Tests & Features

IML

Cédric Buche

ENIB

29 août 2019

2019-08-29

IML

Tests & Features
IML
Cédric Buche
ENIB
29 août 2019

Page 1 :

2019-08-29

IML

1. Tests
   - Training vs Testing
   - K-Fold Cross Validation
   - Model performance
2. Feature extraction
   - Feature
   - Feature extraction
   - Image processing : Object detection and tracking

Page 2 :

1. Tests
   - Training vs Testing
   - K-Fold Cross Validation
   - Model performance
2. Feature extraction
   - Feature
   - Feature extraction
   - Image processing : Object detection and tracking

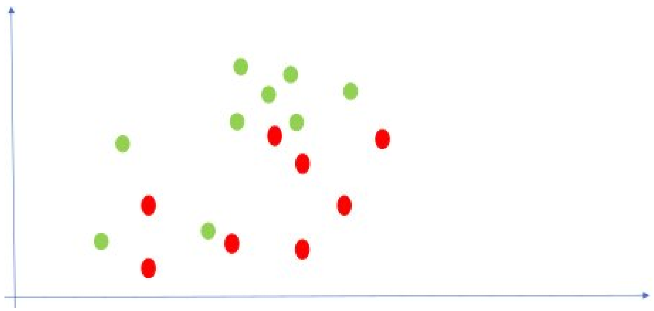
Page 3 :

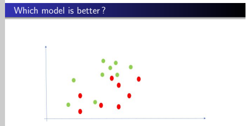
Testing

- ▷ How well is my model doing ?
- ▷ How do I improve it ?

Page 4 :

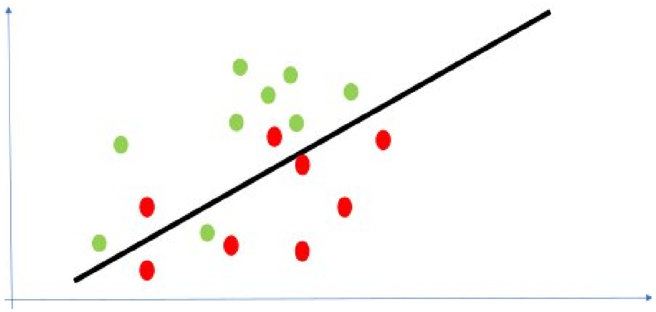
Which model is better ?

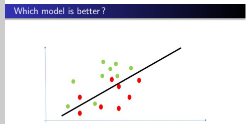




Page 5 :

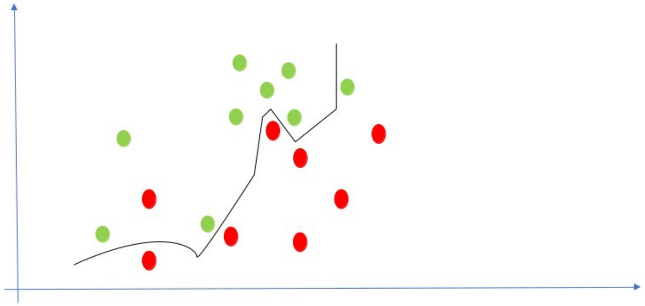
Which model is better ?

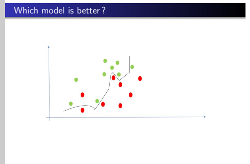




Page 6 :

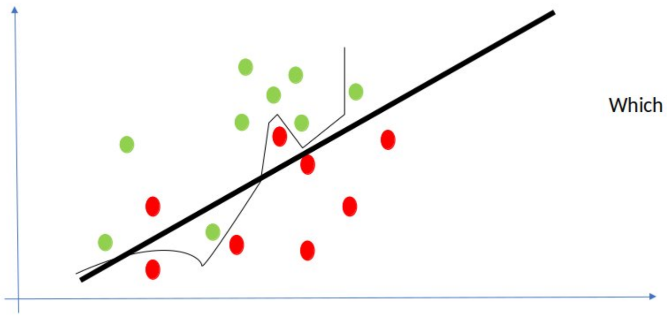
Which model is better ?



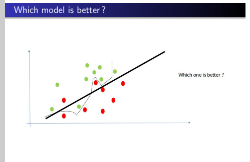


Page 7 :

Which model is better ?

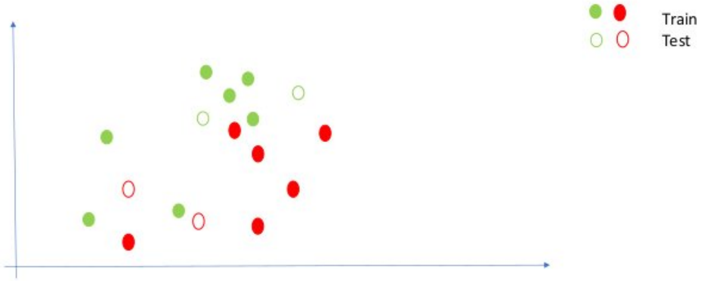


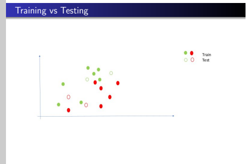
Which one is better ?



Page 8 :

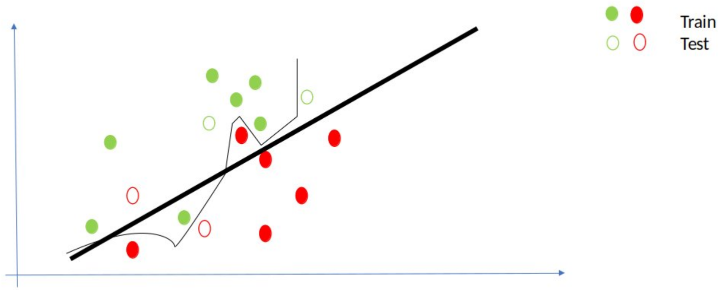
Training vs Testing

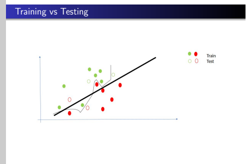




Page 9 :

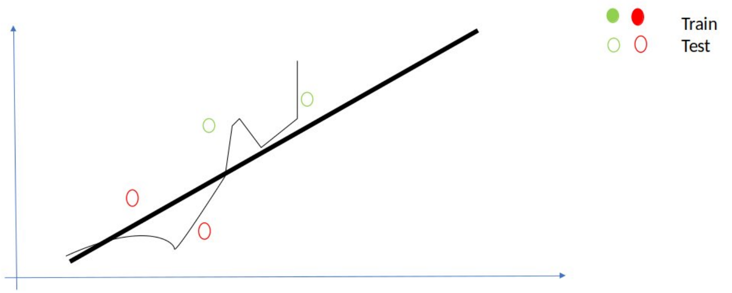
Training vs Testing





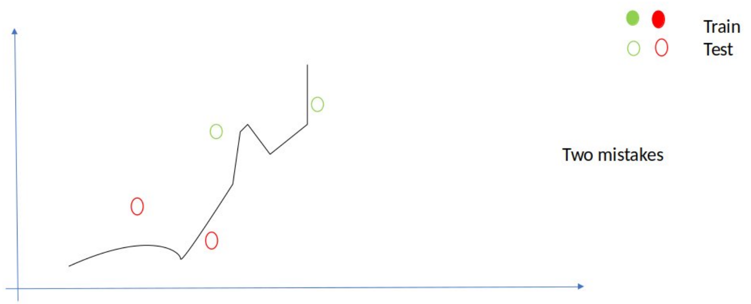
Page 10 :

Training vs Testing



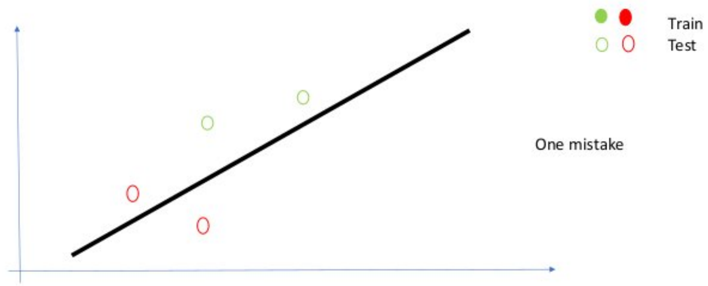
Page 11 :

Training vs Testing



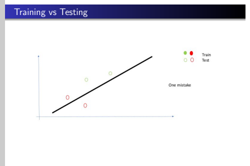
Page 12 :

Training vs Testing



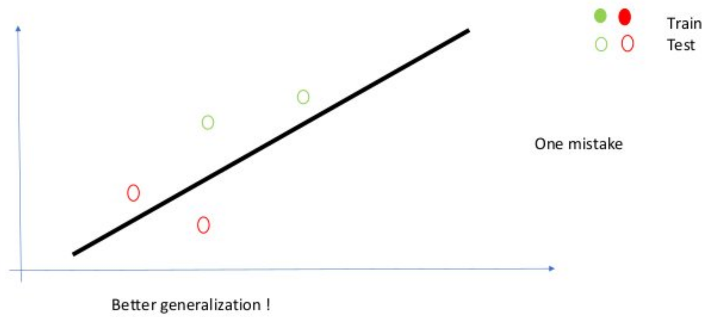
2019-08-29

IML
└Tests
└Training vs Testing
└Training vs Testing



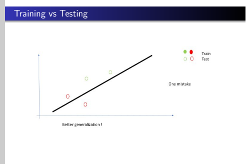
Page 13 :

Training vs Testing



2019-08-29

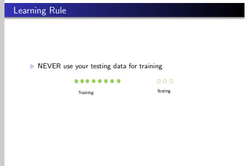
IML
└Tests
└Training vs Testing
└Training vs Testing



Page 14 :

Learning Rule

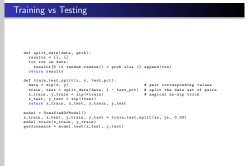
- ▷ NEVER use your testing data for training



Page 15 :

Training vs Testing

```python
def split_data(data, prob):
  results = [], []
  for row in data:
    results[0 if random.random() < prob else 1].append(row)
  return results

def train_test_split(x, y, test_pct):
  data = zip(x, y)                              # pair corresponding values
  train, test = split_data(data, 1 - test_pct)  # split the data set of pairs
  x_train, y_train = zip(*train)                # magical un-zip trick
  x_test, y_test = zip(*test)
  return x_train, x_test, y_train, y_test

model = SomeKindOfModel()
x_train, x_test, y_train, y_test = train_test_split(xs, ys, 0.33)
model.train(x_train, y_train)
performance = model.test(x_test, y_test)
```

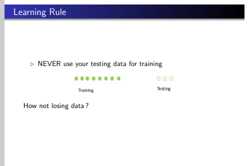



Page 16 :

Learning Rule

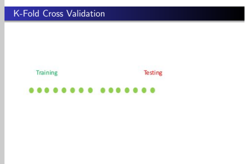
▷ NEVER use your testing data for training

How not losing data ?



Page 17 :

K-Fold Cross Validation

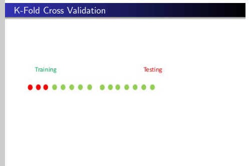


Page 18 :

K-Fold Cross Validation

2019-08-29

IML
└Tests
└K-Fold Cross Validation
└K-Fold Cross Validation

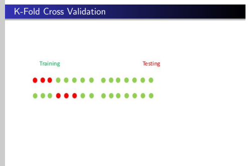


Page 19 :

K-Fold Cross Validation

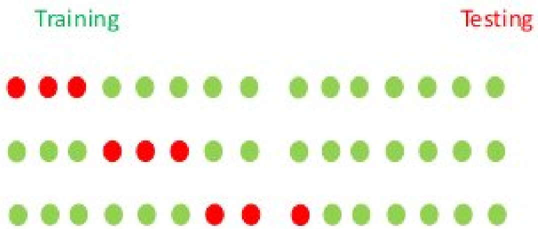
2019-08-29

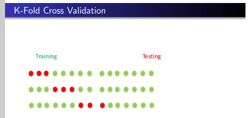
IML
└Tests
└K-Fold Cross Validation
└K-Fold Cross Validation



Page 20 :

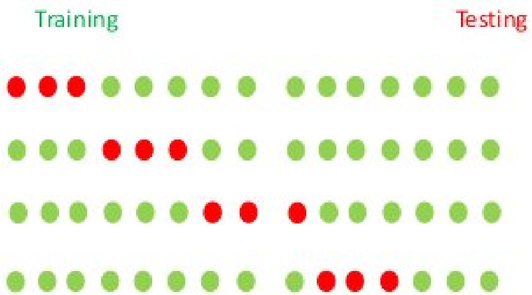
K-Fold Cross Validation

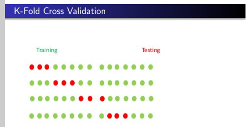




Page 21 :

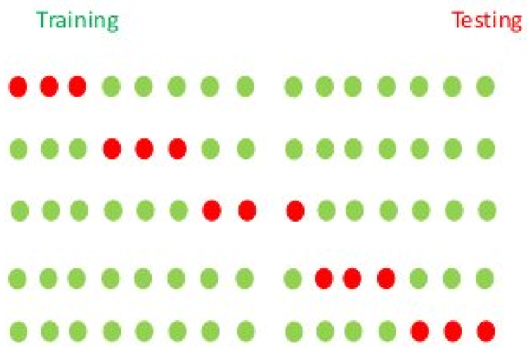
K-Fold Cross Validation

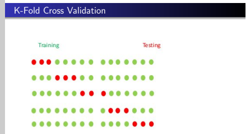




Page 22 :

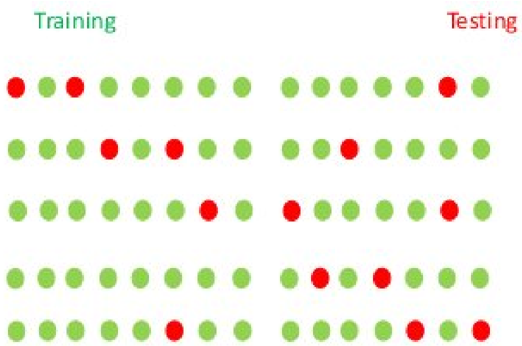
K-Fold Cross Validation

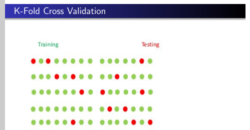




Page 23 :

K-Fold Cross Validation





Page 24 :

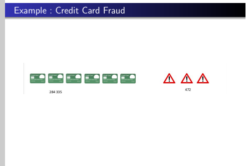
- How well is my model doing ?
  - Tricky question

Page 25 :

Example : Credit Card Fraud

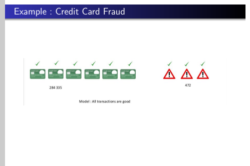
Page 26 :

Example : Credit Card Fraud



Page 27 :

Example : Credit Card Fraud

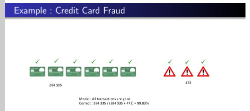


Page 28 :

Example : Credit Card Fraud

2019-08-29

IML
└Tests
  └Model performance
    └Example : Credit Card Fraud

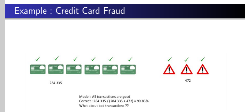


Page 29 :

Example : Credit Card Fraud

2019-08-29

IML
└Tests
  └Model performance
    └Example : Credit Card Fraud

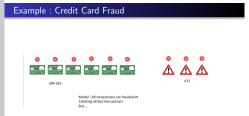


Page 30 :

Example : Credit Card Fraud

2019-08-29

IML
└Tests
└Model performance
└Example : Credit Card Fraud

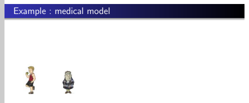


Page 31 :

Example : medical model

2019-08-29

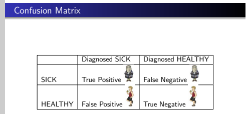
IML
└Tests
└Model performance
└Example : medical model



Page 32 :

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK	True Positive 	False Negative 
HEALTHY	False Positive 	True Negative 

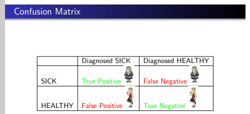


Page 33 :

Tests
Feature extraction
Training vs Testing
K-Fold Cross Validation
Model performance

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK	True Positive 	False Negative 
HEALTHY	False Positive 	True Negative 



Page 34 :

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Page 35 :

Example : spam model

Example : spam model

Page 36 :

Confusion Matrix

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	True Positive	False Negative
NON SPAM	False Positive	True Negative

Page 37 :

Tests
Feature extraction
Training vs Testing
K-Fold Cross Validation
Model performance

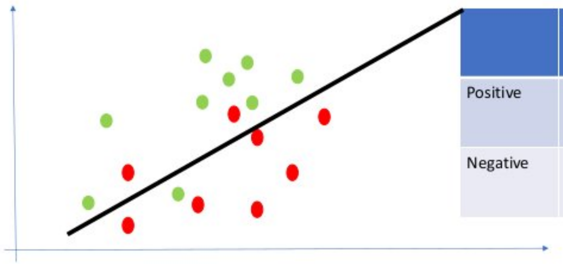
Confusion Matrix

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

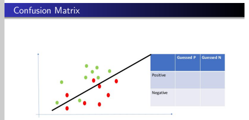
2019-08-29
IML
Tests
Model performance
Confusion Matrix

Page 38 :

Confusion Matrix

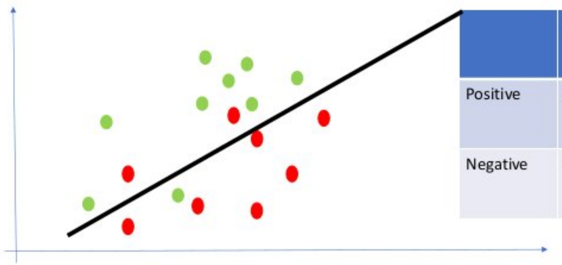


	Guessed P	Guessed N
Positive		
Negative		

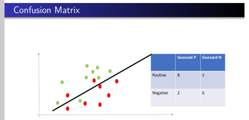


Page 39 :

Confusion Matrix



	Guessed P	Guessed N
Positive	8	1
Negative	2	6



Page 40 :

Accuracy

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Page 41 :

Accuracy

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Accuracy = (1000+8000)/10000 = 90%

Page 42 :

Accuracy

How many did we classify correctly ?

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

Accuracy = 80%
Accuracy = Correctly classified / all

Accuracy
How many did we classify correctly ?
Accuracy = 80%
Accuracy = Correctly classified / all

Page 43 :

Accuracy

```
def accuracy(tp, fp, fn, tn):
  correct = tp + tn
  total = tp + fp + fn + tn
  return correct / total
```

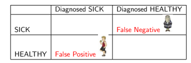

Page 44 :

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK		False Negative 
HEALTHY	False Positive 	

2019-08-29

IML
└Tests
└Model performance
└Confusion Matrix



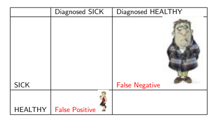
Page 45 :

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK		False Negative 
HEALTHY	False Positive 	

2019-08-29

IML
└Tests
└Model performance
└Confusion Matrix



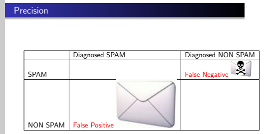
Page 46 :

Precision

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM		False Negative 
NON SPAM	False Positive 	

2019-08-29

IML
└Tests
└Model performance
└Precision



Page 47 :

Precision

- ▷ high PRECISION 
- ▷ high RECALL 

2019-08-29

IML
└Tests
└Model performance
└Precision

Page 48 :

Precision

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Precision = 1000/ (1000+800) =55,7%

2019-08-29

IML
└Tests
└Model performance
└Precision

Page 49 :

Precision

How many did we classify correctly ?

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

Precision= 76.8%

Precision = True Positives/ (True Positives + False Positives)

2019-08-29

IML
└Tests
└Model performance
└Precision

Page 50 :

Precision

```
def precision(tp, fp, fn, tn):
  return tp / (tp + fp)
```


Precision

def precision(tp, fp, fn, tn):
return tp / (tp + fp)

Page 51 :

Recall

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Recall = 1000/ (1000+200) =83.3%

Recall

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Recall = 1000/ (1000+200) =83.3%

Page 52 :

Recall

How many did we classify correctly ?

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

Recall= 37%

Recall = True Positives/ (True Positives + False Negatives)

Recall
How many did we classify correctly ?
Recall= 37%
Recall = True Positives/ (True Positives + False Negatives)

Page 53 :

Recall

```
def recall(tp, fp, fn, tn):
  return tp / (tp + fn)
```

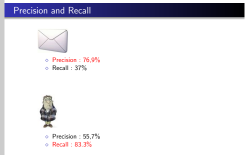

Recall

Page 54 :

Precision and Recall

- Precision : 76,9%
- Recall : 37%

- Precision : 55,7%
- Recall : 83.3%



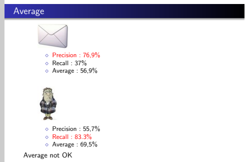
Page 55 :

Average

- Precision : 76,9%
- Recall : 37%
- Average : 56,9%

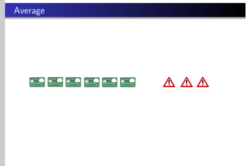
- Precision : 55,7%
- Recall : 83.3%
- Average : 69,5%

Average not OK



Page 56 :

Average



Page 57 :

Average

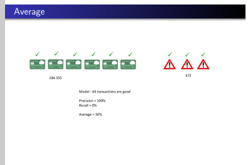
284 335

472

Model : All transactions are good

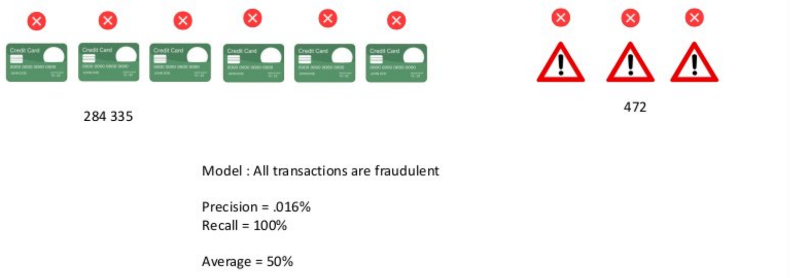
Precision = 100%
Recall = 0%

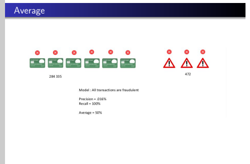
Average = 50%



Page 58 :

Average





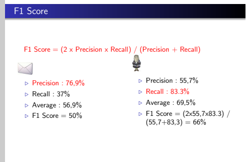
Page 59 :

F1 Score

F1 Score = (2 x Precision x Recall) / (Precision + Recall)

- Precision : 76,9%
- Recall : 37%
- Average : 56,9%
- F1 Score = 50%

- Precision : 55,7%
- Recall : 83.3%
- Average : 69,5%
- F1 Score = (2x55,7x83.3) / (55,7+83,3) = 66%



Page 60 :

F1 Score

```
def f1_score(tp, fp, fn, tn):
  p = precision(tp, fp, fn, tn)
  r = recall(tp, fp, fn, tn)
  return 2 * p * r / (p + r)
```


Page 61 :

Demo : test.py

sklearn : Measuring the quality of a predictive algorithm

```
01: # Import datasets, classifiers and performance metrics
02: from sklearn import datasets, svm, metrics
03: digits = datasets.load_digits()

04: # To apply a classifier on this data, we need to flatten the image, to
05: # turn the data in a (samples, feature) matrix:
06: n_samples = len(digits.images)
07: data = digits.images.reshape((n_samples, -1))

08: # Create a classifier: a support vector classifier
09: classifier = svm.SVC(gamma=0.001)

10: # We learn the digits on the first half of the digits
11: classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])

12: # Now predict the value of the digit on the second half:
13: expected = digits.target[n_samples // 2:]
14: predicted = classifier.predict(data[n_samples // 2:])
15: print("Classification␣report␣for␣classifier␣%s:\n%s\n" \
16:     % (classifier, metrics.classification_report(expected, predicted)))
17: print("Confusion␣matrix:\n%s" % metrics.confusion_matrix(expected, predicted
     ))
```


Page 62 :

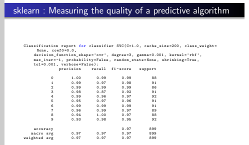
Lines 15 and 16 show the synthesis of the mathematical measurements proposed by sklearn for the 10 classes to be predicted (the 10 possible figures recognized) ; line 17 shows the confusion matrix, that is, a table showing the measures that summarize the quality of the model for these 10 classes.The interest of this table is that it shows very visually the proportion of good predictions. , and the distribution by bad prediction carried out.

sklearn : Measuring the quality of a predictive algorithm

```
Classification report for classifier SVC(C=1.0, cache_size=200, class_weight=
     None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899
```


2019-08-29

IML
└Tests
 └Model performance
  └sklearn : Measuring the quality of a predictive algorithm



Page 63 :

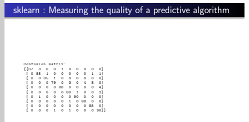
Demo : measure.py

sklearn : Measuring the quality of a predictive algorithm

```
Confusion matrix:
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```


2019-08-29

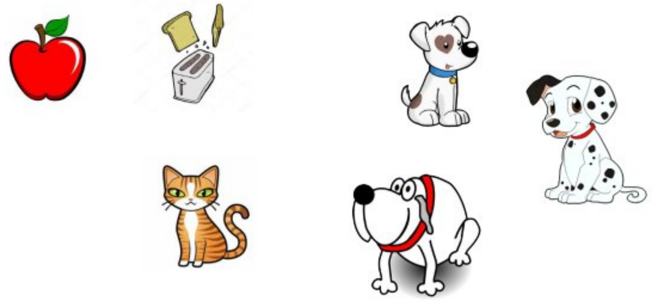
IML
└Tests
 └Model performance
  └sklearn : Measuring the quality of a predictive algorithm



Page 64 :

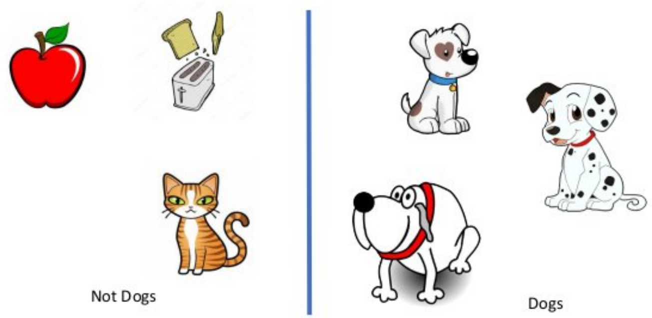
Demo : measure.py

Underfitting/overfitting



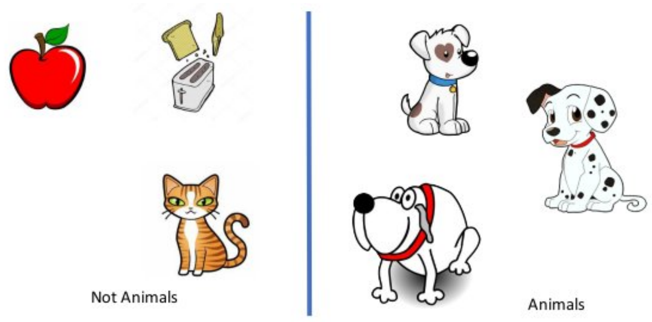
Page 65 :

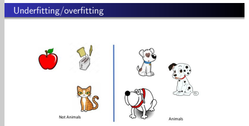
Underfitting/overfitting



Page 66 :

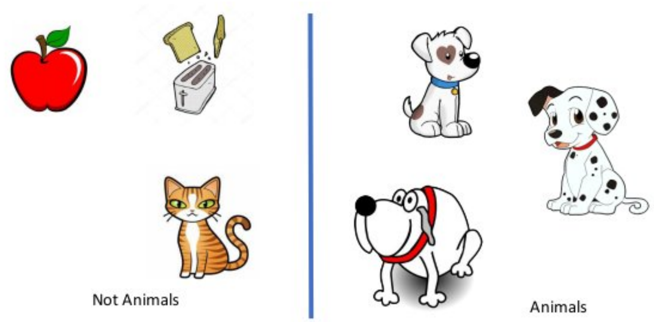
Underfitting/overfitting

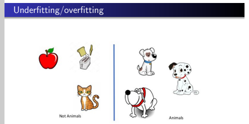




Page 67 :

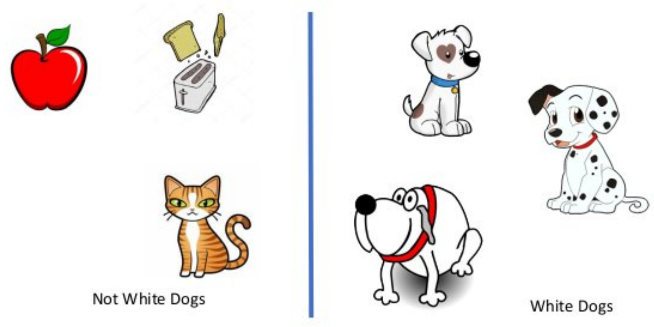
Underfitting/overfitting





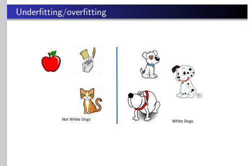
Page 68 :

Underfitting/overfitting



2019-08-29

IML
└Tests
└Model performance
└Underfitting/overfitting



Page 69 :

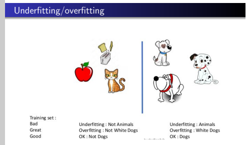
Underfitting/overfitting

2019-08-29

IML
└Tests
└Model performance
└Underfitting/overfitting

Page 70 :

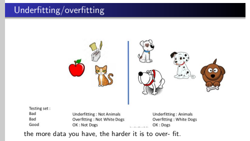
Underfitting/overfitting



Page 71 :

Underfitting/overfitting

the more data you have, the harder it is to over- fit.



Page 72 :

To sum up

- ▷ Define problem (data)
- ▷ List tools (algorithms)
- ▷ Evaluate tools to find the best one
  - ◇ Accuracy
  - ◇ Precision
  - ◇ Recall
  - ◇ F1

Page 73 :

Page 74 :

Features

- As we mentioned, when your data doesn't have enough features, your model is likely to underfit.
- When your data has too many features, it's easy to overfit.
- What are features and where do they come from ?

Features are whatever inputs we provide to our model.

Features
As we mentioned, when your data doesn't have enough features, your model is likely to underfit.
When your data has too many features, it's easy to overfit.
What are features and where do they come from ?
Features are whatever inputs we provide to our model.

Page 75 :

Type of features

Type of features we have constrains the type of models we can use :

- The Naive Bayes classifier is suited to *yes-or-no features*
- Regression models require *numeric features*
- Decision trees can deal with *numeric or categorical data*.

Type of features
Type of features we have constrains the type of models we can use :
The Naive Bayes classifier is suited to yes-or-no features
Regression models require numeric features
Decision trees can deal with numeric or categorical data.

Page 76 :

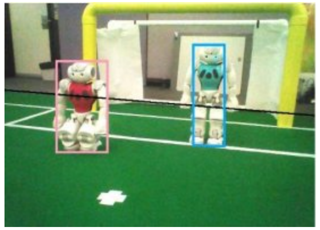
Features extraction

- feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps
- Feature extraction is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and completely describing the original data set

Features extraction
feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps
Feature extraction is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and completely describing the original data set

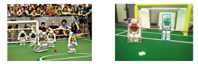
Page 77 :

Example : robot detection



Can we detect robot using low quality images ?

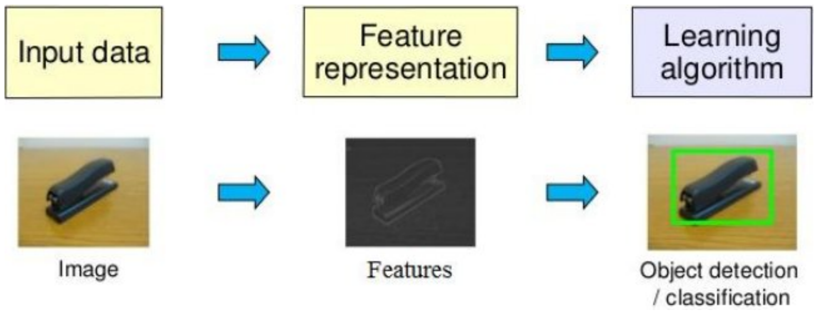
Example : robot detection

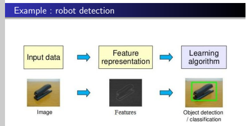


Can we detect robot using low quality images ?

Page 78 :

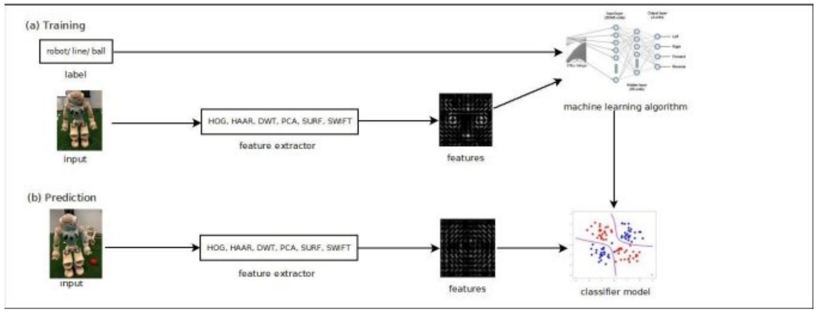
Example : robot detection

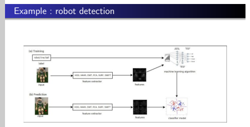




Page 79 :

Example : robot detection

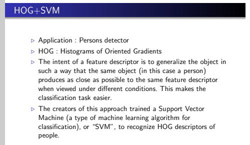




Page 80 :

HOG+SVM

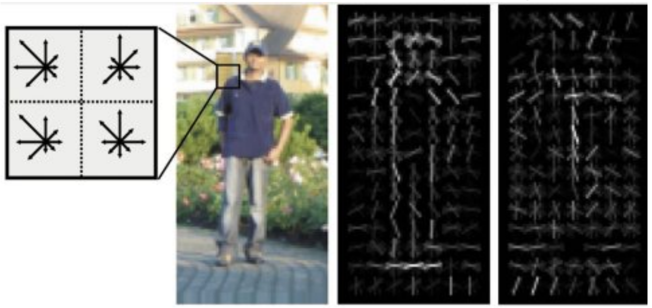
- Application : Persons detector
- HOG : Histograms of Oriented Gradients
- The intent of a feature descriptor is to generalize the object in such a way that the same object (in this case a person) produces as close as possible to the same feature descriptor when viewed under different conditions. This makes the classification task easier.
- The creators of this approach trained a Support Vector Machine (a type of machine learning algorithm for classification), or "SVM", to recognize HOG descriptors of people.



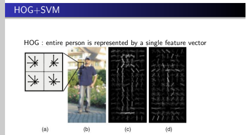
Page 81 :

HOG+SVM

HOG : entire person is represented by a single feature vector



(a) (b) (c) (d)



Page 82 :

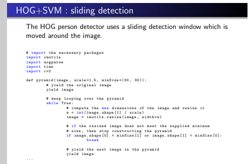
HOG+SVM : sliding detection

The HOG person detector uses a sliding detection window which is moved around the image.

```
# import the necessary packages
import imutils
import argparse
import time
import cv2

def pyramid(image, scale=1.5, minSize=(30, 30)):
        # yield the original image
        yield image

        # keep looping over the pyramid
        while True:
                # compute the new dimensions of the image and resize it
                w = int(image.shape[1] / scale)
                image = imutils.resize(image, width=w)

                # if the resized image does not meet the supplied minimum
                # size, then stop constructing the pyramid
                if image.shape[0] < minSize[1] or image.shape[1] < minSize[0]:
                        break

                # yield the next image in the pyramid
                yield image
...
```


2019-08-29

IML
└Feature extraction
  └Image processing : Object detection and tracking
    └HOG+SVM : sliding detection



Page 83 :

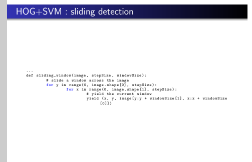
Video : Sliding Window for Object Detection.mp4

HOG+SVM : sliding detection

```
...
def sliding_window(image, stepSize, windowSize):
        # slide a window across the image
        for y in range(0, image.shape[0], stepSize):
                for x in range(0, image.shape[1], stepSize):
                        # yield the current window
                        yield (x, y, image[y:y + windowSize[1], x:x + windowSize
                             [0]])
```


2019-08-29

IML
└Feature extraction
  └Image processing : Object detection and tracking
    └HOG+SVM : sliding detection



Page 84 :

Video : Sliding Window for Object Detection.mp4

HOG+SVM : sliding detection

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True, help="Path␣to␣the␣image")
args = vars(ap.parse_args())

# load the image and define the window width and height
image = cv2.imread(args["image"])
(winW, winH) = (128, 128)

# loop over the image pyramid
for resized in pyramid(image, scale=1.5):
        # loop over the sliding window for each layer of the pyramid
        for (x, y, window) in sliding_window(resized, stepSize=32, windowSize=(
             winW, winH)):
                # if the window does not meet our desired window size, ignore it
                if window.shape[0] != winH or window.shape[1] != winW:
                        continue

                # WHERE APPLY A CLASSIFIER

                # since we do not have a classifier, we will just draw the
                     window
                clone = resized.copy()
                cv2.rectangle(clone, (x, y), (x + winW, y + winH), (0, 255, 0),
                     2)
                cv2.imshow("Window", clone)
                cv2.waitKey(1)
                time.sleep(0.025)
```


Page 85 :

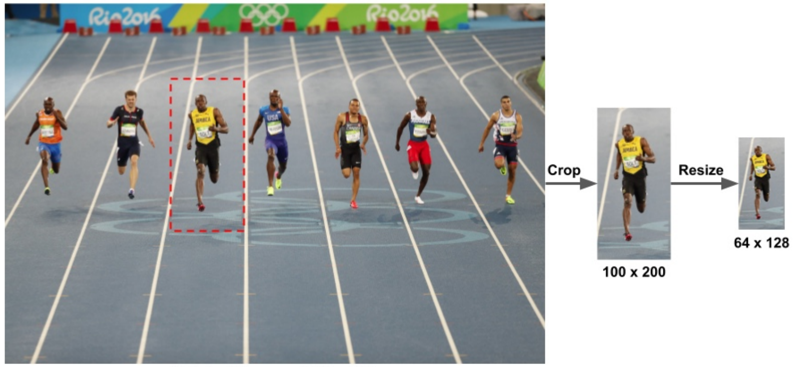
Demo : python3 sliding_window.py –image images/image.jpg

HOG+SVM

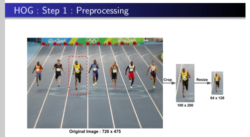
- At each position of the detector window, a HOG descriptor is computed for the detection window.
- This descriptor is then shown to the trained SVM, which classifies it as either "person" or "not a person".
- To recognize persons at different scales, the image is subsampled to multiple sizes. Each of these subsampled images is searched

Page 86 :

HOG : Step 1 : Preprocessing



Original Image : 720 x 475



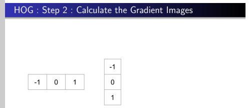
Page 87 :

As mentioned earlier HOG feature descriptor used for pedestrian detection is calculated on a 64x128 patch of an image. Of course, an image may be of any size. Typically patches at multiple scales are analyzed at many image locations. The only constraint is that the patches being analyzed have a fixed aspect ratio. In our case, the patches need to have an aspect ratio of 1 :2. For example, they can be 100x200, 128x256, or 1000x2000 but not 101x205. To illustrate this point I have shown a large image of size 720x475. We have selected a patch of size 100x200 for calculating our HOG feature descriptor. This patch is cropped out of an image and resized to 64x128. Now we are ready to calculate the HOG descriptor for this image patch.

HOG : Step 2 : Calculate the Gradient Images

-1	0	1

-1
0
1



Page 88 :

To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients ; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

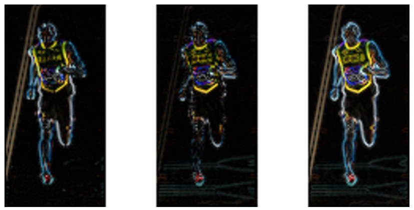
HOG : Step 2 : Calculate the Gradient Images

```
# Read image
im = cv2.imread('bolt.png')
im = np.float32(im) / 255.0

# Calculate gradient
gx = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=1)
gy = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=1)

# Python Calculate gradient magnitude and direction ( in degrees )
mag, angle = cv2.cartToPolar(gx, gy, angleInDegrees=True)
```


Page 89 :

We can also achieve the same results, by using Sobel operator in OpenCV with kernel size 1.

HOG : Step 2 : Calculate the Gradient Images

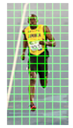


Left : Absolute value of x-gradient.
Center : Absolute value of y-gradient.
Right : Magnitude of gradient.

Page 90 :

Notice, the x-gradient fires on vertical lines and the y-gradient fires on horizontal lines. The magnitude of gradient fires where ever there is a sharp change in intensity. None of them fire when the region is smooth. I have deliberately left out the image showing the direction of gradient because direction shown as an image does not convey much.
The gradient image removed a lot of non-essential information (e.g. constant colored background), but highlighted outlines. In other words, you can look at the gradient image and still easily say there is a person in the picture.
At every pixel, the gradient has a magnitude and a direction. For color images, the gradients of the three channels are evaluated (as shown in the figure above). The magnitude of gradient at a pixel is the maximum of the magnitude of gradients of the three channels, and the angle is the angle corresponding to the maximum gradient.

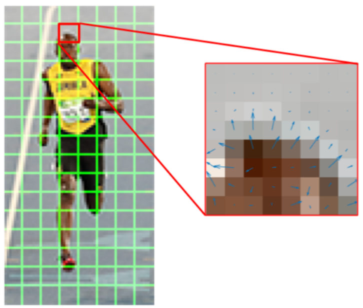
HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



Page 91 :

In this step, the image is divided into 8x8 cells and a histogram of gradients is calculated for each 8x8 cells.
We will learn about the histograms in a moment, but before we go there let us first understand why we have divided the image into 8x8 cells. One of the important reasons to use a feature descriptor to describe a patch of an image is that it provides a compact representation. An 8x8 image patch contains 8x8x3 = 192 pixel values. The gradient of this patch contains 2 values (magnitude and direction) per pixel which adds up to 8x8x2 = 128 numbers. By the end of this section we will see how these 128 numbers are represented using a 9-bin histogram which can be stored as an array of 9 numbers. Not only is the representation more compact, calculating a histogram over a patch makes this represenation more robust to noise. Individual graidents may have noise, but a histogram over 8x8 patch makes the representation much less sensitive to noise.
But why 8x8 patch ? Why not 32x32 ? It is a design choice informed by the scale of features we are looking for. HOG was used for pedestrian detection initially. 8x8 cells in a photo of a pedestrian scaled to 64x128 are big enough to capture interesting features (e.g. the face, the top of the head etc.).
The histogram is essentially a vector (or an array) of 9 bins (numbers) corresponding to angles 0, 20, 40, 60 . . . 160.

HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



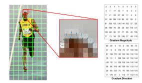
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

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

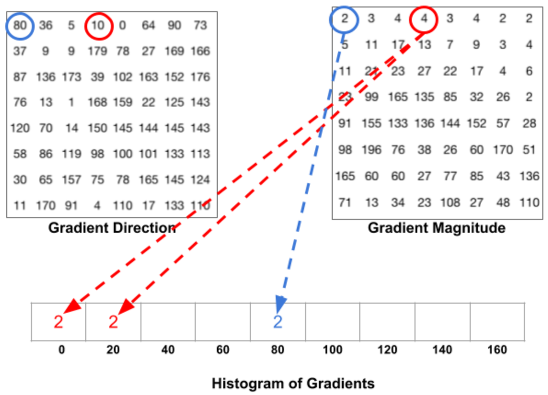
Center : The RGB patch and gradients represented using arrows.
Right : The gradients in the same patch represented as numbers



Page 92 :

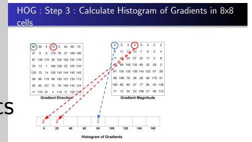
If you are a beginner in computer vision, the image in the center is very informative. It shows the patch of the image overlaid with arrows showing the gradient — the arrow shows the direction of gradient and its length shows the magnitude. Notice how the direction of arrows points to the direction of change in intensity and the magnitude shows how big the difference is.
On the right, we see the raw numbers representing the gradients in the 8x8 cells with one minor difference — the angles are between 0 and 180 degrees instead of 0 to 360 degrees. These are called "unsigned" gradients because a gradient and it's negative are represented by the same numbers. In other words, a gradient arrow and the one 180 degrees opposite to it are considered the same. But, why not use the 0 – 360 degrees ? Empirically it has been shown that unsigned gradients work better than signed gradients for pedestrian detection. Some implementations of HOG will allow you to specify if you want to use signed gradients.
The next step is to create a histogram of gradients in these 8x8 cells. The histogram contains 9 bins corresponding to angles 0, 20, 40 . . . 160.

HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



2019-08-29

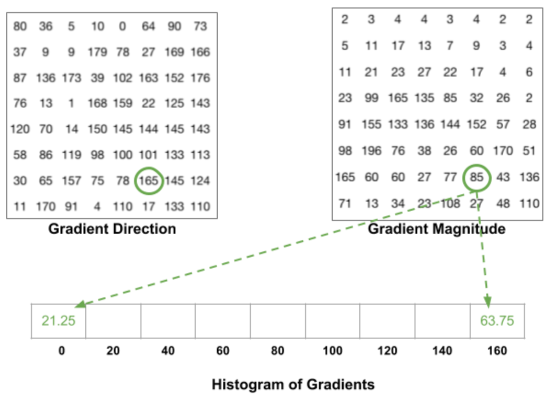
IML
Feature extraction
Image processing : Object detection and tracking
HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



Page 93 :

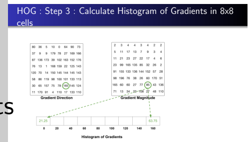
The following figure illustrates the process. We are looking at magnitude and direction of the gradient of the same 8x8 patch as in the previous figure. A bin is selected based on the direction, and the vote (the value that goes into the bin) is selected based on the magnitude. Let's first focus on the pixel encircled in blue. It has an angle (direction) of 80 degrees and magnitude of 2. So it adds 2 to the 5th bin. The gradient at the pixel encircled using red has an angle of 10 degrees and magnitude of 4. Since 10 degrees is half way between 0 and 20, the vote by the pixel splits evenly into the two bins.

HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



2019-08-29

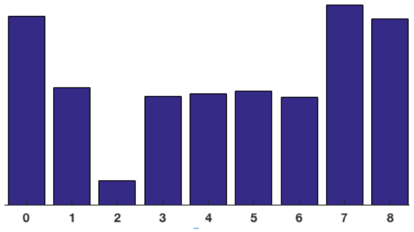
IML
Feature extraction
Image processing : Object detection and tracking
HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells

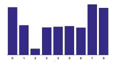


Page 94 :

There is one more detail to be aware of. If the angle is greater than 160 degrees, it is between 160 and 180, and we know the angle wraps around making 0 and 180 equivalent. So in the example below, the pixel with angle 165 degrees contributes proportionally to the 0 degree bin and the 160 degree bin.

HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells

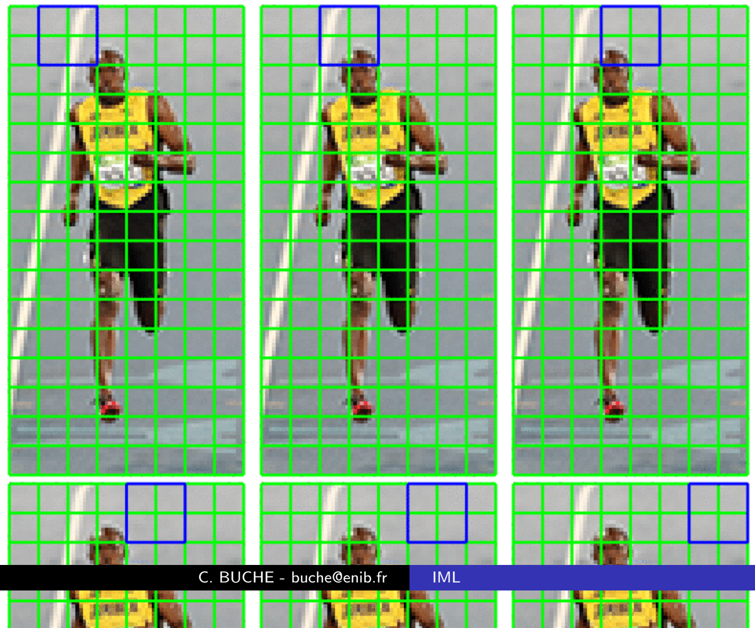


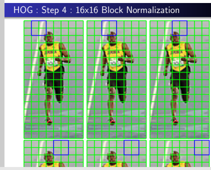


Page 95 :

The contributions of all the pixels in the 8x8 cells are added up to create the 9-bin histogram. For the patch above, it looks like this.
In our representation, the y-axis is 0 degrees. You can see the histogram has a lot of weight near 0 and 180 degrees, which is just another way of saying that in the patch gradients are pointing either up or down.

HOG : Step 4 : 16x16 Block Normalization





Page 96 :

In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half. Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations.
Before I explain how the histogram is normalized, let's see how a vector of length 3 is normalized.
Let's say we have an RGB color vector [128, 64, 32]. The length of this vector is $\sqrt{128^2 + 64^2 + 32^2} = 146.64$. This is also called the L2 norm of the vector. Dividing each element of this vector by 146.64 gives us a normalized vector [0.87, 0.43, 0.22]. Now consider another vector in which the elements are twice the value of the first vector 2 x [128, 64, 32] = [256, 128, 64]. You can work it out yourself to see that normalizing [256, 128, 64] will result in [0.87, 0.43, 0.22], which is the same as the normalized version of the original RGB vector. You can see that normalizing a vector removes the scale.
Now that we know how to normalize a vector, you may be tempted to think that while calculating HOG you can simply normalize the 9x1 histogram the same way we normalized the 3x1 vector above. It is not a bad idea, but a better idea is to normalize over a bigger sized block of 16x16. A 16x16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3x1 vector is normalized. The window is then moved by 8 pixels (see animation) and a normalized 36x1 vector is calculated over this window and the process is repeated.

HOG : Step 5 : Calculate the HOG feature vector

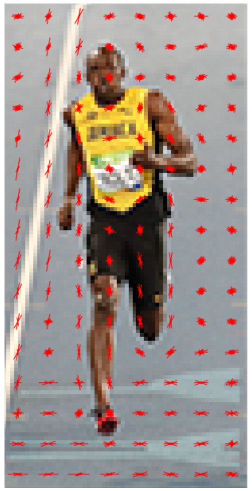
To calculate the final feature vector for the entire image patch, the 36x1 vectors are concatenated into one giant vector. What is the size of this vector ? Let us calculate

1. How many positions of the 16x16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
2. Each 16x16 block is represented by a 36x1 vector. So when we concatenate them all into one gaint vector we obtain a 36x105 = 3780 dimensional vector.

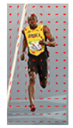
HOG : Step 5 : Calculate the HOG feature vector
To calculate the final feature vector for the entire image patch, the 36x1 vectors are concatenated into one giant vector. What is the size of this vector ? Let us calculate
1. How many positions of the 16x16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
2. Each 16x16 block is represented by a 36x1 vector. So when we concatenate them all into one gaint vector we obtain a 36x105 = 3780 dimensional vector.

Page 97 :

HOG : Visualizing Histogram of Oriented Gradients



HOG : Visualizing Histogram of Oriented Gradients



Page 98 :

The HOG descriptor of an image patch is usually visualized by plotting the 9x1 normalized histograms in the 8x8 cells. See image on the side. You will notice that dominant direction of the histogram captures the shape of the person, especially around the torso and legs.

Frame differencing

```python
import cv2

# Compute the frame differences
def frame_diff(prev_frame, cur_frame, next_frame):
    # Difference between the current frame and the next frame
    diff_frames_1 = cv2.absdiff(next_frame, cur_frame)

    # Difference between the current frame and the previous frame
    diff_frames_2 = cv2.absdiff(cur_frame, prev_frame)

    return cv2.bitwise_and(diff_frames_1, diff_frames_2)
```


Page 99 :

Demo : frame_diff.py
Note : sudo chmod 777 /dev/video0

Tests
Feature extraction
Feature
Feature extraction
Image processing : Object detection and tracking

Frame differencing

```python
# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    return gray
```


Page 100 :

Frame differencing

```
if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the scaling factor for the images
    scaling_factor = 0.5

    # Grab the current frame
    prev_frame = get_frame(cap, scaling_factor)

    # Grab the next frame
    cur_frame = get_frame(cap, scaling_factor)

    # Grab the frame after that
    next_frame = get_frame(cap, scaling_factor)
```


Page 101 :

Frame differencing

```
if __name__=='__main__':
 ....

    # Keep reading the frames from the webcam
    # until the user hits the 'Esc' key
    while True:
        # Display the frame difference
        cv2.imshow('Object␣Movement', frame_diff(prev_frame,
                cur_frame, next_frame))

        # Update the variables
        prev_frame = cur_frame
        cur_frame = next_frame

        # Grab the next frame
        next_frame = get_frame(cap, scaling_factor)

        # Check if the user hit the 'Esc' key
        key = cv2.waitKey(10)
        if key == 27:
            break

    # Close all the windows
cv2.destroyAllWindows()
```


Page 102 :

Background substraction

```python
import cv2
import numpy as np

# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    return frame
```


Page 103 :

Demo : background_sub.py

Background substraction

```python
if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the background subtractor object
    bg_subtractor = cv2.createBackgroundSubtractorMOG2()

    # Define the number of previous frames to use to learn.
    # This factor controls the learning rate of the algorithm.
    # The learning rate refers to the rate at which your model
    # will learn about the background. Higher value for
    # "history" indicates a slower learning rate. You can
    # play with this parameter to see how it affects the output.
    history = 100

    # Define the learning rate
    learning_rate = 1.0/history
```


Page 104 :

Background substraction

```python
if __name__=='__main__':
    .....

     # Keep reading the frames from the webcam
     # until the user hits the 'Esc' key
     while True:
         # Grab the current frame
         frame = get_frame(cap, 0.5)

         # Compute the mask
         mask = bg_subtractor.apply(frame, learningRate=learning_rate)

         # Convert grayscale image to RGB color image
         mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

         # Display the images
         cv2.imshow('Input', frame)
         cv2.imshow('Output', mask & frame)

         # Check if the user hit the 'Esc' key
         c = cv2.waitKey(10)
         if c == 27:
             break

     # Release the video capture object
     cap.release()

     # Close all the windows
```


2019-08-29

IML
└Feature extraction
 └Image processing : Object detection and tracking
  └Background substraction

Page 105 :

CAMShift

- Color space tracker : define the color first.
- CAMShift : consider a region of interest
- Select the region

2019-08-29

IML
└Feature extraction
 └Image processing : Object detection and tracking
  └CAMShift

CAMShift
- Color space tracker : define the color first.
- CAMShift : consider a region of interest
- Select the region

Page 106 :

Demo : colorspacing.py
Demo : camshift.py

CAMShift

```
import cv2
import numpy as np

# Define a class to handle object tracking related functionality
class ObjectTracker(object):
    def __init__(self, scaling_factor=0.5):
        # Initialize the video capture object
        self.cap = cv2.VideoCapture(0)
        # Capture the frame from the webcam
        _, self.frame = self.cap.read()
        # Scaling factor for the captured frame
        self.scaling_factor = scaling_factor
        # Resize the frame
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)
        # Create a window to display the frame
        cv2.namedWindow('Object␣Tracker')
        # Set the mouse callback function to track the mouse
        cv2.setMouseCallback('Object␣Tracker', self.mouse_event)
        # Initialize variable related to rectangular region selection
        self.selection = None
        # Initialize variable related to starting position
        self.drag_start = None
        # Initialize variable related to the state of tracking
        self.tracking_state = 0
```


Page 107 :

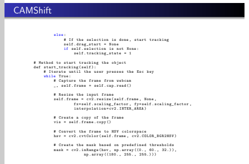
CAMShift

```
    # Define a method to track the mouse events
    def mouse_event(self, event, x, y, flags, param):
        # Convert x and y coordinates into 16-bit numpy integers
        x, y = np.int16([x, y])

        # Check if a mouse button down event has occurred
        if event == cv2.EVENT_LBUTTONDOWN:
            self.drag_start = (x, y)
            self.tracking_state = 0

        # Check if the user has started selecting the region
        if self.drag_start:
            if flags & cv2.EVENT_FLAG_LBUTTON:
                # Extract the dimensions of the frame
                h, w = self.frame.shape[:2]
                # Get the initial position
                xi, yi = self.drag_start
                # Get the max and min values
                x0, y0 = np.maximum(0, np.minimum([xi, yi], [x, y]))
                x1, y1 = np.minimum([w, h], np.maximum([xi, yi], [x, y]))
                # Reset the selection variable
                self.selection = None
                # Finalize the rectangular selection
                if x1-x0 > 0 and y1-y0 > 0:
                    self.selection = (x0, y0, x1, y1)
```


Page 108 :

CAMShift

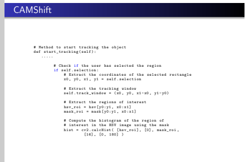
```
        else:
            # If the selection is done, start tracking
            self.drag_start = None
            if self.selection is not None:
                self.tracking_state = 1

# Method to start tracking the object
def start_tracking(self):
    # Iterate until the user presses the Esc key
    while True:
        # Capture the frame from webcam
        _, self.frame = self.cap.read()

        # Resize the input frame
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)

        # Create a copy of the frame
        vis = self.frame.copy()

        # Convert the frame to HSV colorspace
        hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)

        # Create the mask based on predefined thresholds
        mask = cv2.inRange(hsv, np.array((0., 60., 32.)),
                    np.array((180., 255., 255.)))
```

Page 109 :

CAMShift

```
# Method to start tracking the object
def start_tracking(self):
   .....

        # Check if the user has selected the region
        if self.selection:
            # Extract the coordinates of the selected rectangle
            x0, y0, x1, y1 = self.selection

            # Extract the tracking window
            self.track_window = (x0, y0, x1-x0, y1-y0)

            # Extract the regions of interest
            hsv_roi = hsv[y0:y1, x0:x1]
            mask_roi = mask[y0:y1, x0:x1]

            # Compute the histogram of the region of
            # interest in the HSV image using the mask
            hist = cv2.calcHist( [hsv_roi], [0], mask_roi,
                    [16], [0, 180] )
```

Page 110 :

CAMShift

```
# Method to start tracking the object
def start_tracking(self):
   .....

            # Normalize and reshape the histogram
            cv2.normalize(hist, hist, 0, 255, cv2.NORM_MINMAX);
            self.hist = hist.reshape(-1)

            # Extract the region of interest from the frame
            vis_roi = vis[y0:y1, x0:x1]

            # Compute the image negative (for display only)
            cv2.bitwise_not(vis_roi, vis_roi)
            vis[mask == 0] = 0
```


Page 111 :

CAMShift

```
# Method to start tracking the object
def start_tracking(self):
  ....
        # Check if the system in the "tracking" mode
        if self.tracking_state == 1:
            # Reset the selection variable
            self.selection = None
            # Compute the histogram back projection
            hsv_backproj = cv2.calcBackProject([hsv], [0],
                    self.hist, [0, 180], 1)
            # Compute bitwise AND between histogram
            # backprojection and the mask
            hsv_backproj &= mask
            # Define termination criteria for the tracker
            term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,
                    10, 1)
```


Page 112 :

CAMShift

```
    # Method to start tracking the object
    def start_tracking(self):
      ....
                # Apply CAMShift on 'hsv_backproj'
                track_box, self.track_window = cv2.CamShift(hsv_backproj,
                        self.track_window, term_crit)
                # Draw an ellipse around the object
                cv2.ellipse(vis, track_box, (0, 255, 0), 2)

            # Show the output live video
            cv2.imshow('Object␣Tracker', vis)
            # Stop if the user hits the 'Esc' key
            c = cv2.waitKey(5)
            if c == 27:
                break

        # Close all the windows
        cv2.destroyAllWindows()

if __name__ == '__main__':
    ObjectTracker().start_tracking()
```


IML
└Feature extraction
 └Image processing : Object detection and tracking
  └CAMShift

2019-08-29

Page 113 :

Tests & Features

IML

Cédric Buche

ENIB

29 août 2019

IML
└Feature extraction
 └Image processing : Object detection and tracking

2019-08-29

Page 114 :