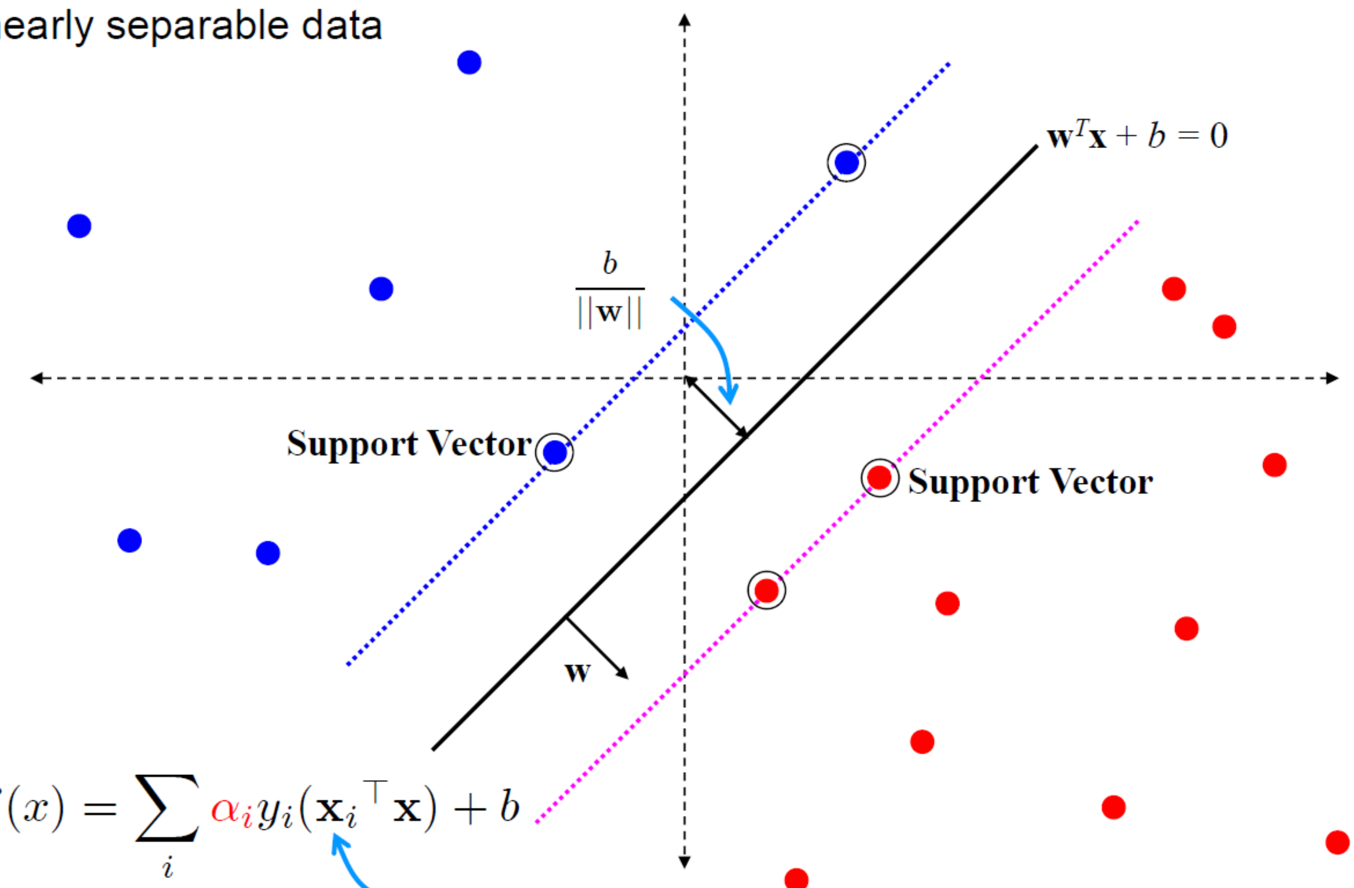
Support Vector Machines

- In a realistic scenario, the two classes are normally separated by a margin with two boundaries where a few elements lie. Those elements are called support vectors. For a more generic mathematical expression, it's preferable to renormalize our dataset so that the support vectors will lie on two hyperplanes with equations: $\bar{w}^T\bar{x} + b = -1$ and $\bar{w}^T\bar{x} + b = 1$



Support Vector Machines

linearly separable data

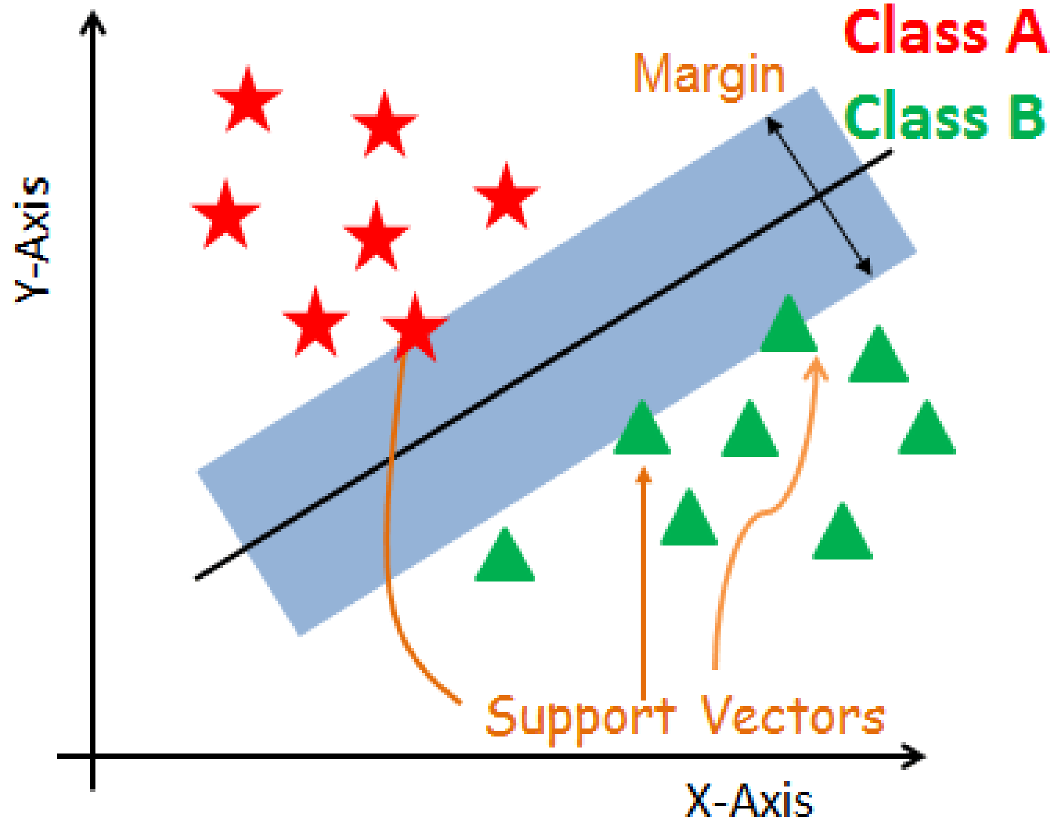


$$f(x) = \sum_i \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$$

support vectors

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Support Vector Machines



```
#Import scikit-learn dataset library
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import svm

#Load dataset
cancer = datasets.load_breast_cancer()
dia = datasets.load_diabetes()
# print the names of the 13 features (X)
print("Features: ", cancer.feature_names)
# print the label type of cancer('malignant' 'benign') (Y)
print("Labels: ", cancer.target_names)

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3,random_state=109) # 70%
training and 30% test

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Let's play with images!

python™

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.