Detection & Tests

IML

Cédric Buche

ENIB

30 août 2018

Detection & Tests
IML
Cédric Buche
ENIB
30 août 2018

Page 1 :

1 Classification
- KNN

2 Clustering
- k-means
- Hierarchical clustering
- Distance

3 Tests
- Training vs Testing
- K-Fold Cross Validation
- Model performance

4 Feature extraction

1 Classification
- KNN
2 Clustering
- k-means
- Hierarchical clustering
- Distance
3 Tests
- Training vs Testing
- K-Fold Cross Validation
- Model performance
4 Feature extraction

Page 2 :

1 Classification
- KNN

2 Clustering
- k-means
- Hierarchical clustering
- Distance

3 Tests
- Training vs Testing
- K-Fold Cross Validation
- Model performance

4 Feature extraction

2018-08-30

Page 3 :

Model : KNN

Examples

- predict how I'm going to vote in the next presidential election. If you know nothing else about me, one approach is to look at how my neighbors are planning to vote. Living in Seattle, my neighbors are planning to vote for the Democratic candidate, which suggests that "Democratic candidate" is a good guess for me as well.
- you know more about me : my age, my income, how many kids I have ... To the extent my behavior is influenced by those things, looking just at my neighbors who are close to me among all those dimensions seems likely to be an even better predictor than looking at all my neighbors. This is the idea behind *nearest neighbors classification*.

2018-08-30

Page 4 :

Model : KNN

Requirements

- ▷ Some notion of distance
- ▷ An assumption that points that are close to one another are similar

the prediction for each new point depends only on the handful of points closest to it.

Page 5 :

Model : KNN

- ▷ classify some new data point : find the k nearest labeled points and let them vote on the new output.
- ▷ need a function that counts votes : Reduce k until we find a unique winner.

```
def majority_vote(labels):
  """assumes␣that␣labels␣are␣ordered␣from␣nearest␣to␣farthest"""
  vote_counts = Counter(labels)
  winner, winner_count = vote_counts.most_common(1)[0]
  num_winners = len([count for count in vote_counts.values() if count ==
        winner_count])
  if num_winners == 1:
      return winner # unique winner, so return it
  else:
      return majority_vote(labels[:-1]) # try again without the farthest
```

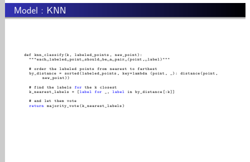

Page 6 :

Model : KNN

```
def knn_classify(k, labeled_points, new_point):
  """each␣labeled␣point␣should␣be␣a␣pair␣(point,␣label)"""

  # order the labeled points from nearest to farthest
  by_distance = sorted(labeled_points, key=lambda (point, _): distance(point,
       new_point))

  # find the labels for the k closest
  k_nearest_labels = [label for _, label in by_distance[:k]]

  # and let them vote
  return majority_vote(k_nearest_labels)
```


2018-08-30

IML
└Classification
  └KNN
    └Model : KNN



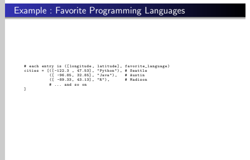
Page 7 :

Example : Favorite Programming Languages

```
# each entry is ([longitude, latitude], favorite_language)
cities = [([-122.3 , 47.53], "Python"), # Seattle
          ([ -96.85, 32.85], "Java"),   # Austin
          ([ -89.33, 43.13], "R"),      # Madison
          # ... and so on
]
```


2018-08-30

IML
└Classification
  └KNN
    └Example : Favorite Programming Languages

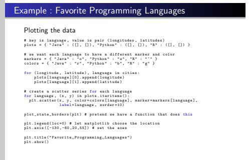


Page 8 :

Example : Favorite Programming Languages

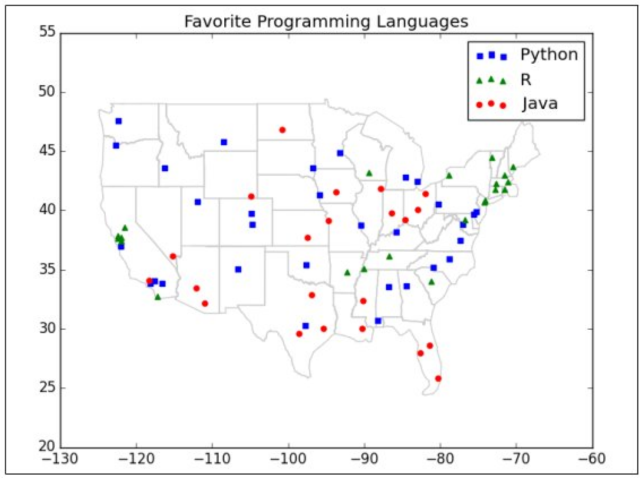
Plotting the data

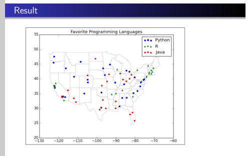
```
# key is language, value is pair (longitudes, latitudes)
plots = { "Java" : ([], []), "Python" : ([], []), "R" : ([], []) }

# we want each language to have a different marker and color
markers = { "Java" : "o", "Python" : "s", "R" : "^" }
colors = { "Java" : "r", "Python" : "b", "R" : "g" }

for (longitude, latitude), language in cities:
    plots[language][0].append(longitude)
    plots[language][1].append(latitude)

# create a scatter series for each language
for language, (x, y) in plots.iteritems():
  plt.scatter(x, y, color=colors[language], marker=markers[language],
              label=language, zorder=10)

plot_state_borders(plt) # pretend we have a function that does this

plt.legend(loc=0) # let matplotlib choose the location
plt.axis([-130,-60,20,55]) # set the axes

plt.title("Favorite␣Programming␣Languages")
plt.show()
```

Page 9 :

Result





Page 10 :

Example : Favorite Programming Languages

Try several different values for k

```
for k in [1, 3, 5, 7]:
    num_correct = 0
for city in cities:
    location, actual_language = city
    other_cities = [other_city
                    for other_city in cities
                    if other_city != city]

    predicted_language = knn_classify(k, other_cities, location)

    if predicted_language == actual_language:
        num_correct += 1

    print k, "neighbor[s]:", num_correct, "correct␣out␣of", len(cities)
```


Page 11 :

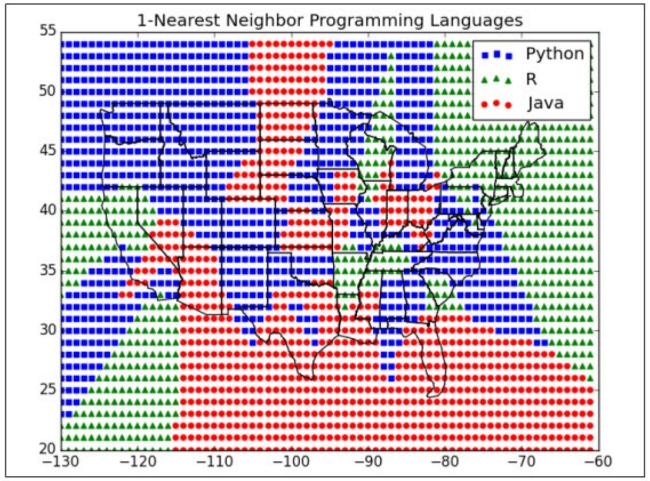
1 neighbor[s] : 40 correct out of 75 3 neighbor[s] : 44 correct out of 75 5 neighbor[s] : 41 correct out of 75 7 neighbor[s] : 35 correct out of 75

Example : Favorite Programming Languages

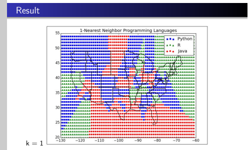
```
plots = { "Java" : ([], []), "Python" : ([], []), "R" : ([], []) }

k = 1 # or 3, or 5, or ...

for longitude in range(-130, -60):
  for latitude in range(20, 55):
    predicted_language = knn_classify(k, cities, [longitude, latitude])
    plots[predicted_language][0].append(longitude)
    plots[predicted_language][1].append(latitude)
```


Page 12 :

Result

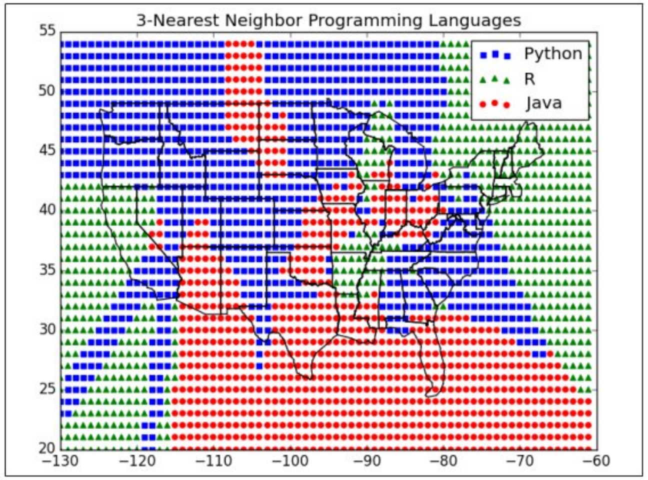


k = 1

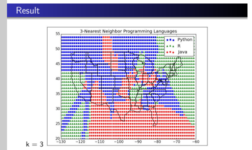


Page 13 :

Result



k = 3



Page 14 :

1. Classification
   - KNN
2. Clustering
   - k-means
   - Hierarchical clustering
   - Distance
3. Tests
   - Training vs Testing
   - K-Fold Cross Validation
   - Model performance
4. Feature extraction

2018-08-30

IML
└─Clustering

Page 15 :

Unsupervised learning

Learning mode

- ▷ supervised learning : set of labeled data for making predictions about new, unlabeled data.
- ▷ unsupervised learning : no label at all
- ▷ Whenever you look at some source of data, the data will somehow form *clusters*.

2018-08-30

IML
└─Clustering
  └─Unsupervised learning

Page 16 :

Idea

Examples

- A data set showing where millionaires live probably has clusters in places like Beverly Hills and Manhattan.
- A data set showing how many hours people work each week probably has a cluster around 40.
- A data set of demographics of registered voters likely forms a variety of clusters (e.g., "soccer moms", "bored retirees" ...)

the clusters won't label themselves. You'll have to do that by looking at the data underlying each one.

Page 17 :

Model : k-means

1. Start with a set of k-means, which are points in d-dimensional space.
2. Assign each point to the mean to which it is closest.
3. If no point's assignment has changed, stop and keep the clusters.
4. If some point's assignment has changed, recompute the means and return to step 2.

Page 18 :

Model : k-means

```
def vector_mean(vectors):
  "compute␣the␣vector␣whose␣ith␣element␣is␣the␣mean␣of␣the␣ith␣elements␣of␣the␣
       input␣vectors"
  n = len(vectors)
  return scalar_multiply(1/n, vector_sum(vectors))

class KMeans:
  """performs␣k-means␣clustering"""

  def __init__(self, k):
    self.k = k        # number of clusters
    self.means = None # means of clusters

  def classify(self, input):
  """return␣the␣index␣of␣the␣cluster␣closest␣to␣the␣input"""
  return min(range(self.k),
    key=lambda i: squared_distance(input, self.means[i]))
```


Page 19 :

Model : k-means

```
  def train(self, inputs):
    # choose k random points as the initial means
    self.means = random.sample(inputs, self.k)
    assignments = None

  while True:
    # Find new assignments
    new_assignments = map(self.classify, inputs)

  # If no assignments have changed, we're␣done.
␣␣if␣assignments␣==␣new_assignments:
␣␣␣␣return

␣␣#␣Otherwise␣keep␣the␣new␣assignments,
␣␣assignments␣=␣new_assignments

␣␣#␣And␣compute␣new␣means␣based␣on␣the␣new␣assignments
␣␣for␣i␣in␣range(self.k):
␣␣␣␣#␣find␣all␣the␣points␣assigned␣to␣cluster␣i
␣␣␣␣i_points␣=␣[p␣for␣p,␣a␣in␣zip(inputs,␣assignments)␣if␣a␣==␣i]

␣␣␣␣#␣make␣sure␣i_points␣is␣not␣empty␣so␣don't divide by 0
    if i_points:
      self.means[i] = vector_mean(i_points)
```


Page 20 :

Example : stickers

Context

- sticker printer can print at most five colors per sticker.
- there's some way to take a design and modify it so that it only contains five colors ?

Data

- images can be represented as two-dimensional array of pixels, where each pixel is itself a three-dimensional vector (red, green, blue) indicating its color.
- five-color version of the image
  1. Choosing five colors
  2. Assigning one of those colors to each pixel

Page 21 :

Example : stickers

```
path_to_png_file = r"C:\images\image.png"
import matplotlib.image as mpimg
img = mpimg.imread(path_to_png_file)

top_row = img[0]
top_left_pixel = top_row[0]
red, green, blue = top_left_pixel

pixels = [pixel for row in img for pixel in row]

clusterer = KMeans(5)
clusterer.train(pixels)

def recolor(pixel):
  cluster = clusterer.classify(pixel)
  return clusterer.means[cluster]

new_img = [[recolor(pixel) for pixel in row]
           for row in img]

plt.imshow(new_img)
plt.axis('off')
plt.show()
```


Page 22 :

Alternative approach

"grow" clusters from the bottom up

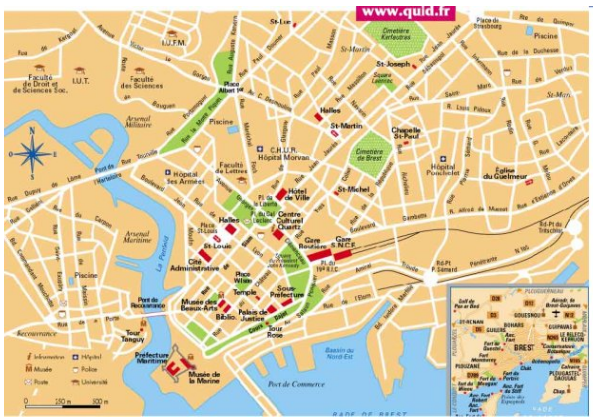
1. Make each input its own cluster of one.
2. As long as there are multiple clusters remaining, find the two closest clusters and merge them.
3. At the end, we'll have one giant cluster containing all the inputs. If we keep track of the merge order, we can recreate any number of clusters by unmerging. For example, if we want three clusters, we can just undo the last two merges.

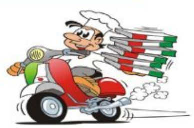
Alternative approach

"grow" clusters from the bottom up
1. Make each input its own cluster of one.
2. As long as there are multiple clusters remaining, find the two closest clusters and merge them.
3. At the end, we'll have one giant cluster containing all the inputs. If we keep track of the merge order, we can recreate any number of clusters by unmerging. For example, if we want three clusters, we can just undo the last two merges.

Page 23 :

Example : pizza

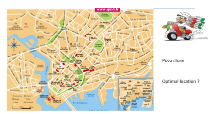




Pizza chain

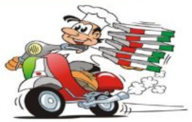
Optimal location ?

Example : pizza



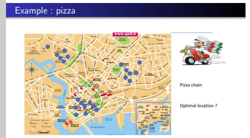
Page 24 :

Example : pizza



Pizza chain

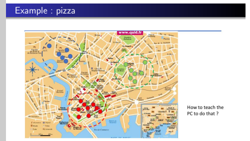
Optimal location ?



Page 25 :

Example : pizza

How to teach the PC to do that ?



Page 26 :

Example : pizza

Page 27 :

Example : pizza

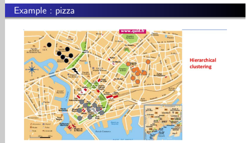
Page 28 :

Example : pizza

Page 29 :

Example : pizza

Hierarchical clustering



Page 30 :

Distance

Name	Egg-laying	Scales	Poisonous	Cold-blooded	Legs nb	Reptile
Cobra	True	True	True	True	0	Yes
Rattlesnake	True	True	True	True	0	Yes
Boa	False	False	False	True	0	Yes
Chicken	True	False	False	False	2	No
Alligator	True	False	False	True	4	Yes
Frog	True	True	True	True	4	No
Salmon	True	False	False	True	0	No
Python	True	False	False	True	0	Yes

Features = four binary and one integer
Boa = (0,1,0,1,0)
Frog =(1,0,1,1,4)
Distance to separate ?

Page 31 :

Distance : Euclidean

	rattlesnake	boa	frog
rattlesnake		1.4	4.2
boa	1.4		4.4
frog	4.2	4.4	

Page 32 :

Distance : Euclidean

	rattlesnake	boa	frog	Alligator
rattlesnake		1.4	4.2	4.1
boa	1.4		4.4	4.1
frog	4.2	4.4		1.7
Alligator	4.1	4.1	1.7	

Alligator is closer to a frog than a snake

Page 33 :

Distance : Euclidean

	rattlesnake	boa	frog	Alligator
rattlesnake		1.4	1.7	1.4
boa	1.4		2.2	1.4
frog	1.7	2.2		1.7
Alligator	1.4	1.4	1.7	

Using binary Feature : Alligator is closer to a snake than a frog
Feature Engineering Matters

Page 34 :

1 Classification
- KNN

2 Clustering
- k-means
- Hierarchical clustering
- Distance

3 Tests
- Training vs Testing
- K-Fold Cross Validation
- Model performance

4 Feature extraction

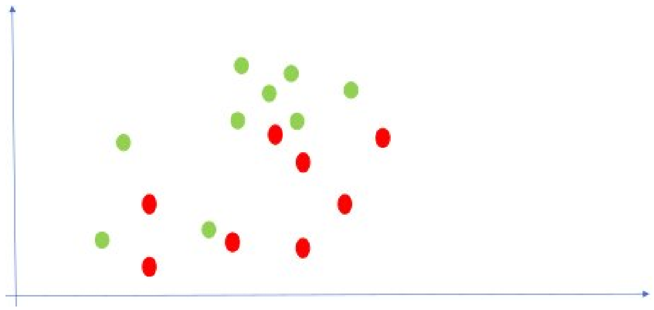
Page 35 :

Testing

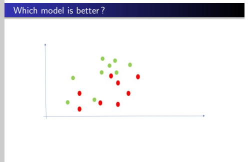
- ▷ How well is my model doing ?
- ▷ How do I improve it ?

Page 36 :

Which model is better ?

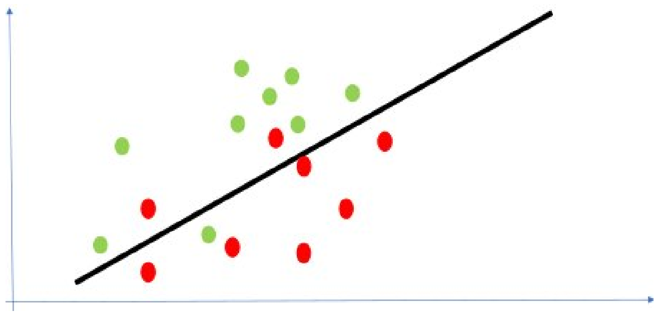


└─Which model is better ?

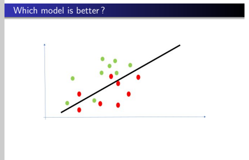


Page 37 :

Which model is better ?

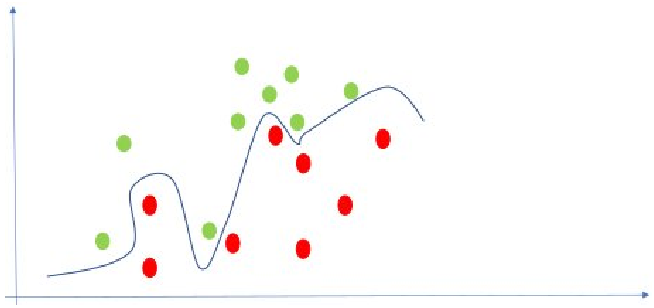


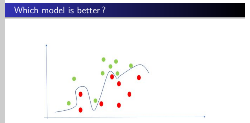
└─Which model is better ?



Page 38 :

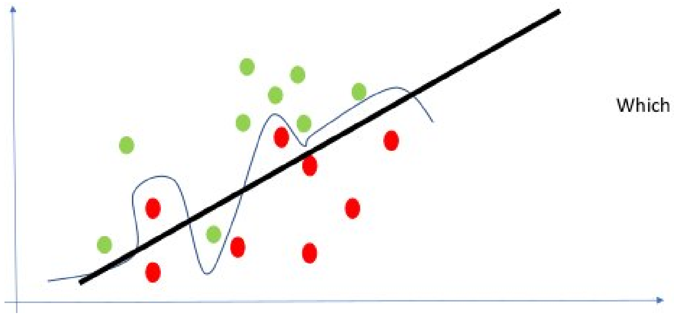
Which model is better ?



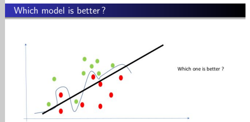


Page 39 :

Which model is better ?

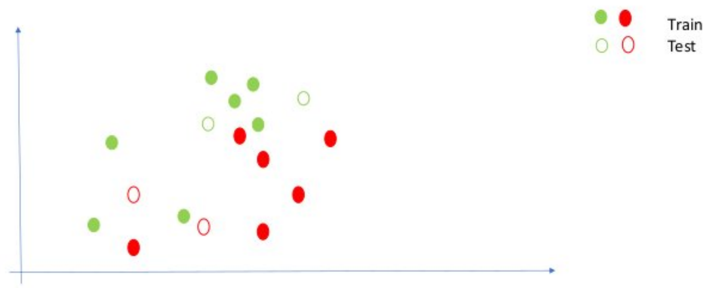


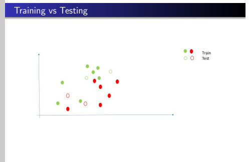
Which one is better ?



Page 40 :

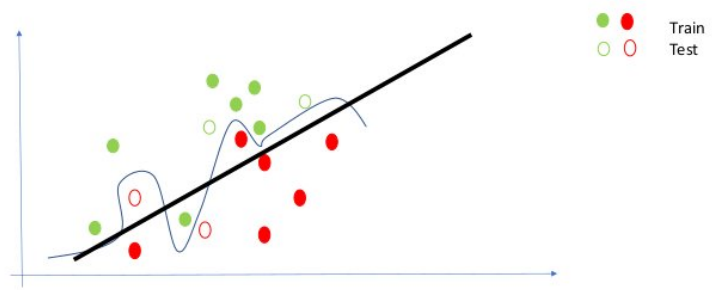
Training vs Testing

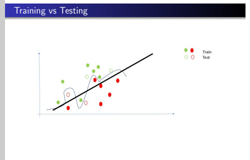




Page 41 :

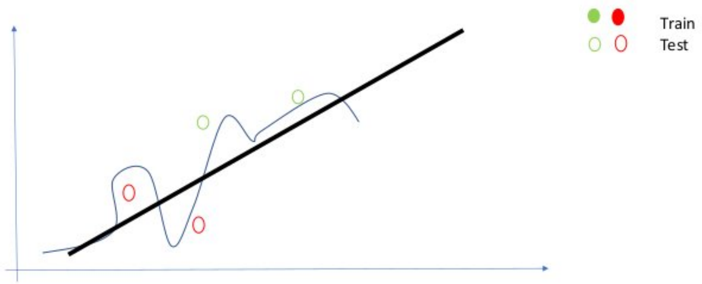
Training vs Testing





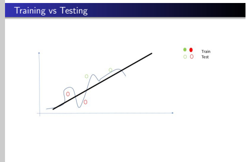
Page 42 :

Training vs Testing



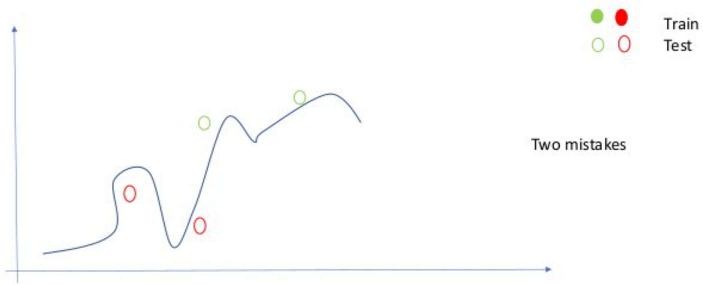
2018-08-30

IML
└Tests
 └Training vs Testing
  └Training vs Testing



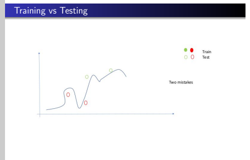
Page 43 :

Training vs Testing



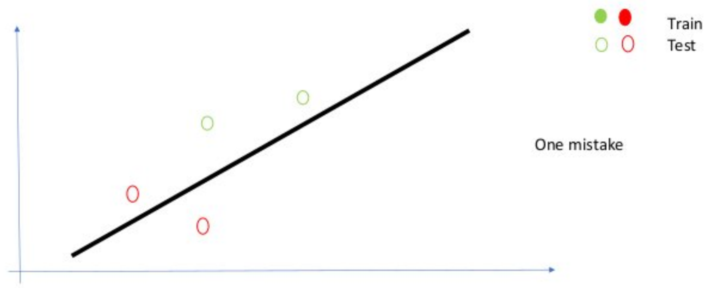
2018-08-30

IML
└Tests
 └Training vs Testing
  └Training vs Testing



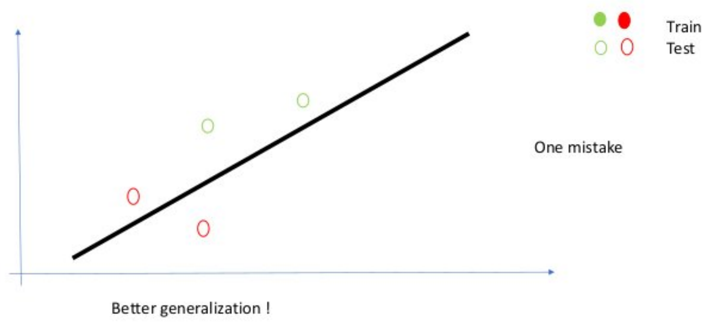
Page 44 :

Training vs Testing



Page 45 :

Training vs Testing



Page 46 :

Learning Rule

- NEVER use your testing data for training

Page 47 :

Training vs Testing

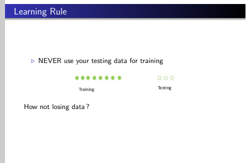
```
def split_data(data, prob):
  results = [], []
  for row in data:
    results[0 if random.random() < prob else 1].append(row)
  return results

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 48 :

Learning Rule

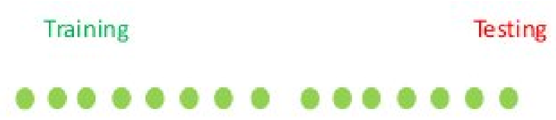
▷ NEVER use your testing data for training

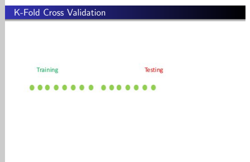
How not losing data ?



Page 49 :

K-Fold Cross Validation



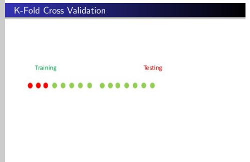


Page 50 :

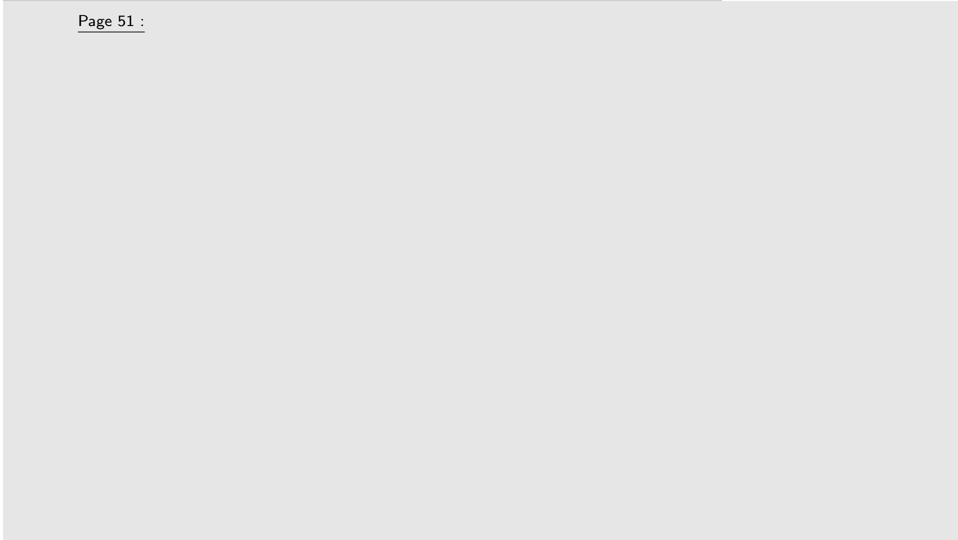
K-Fold Cross Validation

2018-08-30

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



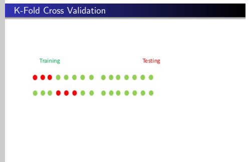
Page 51 :



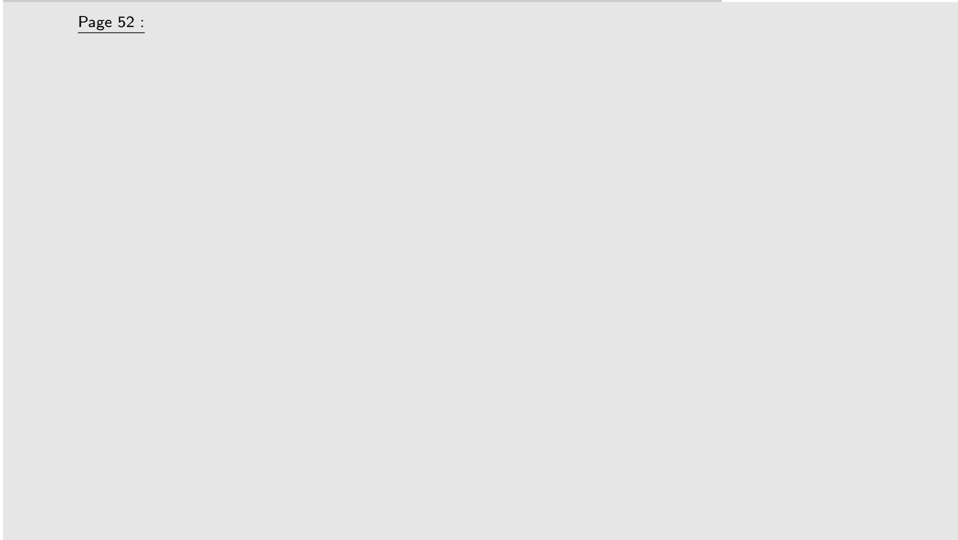
K-Fold Cross Validation

2018-08-30

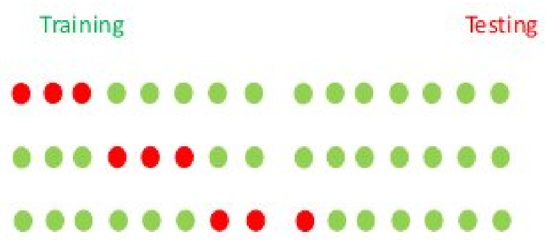
IML
└Tests
└K-Fold Cross Validation
└K-Fold Cross Validation

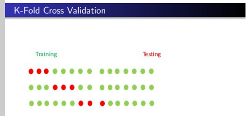


Page 52 :



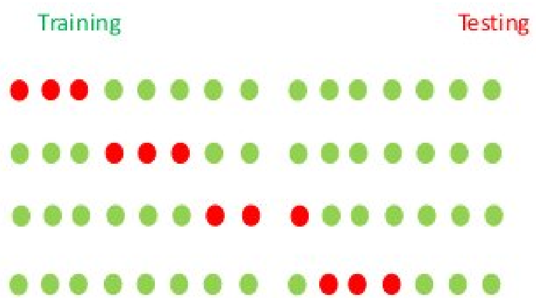
K-Fold Cross Validation

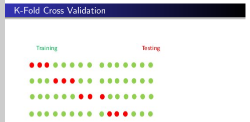




Page 53 :

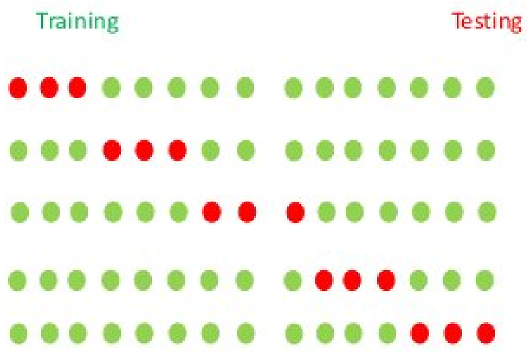
K-Fold Cross Validation

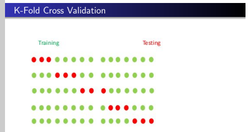




Page 54 :

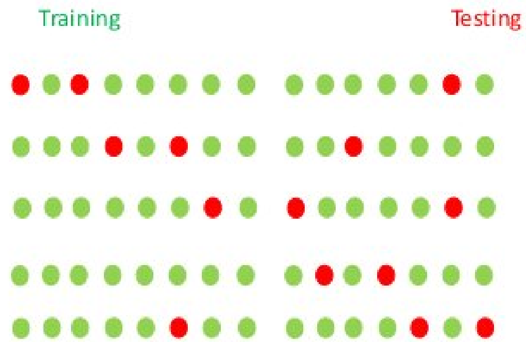
K-Fold Cross Validation

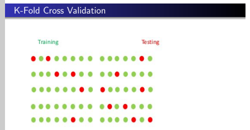




Page 55 :

K-Fold Cross Validation





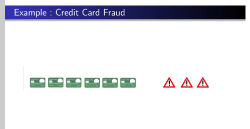
Page 56 :

- How well is my model doing ?
  - Tricky question

How well is my model doing ?
Tricky question

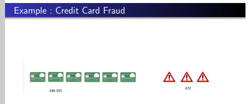
Page 57 :

Example : Credit Card Fraud



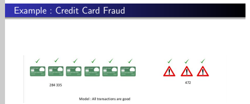
Page 58 :

Example : Credit Card Fraud



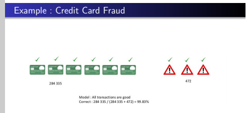
Page 59 :

Example : Credit Card Fraud



Page 60 :

Example : Credit Card Fraud

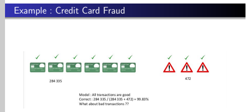


Page 61 :

Classification
Clustering
Tests
Feature extraction

Training vs Testing
K-Fold Cross Validation
Model performance

Example : Credit Card Fraud

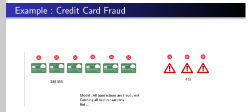


Page 62 :

Example : Credit Card Fraud

2018-08-30

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

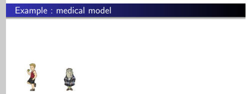


Page 63 :

Example : medical model

2018-08-30

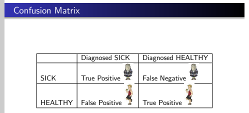
IML
└Tests
└Model performance
└Example : medical model



Page 64 :

Confusion Matrix

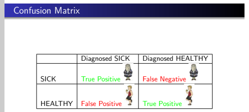
	Diagnosed SICK	Diagnosed HEALTHY
SICK	True Positive 	False Negative 
HEALTHY	False Positive 	True Positive 



Page 65 :

Confusion Matrix

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



Page 66 :

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 67 :

Example : spam model

Example : spam model

Page 68 :

Confusion Matrix

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

Confusion Matrix

Page 69 :

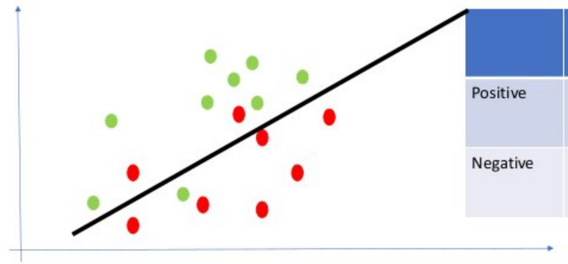
Confusion Matrix

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

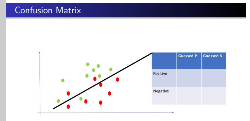
Confusion Matrix

Page 70 :

Confusion Matrix

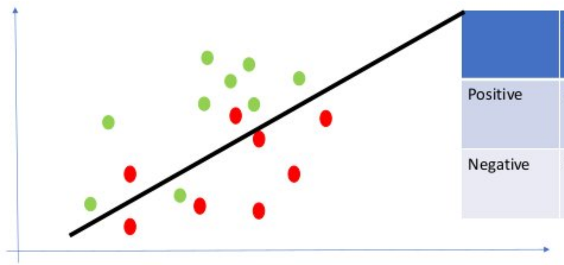


	Guessed P	Guessed N
Positive		
Negative		

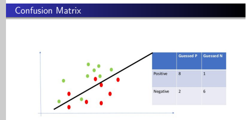


Page 71 :

Confusion Matrix



	Guessed P	Guessed N
Positive	8	1
Negative	2	6



Page 72 :

Accuracy

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Page 73 :

Accuracy

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

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

Page 74 :

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

Page 75 :

Accuracy

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


Page 76 :

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK		False Negative 
HEALTHY	False Positive 	

Page 77 :

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK		False Negative 
HEALTHY	False Positive 	

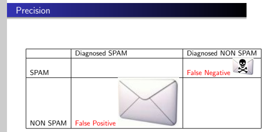
Page 78 :

Precision

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

2018-08-30

IML
└Tests
└Model performance
└Precision



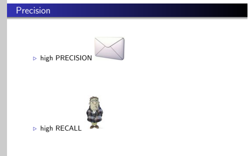
Page 79 :

Precision

- high PRECISION 
- high RECALL 

2018-08-30

IML
└Tests
└Model performance
└Precision



Page 80 :

Precision

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

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

Precision
How many did we classify correctly ?
Precision = 1000/ (1000+800) =55,7%

Page 81 :

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)

Precision
How many did we classify correctly ?
Precision= 76.8%
Precision = True Positives/ (True Positives + False Positives)

Page 82 :

Precision

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


Page 83 :

Recall

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

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

Page 84 :

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 85 :

Recall

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


Page 86 :

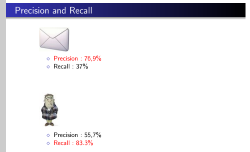
Precision and Recall

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

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

2018-08-30

IML
└Tests
└Model performance
└Precision and Recall



Page 87 :

Average

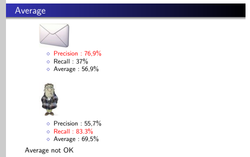
- Precision : 76,9%
- Recall : 37%
- Average : 56,9%

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

Average not OK

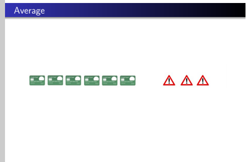
2018-08-30

IML
└Tests
└Model performance
└Average



Page 88 :

Average



Page 89 :

Average

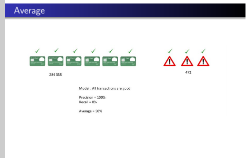
284 335

472

Model : All transactions are good

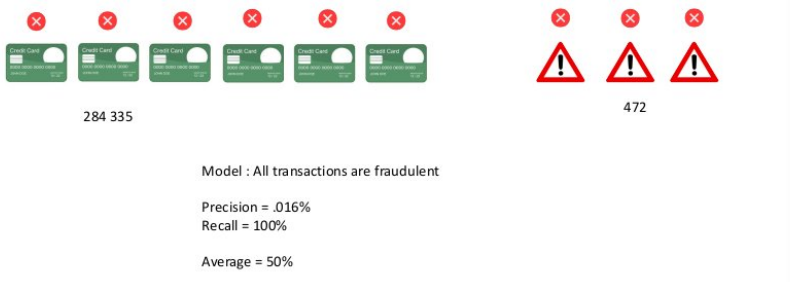
Precision = 100%
Recall = 0%

Average = 50%



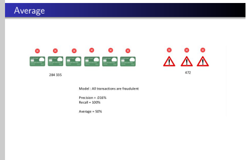
Page 90 :

Average



2018-08-30

IML
└Tests
└Model performance
└Average



Page 91 :

F1 Score

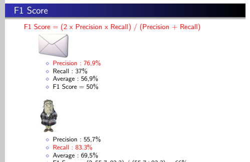
F1 Score = (2 x Precision x Recall) / (Precision + Recall)

- 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%

2018-08-30

IML
└Tests
└Model performance
└F1 Score



Page 92 :

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)
```


F1 Score

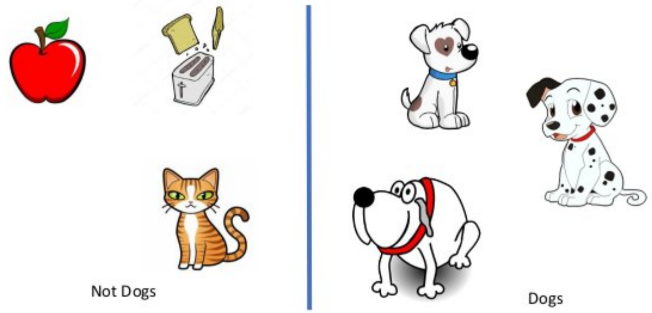
Page 93 :

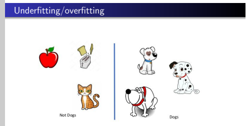
Underfitting/overfitting

Underfitting/overfitting

Page 94 :

Underfitting/overfitting

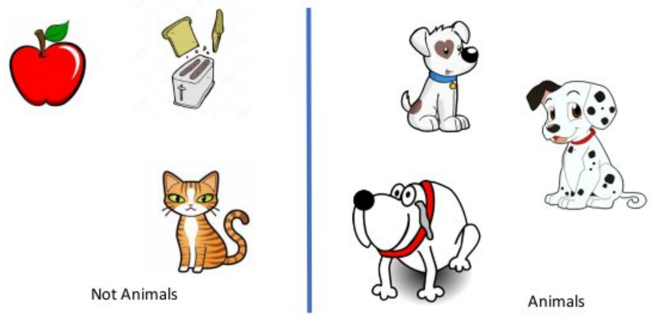




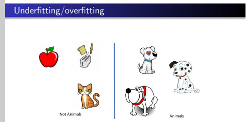
Page 95 :

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

Underfitting/overfitting

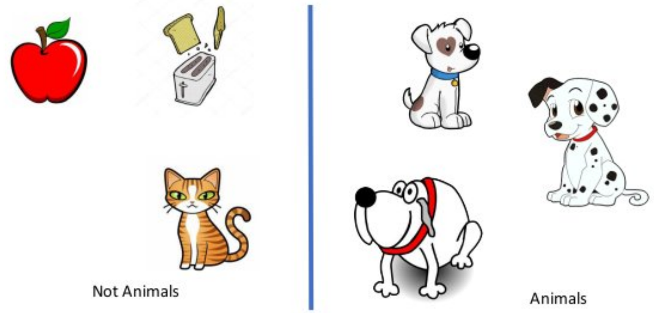


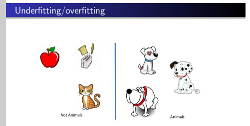
2018-08-30
IML
└Tests
└Model performance
└Underfitting/overfitting



Page 96 :

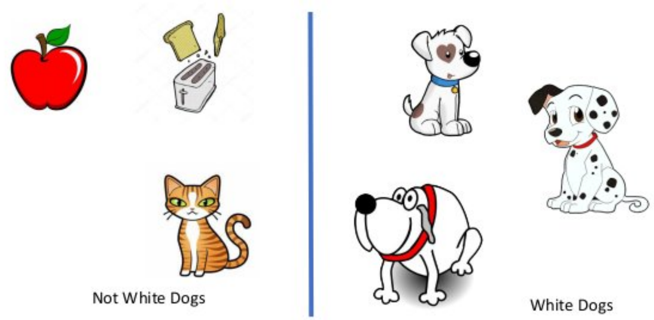
Underfitting/overfitting





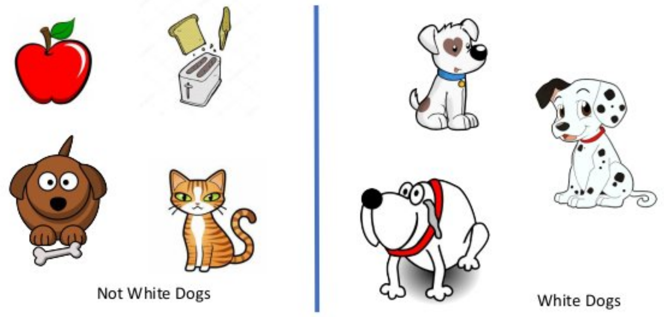
Page 97 :

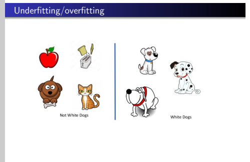
Underfitting/overfitting



Page 98 :

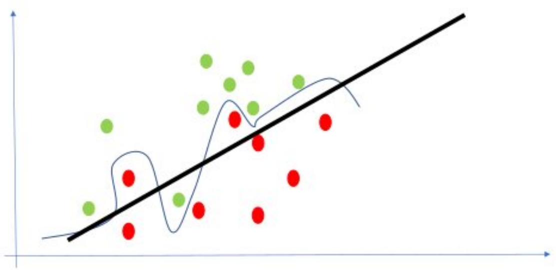
Underfitting/overfitting

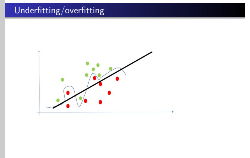




Page 99 :

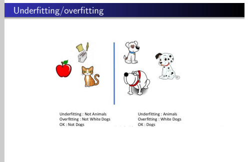
Underfitting/overfitting





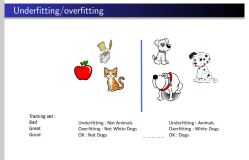
Page 100 :

Underfitting/overfitting



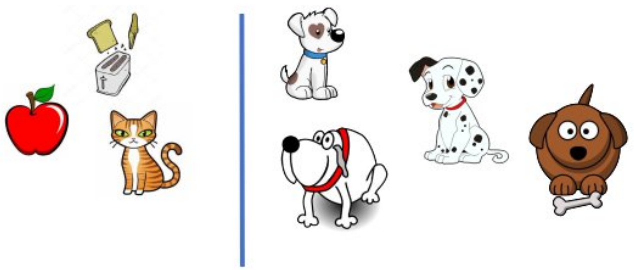
Page 101 :

Underfitting/overfitting



Page 102 :

Underfitting/overfitting



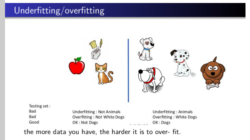
Testing set :

Testing set		
Bad	Underfitting : Not Animals	Underfitting : Animals
Bad	Overfitting : Not White Dogs	Overfitting : White Dogs
Good	OK : Not Dogs	OK : Dogs

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

2018-08-30

IML
└Tests
└Model performance
└Underfitting/overfitting



Page 103 :

1 Classification
- KNN

2 Clustering
- k-means
- Hierarchical clustering
- Distance

3 Tests
- Training vs Testing
- K-Fold Cross Validation
- Model performance

4 Feature extraction

2018-08-30

IML
└Feature extraction

Page 104 :

Features

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

Page 105 :

Type of features

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

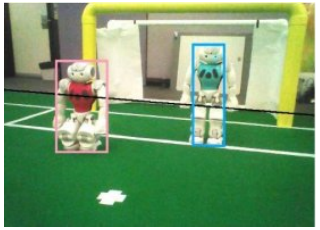
- ▷ The Naive Bayes classifier is suited to yes-or-no features
- ▷ Regression models require numeric features
- ▷ Decision trees can deal with numeric or categorical data.

Type of features

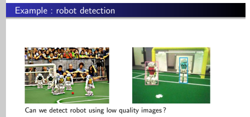
Type of features we have constrains the type of models we can use :
▷ The Naive Bayes classifier is suited to yes-or-no features
▷ Regression models require numeric features
▷ Decision trees can deal with numeric or categorical data.

Page 106 :

Example : robot detection

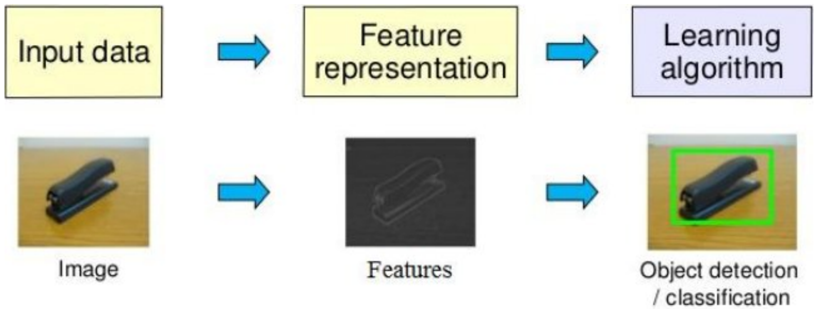


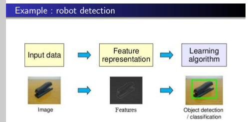
Can we detect robot using low quality images ?



Page 107 :

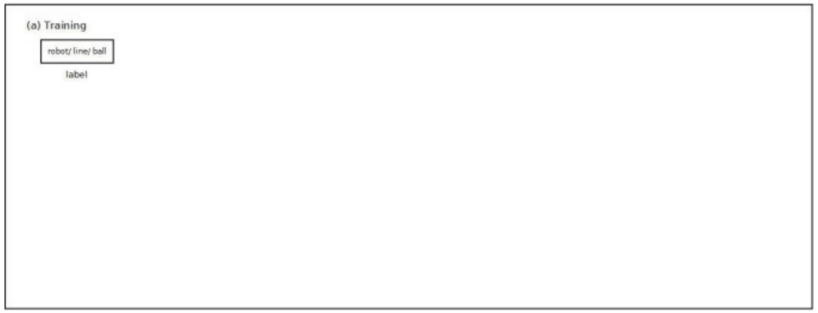
Example : robot detection





Page 108 :

Example : robot detection



Example : robot detection

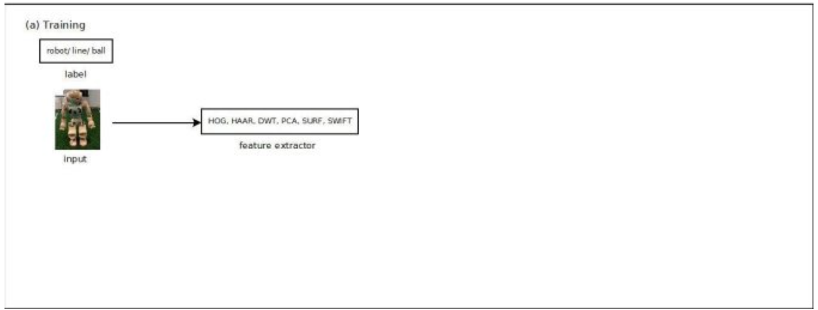
Page 109 :

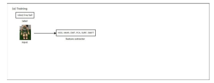
Example : robot detection

Example : robot detection

Page 110 :

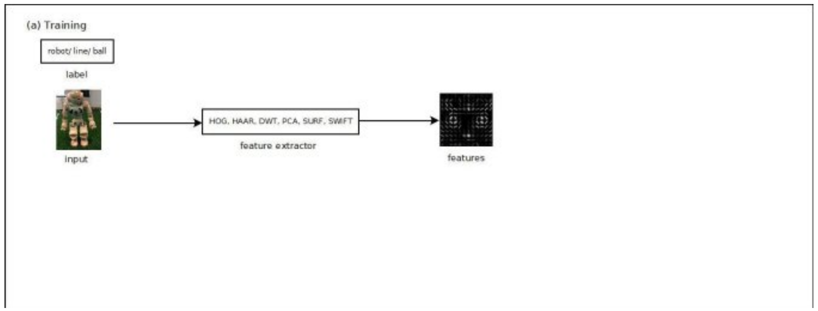
Example : robot detection

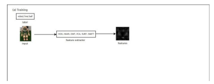




Page 111 :

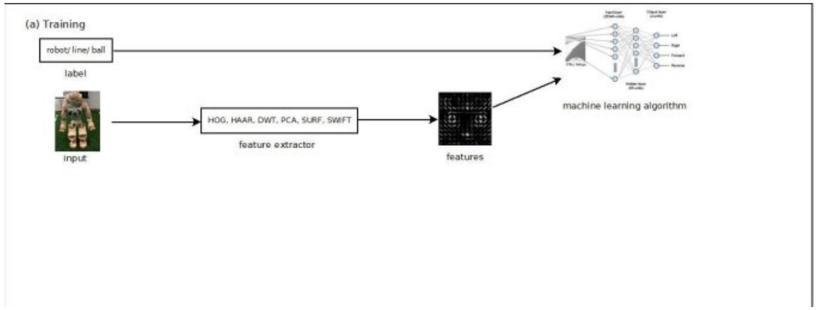
Example : robot detection

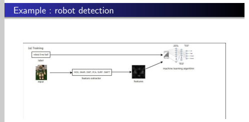




Page 112 :

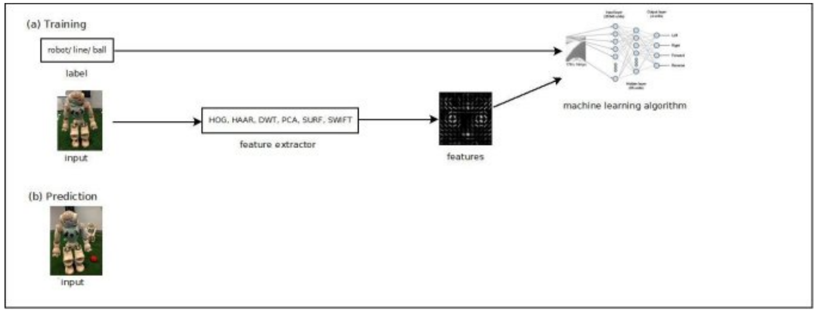
Example : robot detection

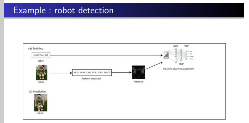




Page 113 :

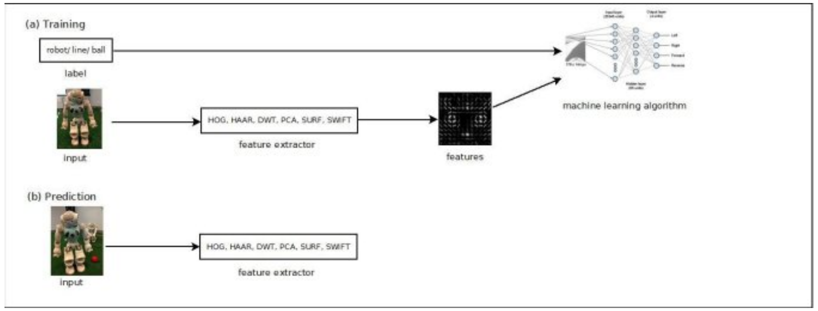
Example : robot detection

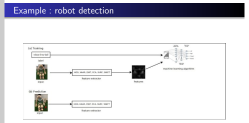




Page 114 :

Example : robot detection

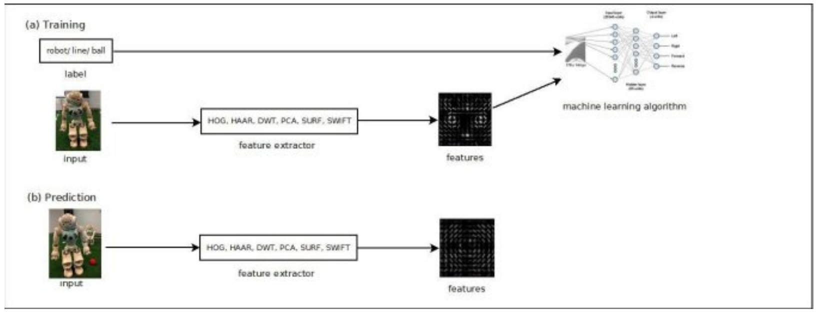


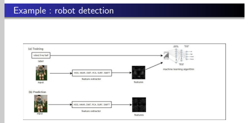


Page 115 :

Classification
Clustering
Tests
Feature extraction

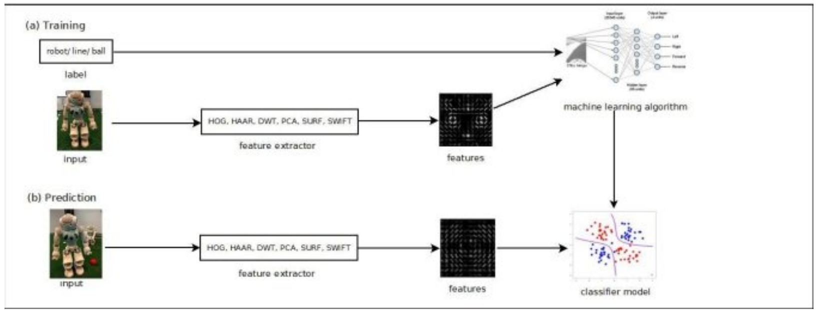
Example : robot detection

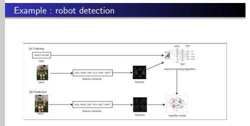




Page 116 :

Example : robot detection

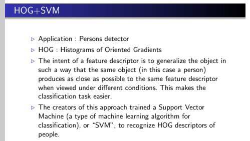




Page 117 :

HOG+SVM

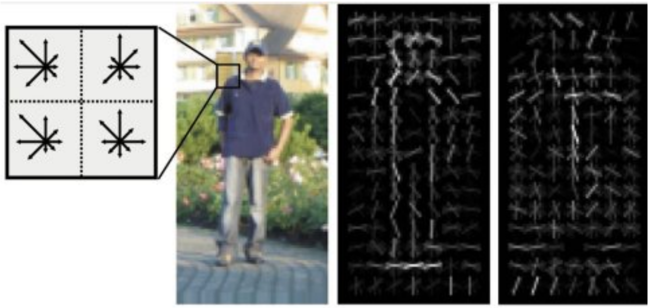
- ▷ Application : Persons detector
- ▷ HOG : Histograms of Oriented Gradients
- ▷ The intent of a feature descriptor is to generalize the object in such a way that the same object (in this case a person) produces as close as possible to the same feature descriptor when viewed under different conditions. 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 118 :

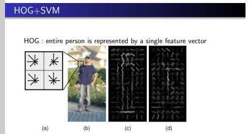
HOG+SVM

HOG : entire person is represented by a single feature vector



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

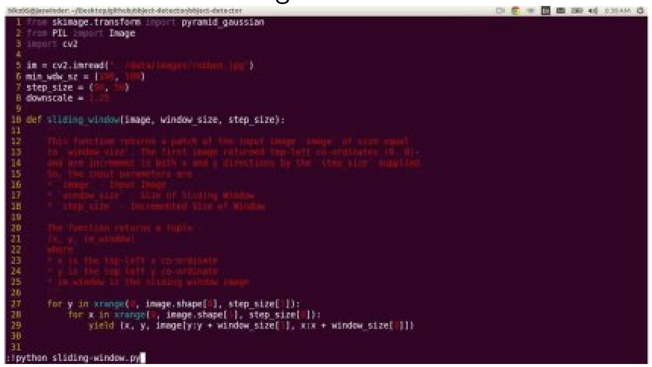
2018-08-30



Page 119 :

HOG+SVM

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



2018-08-30

Page 120 :

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

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 121 :

HOG+SVM

data

HOG+SVM

data

Page 122 :

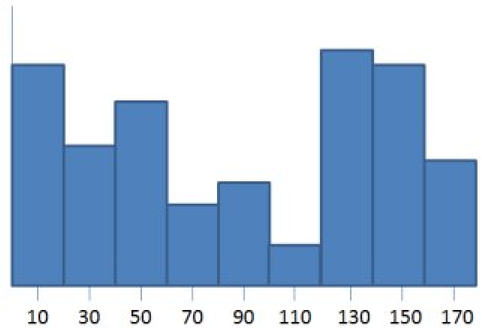
HOG+SVM

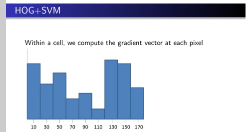
8x8 pixel cells within the detection window

Page 123 :

HOG+SVM

Within a cell, we compute the gradient vector at each pixel





Page 124 :

To sum up

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

2018-08-30

Page 125 :

Detection & Tests

IML

Cédric Buche

ENIB

30 août 2018

2018-08-30

Page 126 :