Introduction

IML

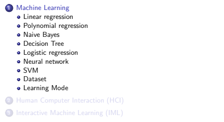
Cédric Buche

ENIB

6 juillet 2018

Page 1 :

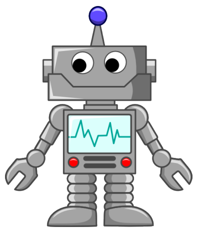
Page 2 :



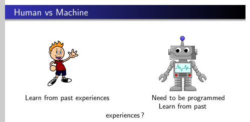
Page 3 :

Human vs Machine

Learn from past experiences

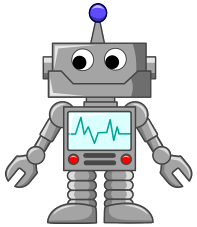


Need to be programmed
Learn from past experiences ?



Page 4 :

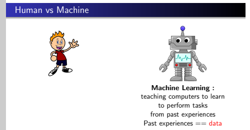
Human vs Machine



Machine Learning :
teaching computers to learn
to perform tasks
from past experiences
Past experiences == data

2018-07-06

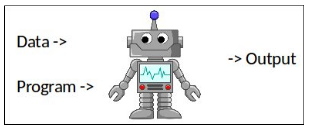
IML
└Machine Learning
└Human vs Machine



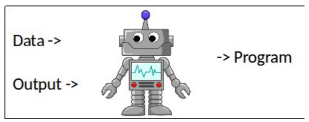
Page 5 :

Machine Learning : What is it ?

Traditional Programming

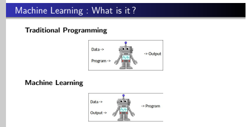


Machine Learning



2018-07-06

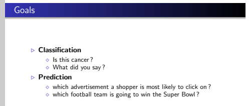
IML
└Machine Learning
└Machine Learning : What is it ?



Page 6 :

Goals

- **Classification**
  - Is this cancer ?
  - What did you say ?
- **Prediction**
  - which advertisement a shopper is most likely to click on ?
  - which football team is going to win the Super Bowl ?



Page 7 :

Machine Learning
Human Computer Interaction (HCI)
Interactive Machine Learning (IML)
Linear regression
Polynomial regression
Naive Bayes
Decision Tree
Logistic regression
Neural network
SVM
Dataset
Learning Mode

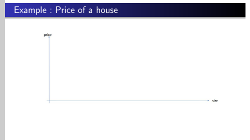
$20, 000

$300, 000

?

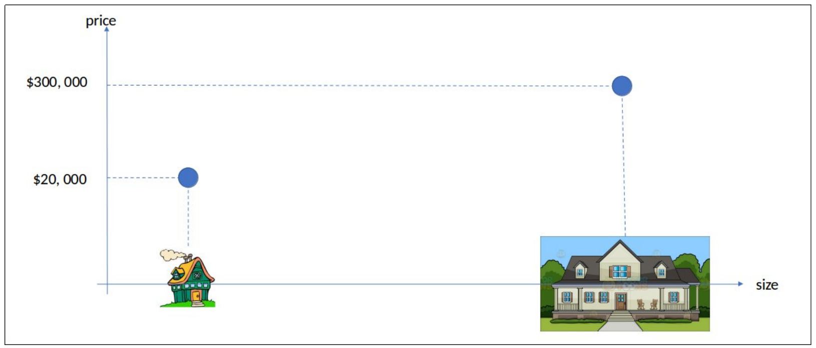
Page 8 :

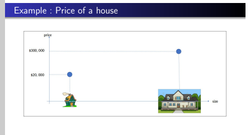
Example : Price of a house



Page 9 :

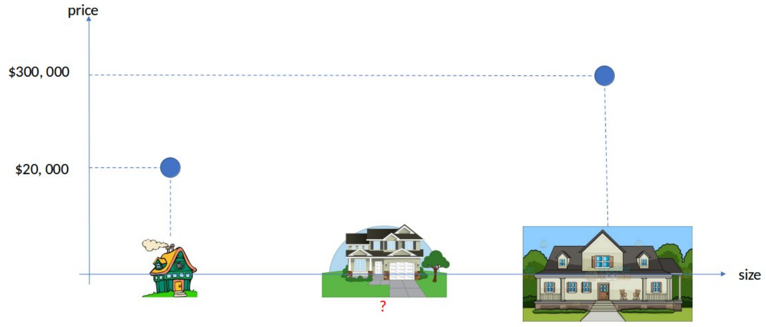
Example : Price of a house

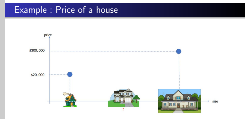




Page 10 :

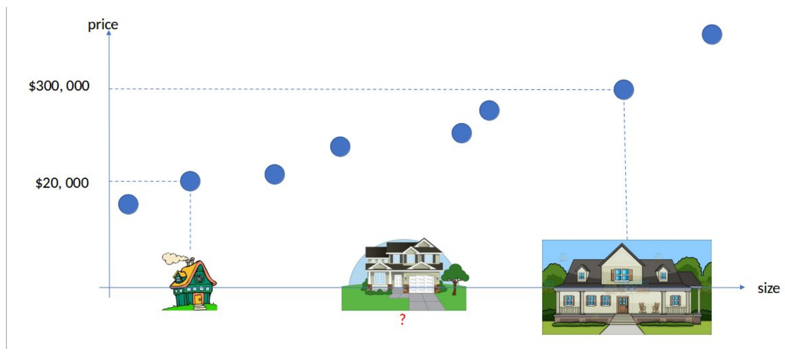
Example : Price of a house

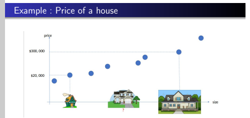




Page 11 :

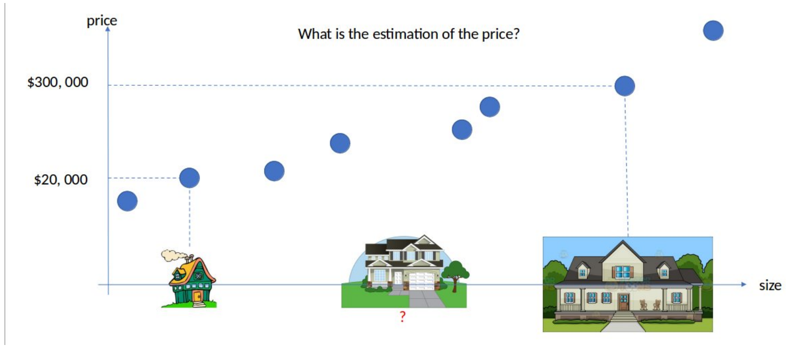
Example : Price of a house

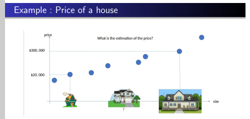




Page 12 :

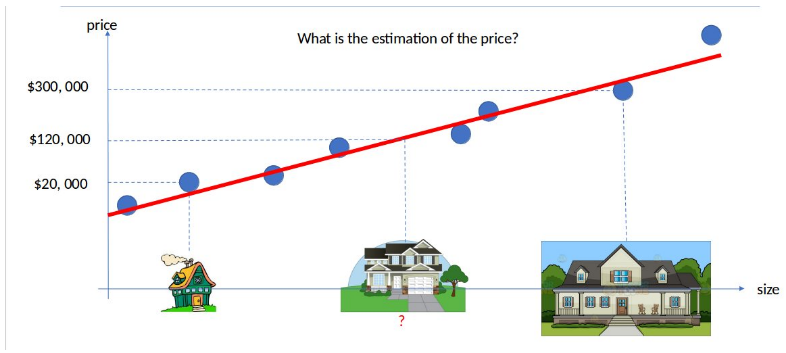
Example : Price of a house

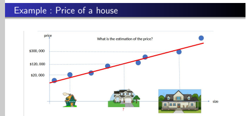




Page 13 :

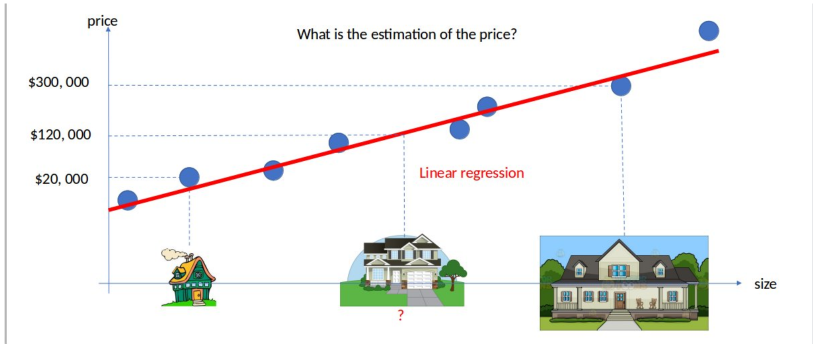
Example : Price of a house





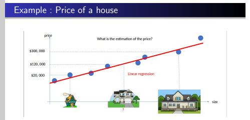
Page 14 :

Example : Price of a house



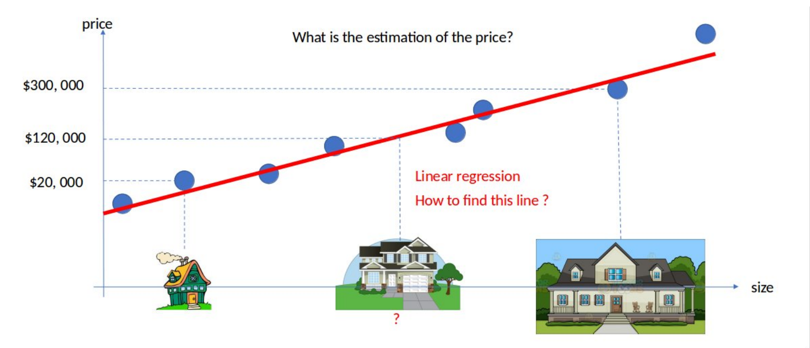
2018-07-06

IML
└Machine Learning
└Linear regression
└Example : Price of a house



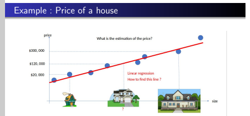
Page 15 :

Example : Price of a house



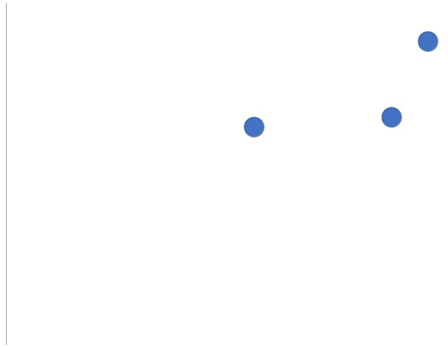
2018-07-06

IML
└Machine Learning
└Linear regression
└Example : Price of a house



Page 16 :

Linear Regression



Page 17 :

Machine Learning
Human Computer Interaction (HCI)
Interactive Machine Learning (IML)
Linear regression
Polynomial regression
Naive Bayes
Decision Tree
Logistic regression
Neural network
SVM
Dataset
Learning Mode

Linear Regression

$$y_i = \beta * x_i + \alpha + \epsilon_i$$

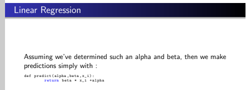
ϵ_i is a (hopefully small) error term representing the fact that there are other factors not accounted for by this simple model.

Page 18 :

Linear Regression

Assuming we've determined such an alpha and beta, then we make predictions simply with :

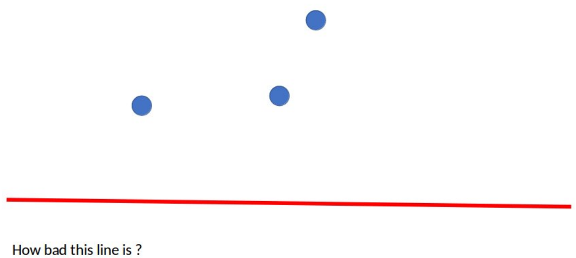
```
def predict(alpha,beta,x_i):
        return beta * x_i +alpha
```

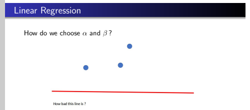



Page 19 :

Linear Regression

How do we choose α and β ?





Page 20 :

Linear Regression

Any choice of α and β gives us a predicted output for each input x_i . Since we know the actual output y_i we can compute the error for each pair :

```
def error ( alpha , beta , x_i , y_i ):
        return y_i - predict ( alpha , beta , x_i )
```


Page 21 :

Linear Regression

We'd really like to know is the total error over the entire data set. But we don't want to just add the errors — if the prediction for x_1 is too high and the prediction for x_2 is too low, the errors may just cancel out.
So instead we add up the squared errors :

```
def sum_of_squared_errors ( alpha , beta , x , y ):
       return sum ( error ( alpha , beta , x_i , y_i ) ** 2 for x_i , y_i in
            zip ( x , y ))
```


Page 22 :

Linear Regression

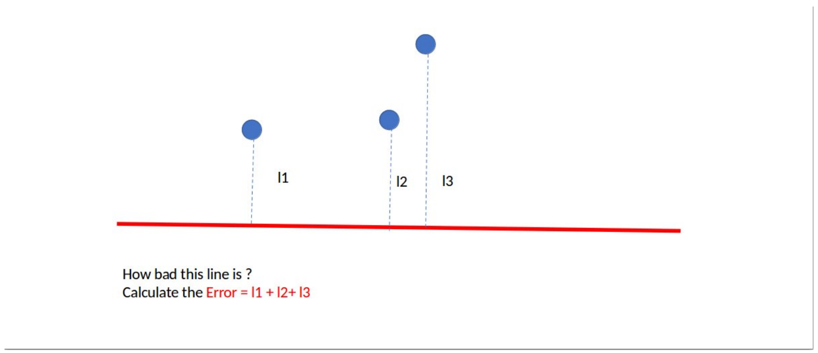
The least squares solution is to choose the α and β that make `sum_of_squared_errors` as small as possible. Using calculus (or tedious algebra), the error-minimizing alpha and beta are given by :

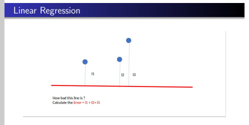
```
def least_squares_fit ( x , y ):
        beta = correlation ( x , y ) * standard_deviation ( y ) /
             standard_deviation ( x )
        alpha = mean ( y ) - beta * mean ( x )
        return alpha , beta
```


Page 23 :

Without going through the exact mathematics, let's think about why this might be a reasonable solution. The choice of alpha simply says that when we see the average value of the independent variable x, we predict the average value of the dependent variable y. The choice of beta means that when the input value increases by standard_deviation(x), the prediction increases by correlation(x, y) *standard_deviation(y). In the case when x and y are perfectly correlated, a one standard deviation increase in x results in a one-standard-deviation-of-y increase in the prediction. When they're perfectly anticorrelated, the increase in x results in a decrease in the prediction. And when the correlation is zero, beta is zero, which means that changes in x don't affect the prediction at all.

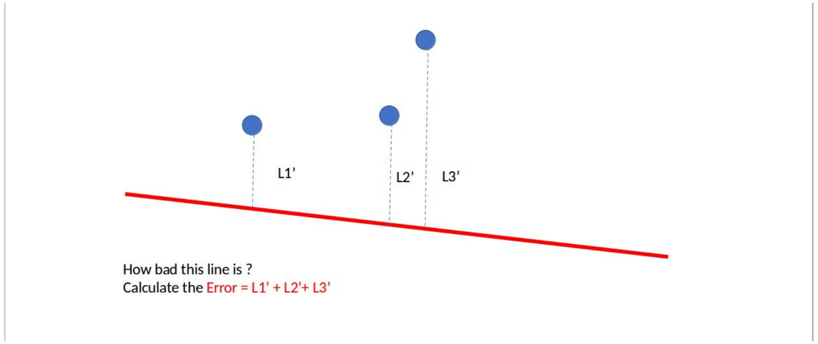
Linear Regression

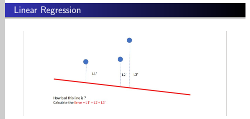




Page 24 :

Linear Regression



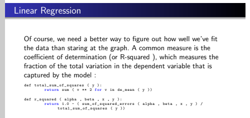


Page 25 :

Linear Regression

Of course, we need a better way to figure out how well we've fit the data than staring at the graph. A common measure is the coefficient of determination (or R-squared), which measures the fraction of the total variation in the dependent variable that is captured by the model :

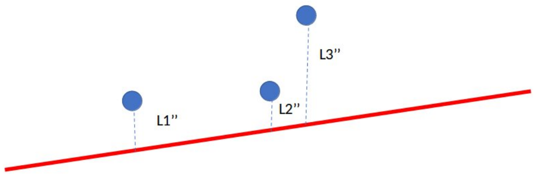
```
def total_sum_of_squares ( y ):
        return sum ( v ** 2 for v in de_mean ( y ))

def r_squared ( alpha , beta , x , y ):
        return 1.0 - ( sum_of_squared_errors ( alpha , beta , x , y ) /
             total_sum_of_squares ( y ))
```

Page 26 :

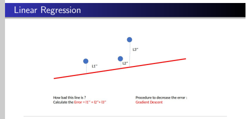
Now, we chose the alpha and beta that minimized the sum of the squared prediction errors. One linear model we could have chosen is "always predict mean(y) " (corresponding to alpha = mean(y) and beta = 0), whose sum of squared errors exactly equals its total sum of squares. This means an R-squared of zero, which indicates a model that (obviously, in this case) performs no better than just predicting the mean. Clearly, the least squares model must be at least as good as that one, which means that the sum of the squared errors is at most the total sum of squares, which means that the R- squared must be at least zero. And the sum of squared errors must be at least 0, which means that the R-squared can be at most 1.

Linear Regression



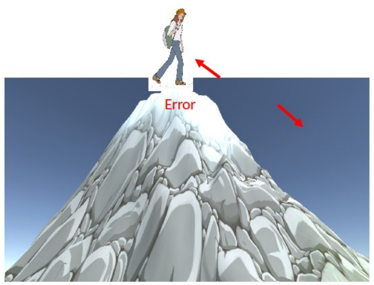
How bad this line is ?
Calculate the Error = l1'' + l2''+ l3''

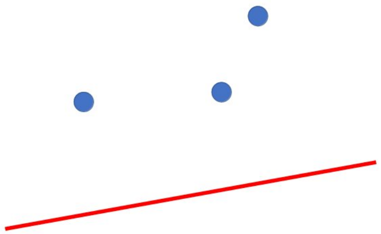
Procedure to decrease the error :
Gradient Descent

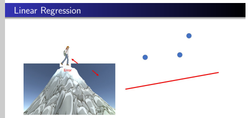


Page 27 :

Linear Regression

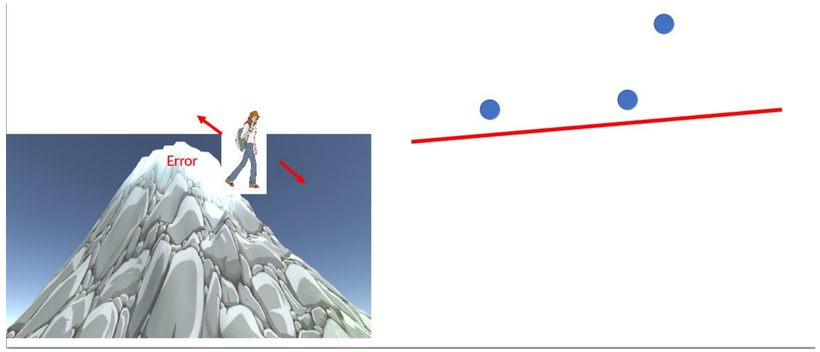


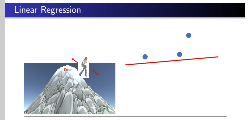




Page 28 :

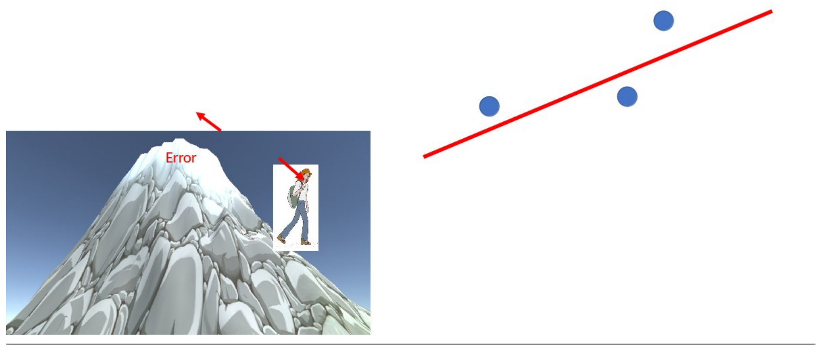
Linear Regression

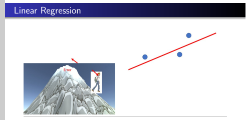




Page 29 :

Linear Regression





Page 30 :

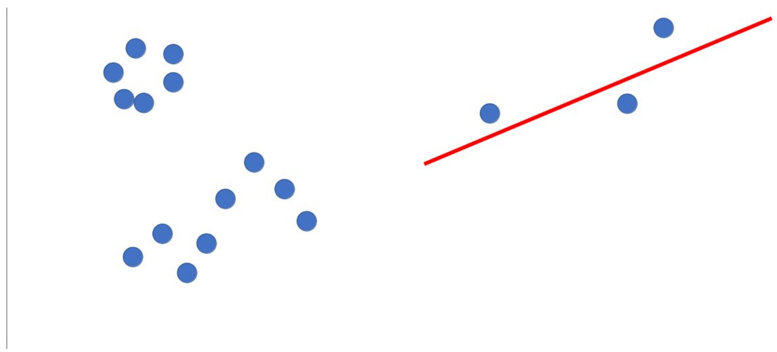
Linear Regression

If we write theta = [alpha, beta] , then we can also solve this using gradient descent :

```
def squared_error ( x_i , y_i , theta ):
        alpha, beta = theta
        return error ( alpha , beta , x_i , y_i ) ** 2

def squared_error_gradient ( x_i , y_i , theta ):
        alpha, beta = theta
        return [ - 2 * error ( alpha , beta , x_i , y_i ), # alpha partial deriv
                 - 2 * error ( alpha , beta , x_i , y_i ) * x_i ] # beta partial
                      deriv

# choose random value to start
random.seed ( 0 )
theta = [ random . random (), random . random ()]
alpha, beta = minimize_stochastic ( squared_error , squared_error_gradient ,
     entry , entry2 , theta , 0.0001 )

print alpha , beta
```

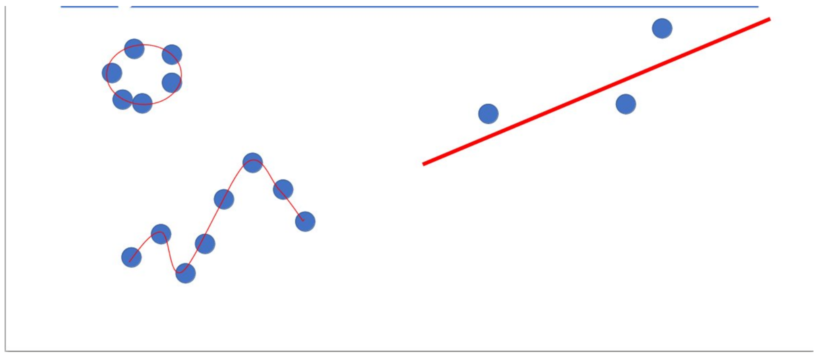

Page 31 :

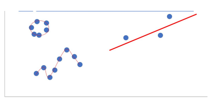
Polynomial Regression



Page 32 :

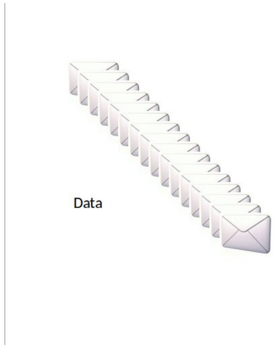
Polynomial Regression

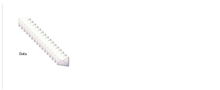




Page 33 :

Example : Spam Detector



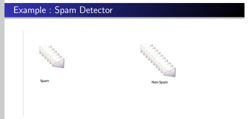


Page 34 :

Example : Spam Detector

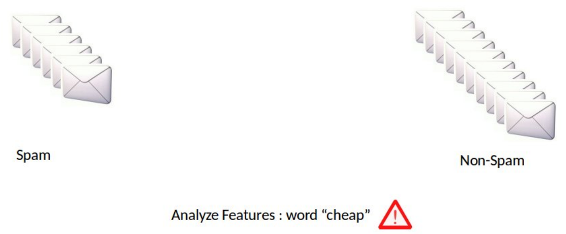
2018-07-06

IML
└─Machine Learning
  └─Naive Bayes
    └─Example : Spam Detector



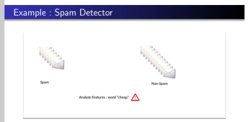
Page 35 :

Example : Spam Detector



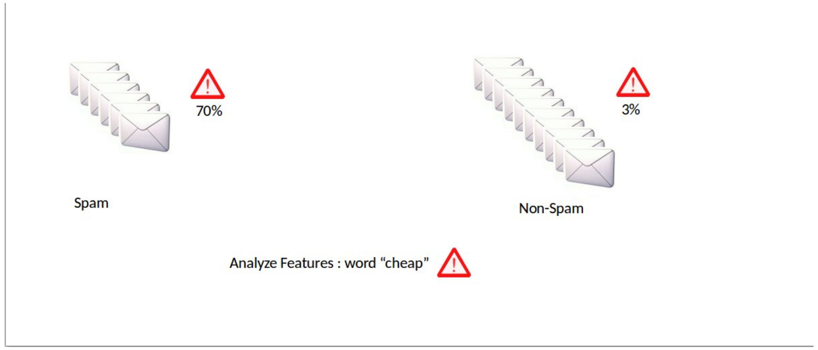
2018-07-06

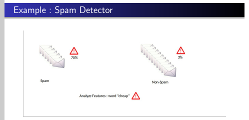
IML
└─Machine Learning
  └─Naive Bayes
    └─Example : Spam Detector



Page 36 :

Example : Spam Detector

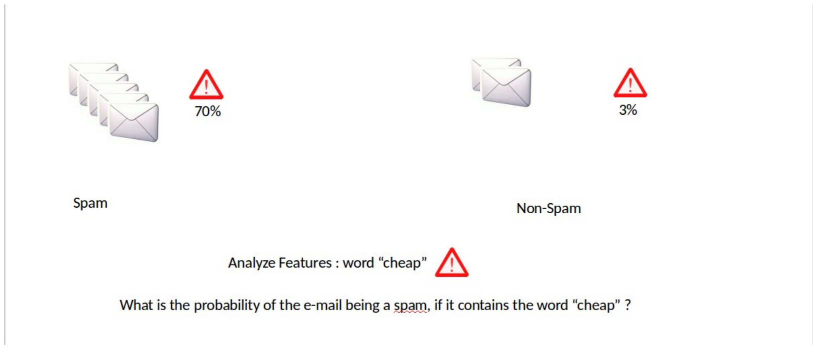


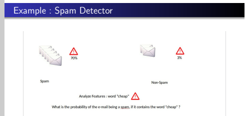


Page 37 :

Machine Learning
Human Computer Interaction (HCI)
Interactive Machine Learning (IML)
Linear regression
Polynomial regression
Naive Bayes
Decision Tree
Logistic regression
Neural network
SVM
Dataset
Learning Mode

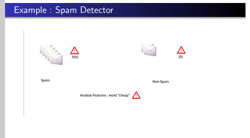
Example : Spam Detector





Page 38 :

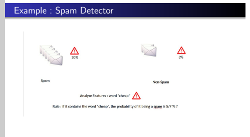
Example : Spam Detector



Page 39 :

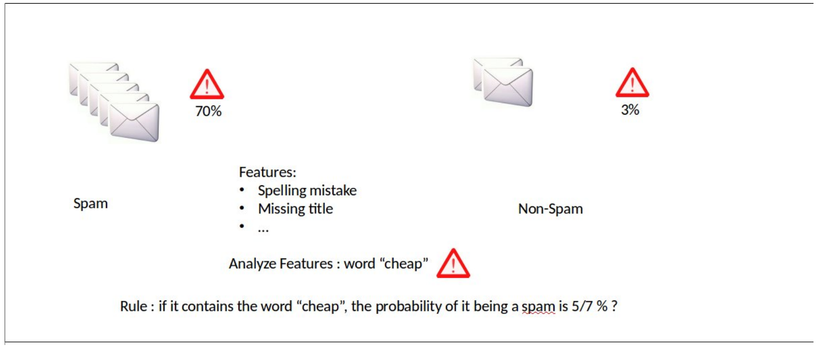
Example : Spam Detector

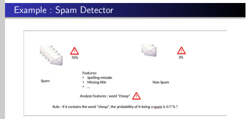
Rule : if it contains the word "cheap", the probability of it being a spam is 5/7 % ?



Page 40 :

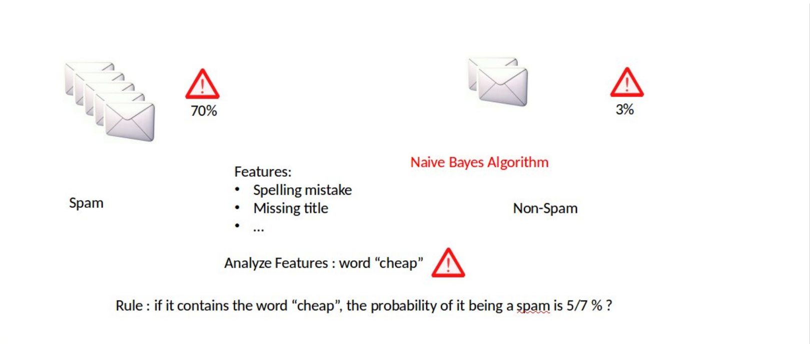
Example : Spam Detector

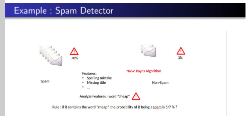




Page 41 :

Example : Spam Detector





Page 42 :

Naive Bayes

- Let S be the event "the message is spam"
- a vocabulary of many words $w_1, ...w_n$
- $P(X_i|S)$: probability that a spam message contains the ith word
- The key to Naive Bayes is making the (big) assumption that the presences (or absences) of each word are independent of one another, conditional on a message being spam or not.
- $P(X_1 = x_1, ...X_n = x_n|S) = P(X_1 = x_1|S) * ...P(X_n = x_n|S)$
- Bayes's Theorem :
  $P(S\ |X = x) = P(X = x|S)/[P(X = x|S) + P(X = x|\neg S)]$

Page 43 :

Naive Bayes

- we usually compute $p_1 * ... * p_n$ as the equivalent :
  $exp(log(p - 1) + ... + log(p_n)$
- Imagine that in our training set the vocabulary word "data" only occurs in nonspam messages. Then we'd estimate $P(''data''|S) = 0$
- $P(X_i|S) =$
  $(k + numberofspamscontainingw_i)/(2k + numberofspams)$

Page 44 :

Naive Bayes

```
def tokenize(message):
        message = message.lower ()                              # convert to
            lowercase
        all_words = re . findall ( "[a-z0-9’]+" , message )     # extract the
            words
        return set ( all_words )                                # remove
            duplicates


def count_words ( training_set ):
        """training␣set␣consists␣of␣pairs␣(message,␣is_spam)"""
        counts = defaultdict ( lambda : [ 0 , 0 ])
        for message , is_spam in training_set :
                for word in tokenize ( message ):
                        counts [ word ][ 0 if is_spam else 1 ] += 1
        return counts
```


2018-07-06

IML
└Machine Learning
  └Naive Bayes
    └Naive Bayes

Page 45 :

Naive Bayes

```
def word_probabilities ( counts , total_spams , total_non_spams , k = 0.5 ):
        """turn␣the␣word_counts␣into␣a␣list␣of␣triplets␣␣␣␣␣w,␣p(w␣|␣spam)␣and␣p
            (w␣|␣~spam)"""
        return [        ( w , ( spam + k ) / ( total_spams + 2 * k ),
                        ( non_spam + k ) / ( total_non_spams + 2 * k )) for w ,
                        ( spam , non_spam ) in counts . iteritems ()]
```


2018-07-06

IML
└Machine Learning
  └Naive Bayes
    └Naive Bayes

Page 46 :

Naive Bayes

```
def spam_probability ( word_probs , message ):
        message_words = tokenize ( message ) log_prob_if_spam =
             log_prob_if_not_spam = 0.0

        # iterate through each word in our vocabulary
        for word , prob_if_spam , prob_if_not_spam in word_probs :

        # if *word* appears in the message,
        # add the log probability of seeing it
        if word in message_words :
                log_prob_if_spam += math . log ( prob_if_spam )
                log_prob_if_not_spam += math . log ( prob_if_not_spam )

        # if *word* doesn't␣appear␣in␣the␣message
␣␣␣␣␣␣␣␣#␣add␣the␣log␣probability␣of␣_not_␣seeing␣it
␣␣␣␣␣␣␣␣#␣which␣is␣log(1␣-␣probability␣of␣seeing␣it)
␣␣␣␣␣␣␣␣else␣:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣log_prob_if_spam␣+=␣math␣.␣log␣(␣1.0␣-␣prob_if_spam␣)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣log_prob_if_not_spam␣+=␣math␣.␣log␣(␣1.0␣-␣prob_if_not_spam␣)

␣␣␣␣␣␣␣␣prob_if_spam␣=␣math␣.␣exp␣(␣log_prob_if_spam␣)
␣␣␣␣␣␣␣␣prob_if_not_spam␣=␣math␣.␣exp␣(␣log_prob_if_not_spam␣)
␣␣␣␣␣␣␣␣return␣prob_if_spam␣/␣(␣prob_if_spam␣+␣prob_if_not_spam␣)
```


Page 47 :

Machine Learning
Human Computer Interaction (HCI)
Interactive Machine Learning (IML)

Naive Bayes

```
class NaiveBayesClassifier :

        def __init__ ( self , k = 0.5 ):
                self . k = k
                self . word_probs = []

        def train ( self , training_set ):
        # count spam and non-spam messages
        num_spams = len ([ is_spam for message , is_spam in training_set if
             is_spam ])
        num_non_spams = len ( training_set ) - num_spams

        # run training data through our "pipeline"
        word_counts = count_words ( training_set )
        self.word_probs = word_probabilities ( word_counts , num_spams ,
             num_non_spams , self.k )

        def classify ( self , message ):
                return spam_probability ( self . word_probs , message)
```


Page 48 :

Naive Bayes

```
import glob , re
# modify the path with wherever you've␣put␣the␣files
path␣=␣r"C:\spam\*\*"
data␣=␣[]

#␣glob.glob␣returns␣every␣filename␣that␣matches␣the␣wildcarded␣path
for␣fn␣in␣glob␣.␣glob␣(␣path␣):
␣␣␣␣␣␣␣␣is_spam␣=␣"ham"␣not␣in␣fn

␣␣␣␣␣␣␣␣with␣open␣(␣fn␣,␣'r'␣)␣as␣file␣:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣line␣in␣file␣:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣line␣.␣startswith␣(␣"Subject:"␣):
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#␣remove␣the␣leading␣"Subject:␣"␣and␣keep␣what's left
                        subject = re . sub ( r"^Subject:␣" , "" , line ) . strip
                              ()
                        data . append (( subject , is_spam ))
```


Page 49 :

Machine Learning
Human Computer Interaction (HCI)
Interactive Machine Learning (IML)

Linear regression
Polynomial regression
Naive Bayes
Decision Tree
Logistic regression
Neural network
SVM
Dataset
Learning Mode

Naive Bayes

```
random . seed ( 0 ) # just so you get the same answers as me
train_data , test_data = split_data ( data , 0.75 )

classifier = NaiveBayesClassifier ()
classifier . train ( train_data )

# triplets (subject, actual is_spam, predicted spam probability)
classified = [( subject , is_spam , classifier . classify ( subject )) for
     subject , is_spam in test_data ]
# assume that spam_probability > 0.5 corresponds to spam prediction
# and count the combinations of (actual is_spam, predicted is_spam)
counts = Counter (( is_spam , spam_probability > 0.5 ) for _ , is_spam ,
     spam_probability in classified )
```


2018-07-06
IML
└Machine Learning
 └Naive Bayes
  └Naive Bayes

Page 50 :

Example : Recommending apps

Gender	Age	App
F	15	Facebook
F	25	Instagram
M	32	Snapchat
F	40	Instagram
M	12	Facebook
M	14	Facebook

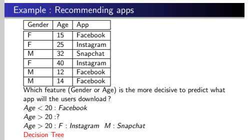
Which feature (Gender or Age) is the more decisive to predict what app will the users download ?

$Age < 20$: *Facebook*

$Age > 20$:?

$Age > 20$: F : *Instagram* M : *Snapchat*

Decision Tree



Page 51 :

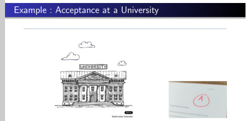
Given how closely decision trees can fit themselves to their training data, it's not surprising that they have a tendency to overfit. One way of avoiding this is a technique called *random forests* , in which we build multiple decision trees and let them vote on how to classify inputs.

Example : Acceptance at a University

buche@enib.fr

Page 52 :

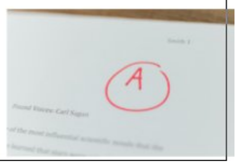
Example : Acceptance at a University

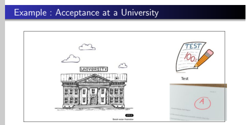


Page 53 :

Example : Acceptance at a University

Test





Page 54 :

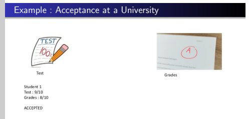
Example : Acceptance at a University

Test

Grades

Student 1
Test : 9/10
Grades : 8/10

ACCEPTED



Page 55 :

Example : Acceptance at a University

Test

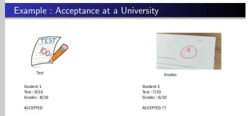
Grades

Student 1
Test : 9/10
Grades : 8/10

ACCEPTED

Student 3
Test : 7/10
Grades : 6/10

ACCEPTED ??



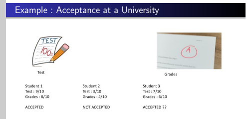
Page 56 :

Example : Acceptance at a University

Test

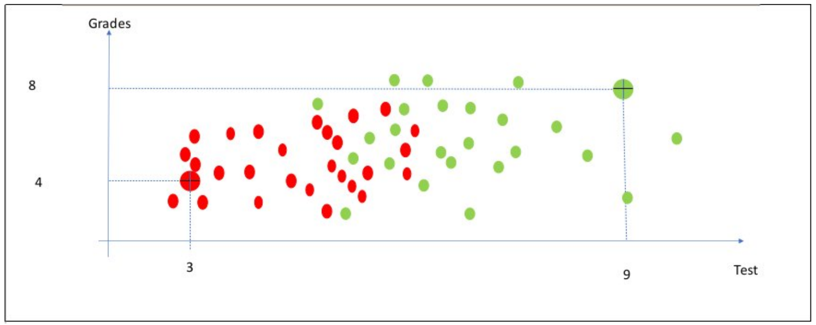
Grades

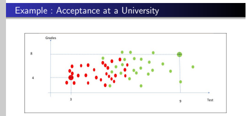
Student 1	Student 2	Student 3
Test : 9/10	Test : 3/10	Test : 7/10
Grades : 8/10	Grades : 4/10	Grades : 6/10
ACCEPTED	NOT ACCEPTED	ACCEPTED ??



Page 57 :

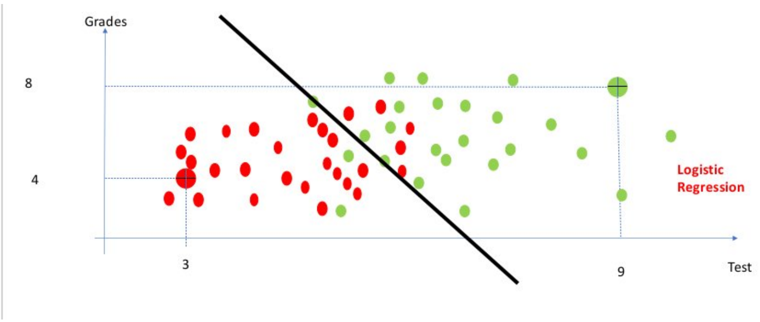
Example : Acceptance at a University

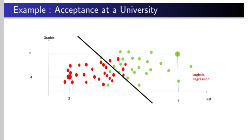




Page 58 :

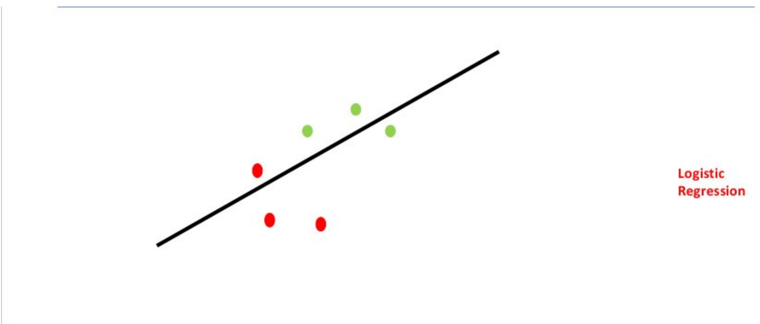
Example : Acceptance at a University

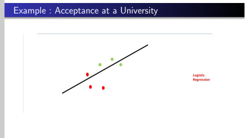




Page 59 :

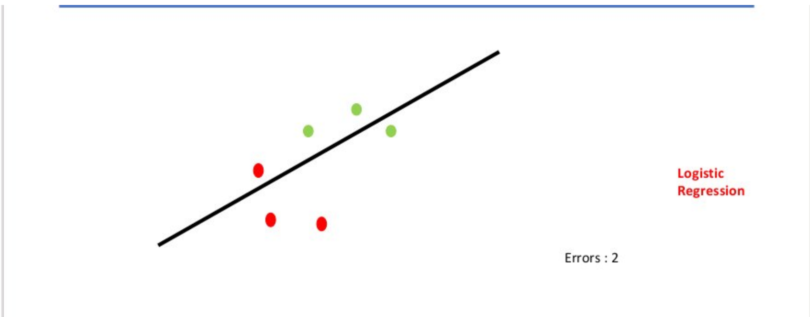
Example : Acceptance at a University

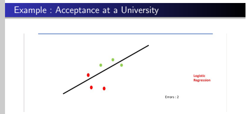




Page 60 :

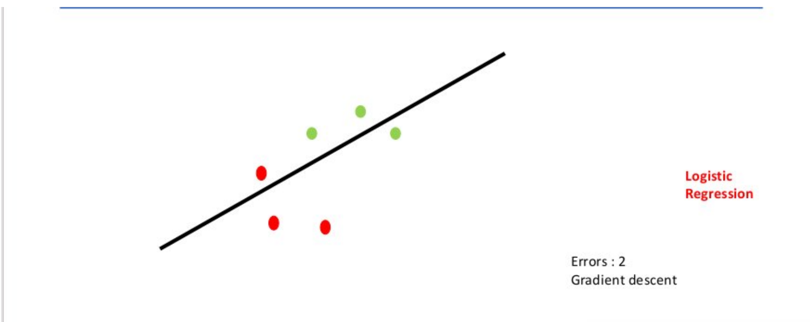
Example : Acceptance at a University

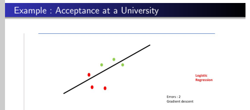




Page 61 :

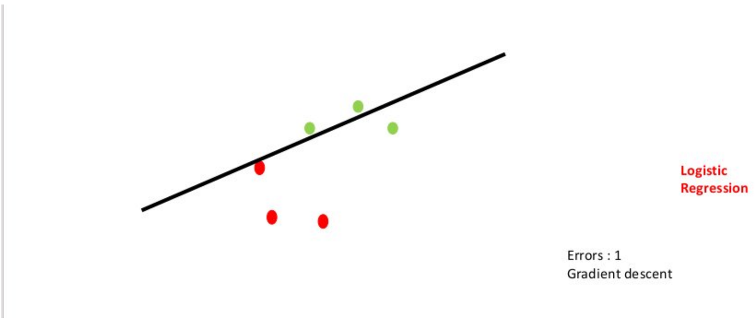
Example : Acceptance at a University

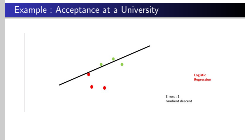




Page 62 :

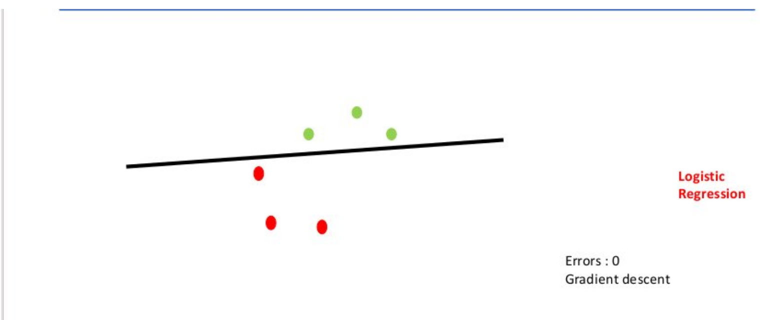
Example : Acceptance at a University

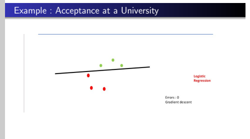




Page 63 :

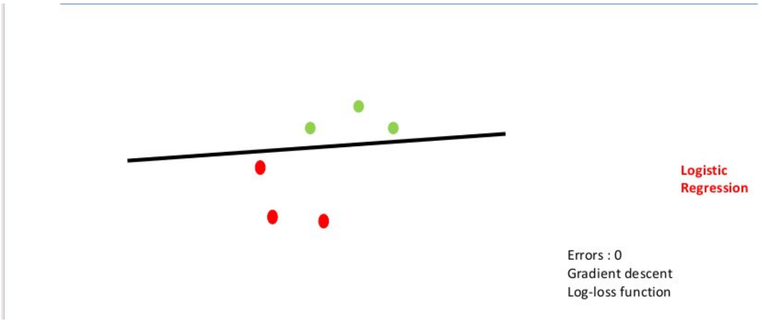
Example : Acceptance at a University

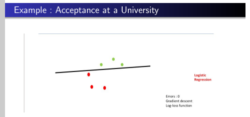




Page 64 :

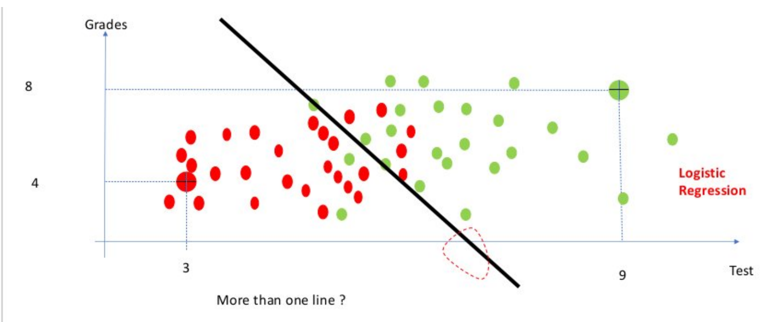
Example : Acceptance at a University

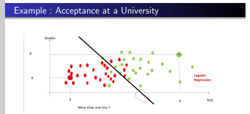




Page 65 :

Example : Acceptance at a University

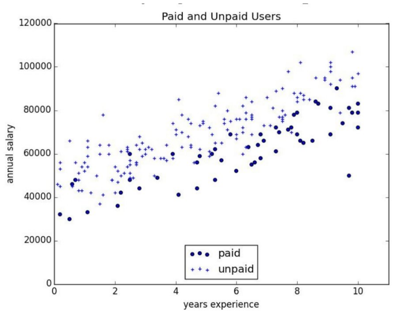


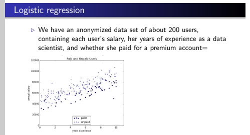


Page 66 :

Logistic regression

- ▷ We have an anonymized data set of about 200 users, containing each user's salary, her years of experience as a data scientist, and whether she paid for a premium account=



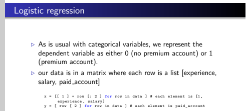


Page 67 :

Logistic regression

- ▷ As is usual with categorical variables, we represent the dependent variable as either 0 (no premium account) or 1 (premium account).
- ▷ our data is in a matrix where each row is a list [experience, salary, paid_account]

```
x = [[ 1 ] + row [: 2 ] for row in data ] # each element is [1,
    experience , salary]
y = [ row [ 2 ] for row in data ] # each element is paid_account
```

Page 68 :

Logistic regression

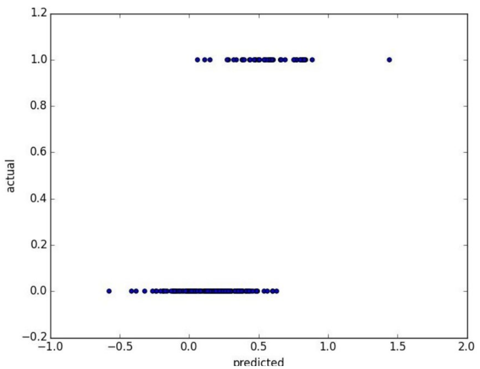
- linear regression :
  $paidAccount = \beta_0 + \beta_1 * experience + \beta_2 * salary + \epsilon$

```
rescaled_x = rescale ( x )
beta = estimate_beta ( rescaled_x , y ) # [0.26, 0.43, -0.43]
predictions = [ predict ( x_i , beta ) for x_i in rescaled_x ]
plt.scatter ( predictions , y )
plt.xlabel ( "predicted" )
plt.ylabel ( "actual" )
plt.show ()
```


Page 69 :

Logistic regression

- result (linear regression) :

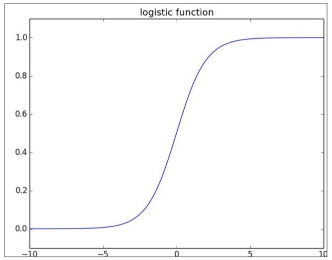


Page 70 :

Logistic regression

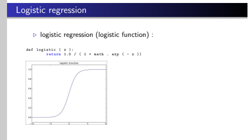
▷ logistic regression (logistic function) :

```
def logistic ( x ):
      return 1.0 / ( 1 + math . exp ( - x ))
```



2018-07-06

IML
└Machine Learning
  └Logistic regression
    └Logistic regression



Page 71 :

Logistic regression

▷ derivative is given by :

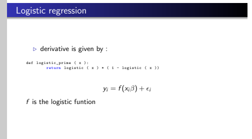
```
def logistic_prime ( x ):
      return logistic ( x ) * ( 1 - logistic ( x ))
```

$$y_i = f(x_i\beta) + \epsilon_i$$

f is the logistic funtion

2018-07-06

IML
└Machine Learning
  └Logistic regression
    └Logistic regression



Page 72 :

Logistic regression

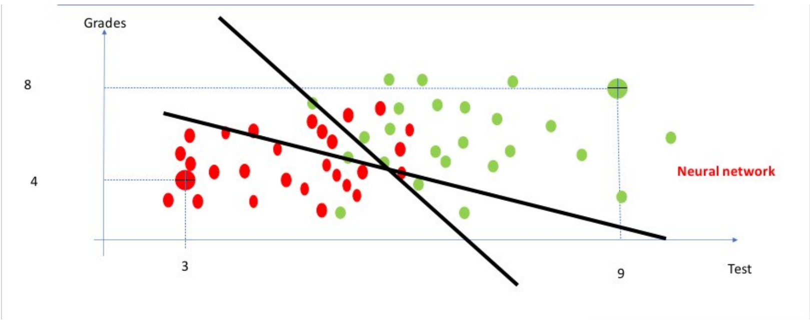
```
def logistic_log_likelihood_i ( x_i , y_i , beta ):
        if y_i == 1 :
                return math . log ( logistic ( dot ( x_i , beta )))
        else :
                return math . log ( 1 - logistic ( dot ( x_i , beta )))

def logistic_log_likelihood ( x , y , beta ):
        return sum ( logistic_log_likelihood_i ( x_i , y_i , beta ) for x_i ,
             y_i in zip ( x , y ))

def logistic_log_gradient_i ( x_i , y_i , beta ):
        """the␣gradient␣of␣the␣log␣likelihood␣␣␣␣␣corresponding␣to␣the␣ith␣data␣
             point"""
        return [ logistic_log_partial_ij ( x_i , y_i , beta , j ) for j , _ in
             enumerate ( beta )]

def logistic_log_gradient ( x , y , beta ):
        return reduce ( vector_add , [ logistic_log_gradient_i ( x_i , y_i ,
             beta ) for x_i , y_i in zip ( x , y )])
```

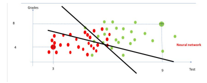

Page 73 :

Logistic regression

```
random . seed ( 0 )
x_train , x_test , y_train , y_test = train_test_split ( rescaled_x , y , 0.33 )

# want to maximize log likelihood on the training data
fn = partial ( logistic_log_likelihood , x_train , y_train )
gradient_fn = partial ( logistic_log_gradient , x_train , y_train )

# pick a random starting point
beta_0 = [ random . random () for _ in range ( 3 )]

# and maximize using gradient descent
beta_hat = maximize_batch ( fn , gradient_fn , beta_0 )
```


Page 74 :

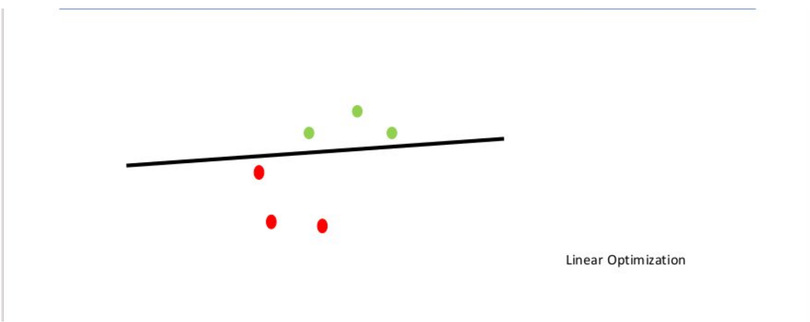


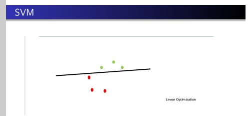


Page 75 :

Machine Learning
Human Computer Interaction (HCI)
Interactive Machine Learning (IML)

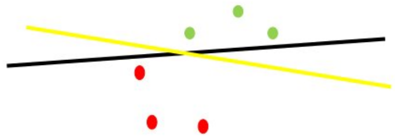
SVM



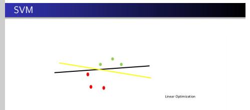


Page 76 :

SVM



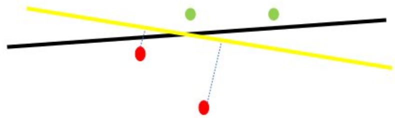
Linear Optimization



Page 77 :

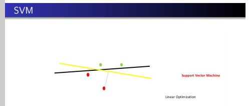
Machine Learning
Human Computer Interaction (HCI)
Interactive Machine Learning (IML)

SVM



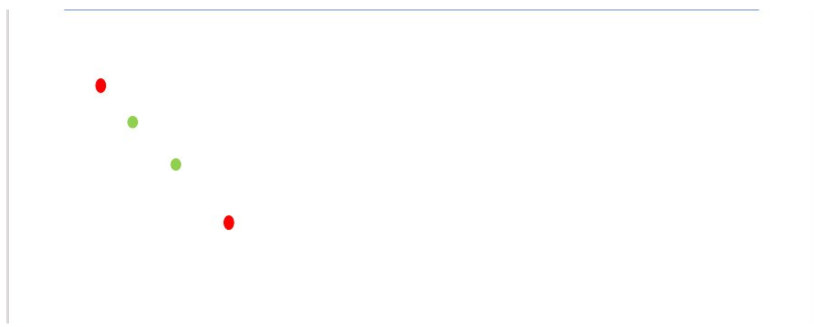
Support Vector Machine

Linear Optimization



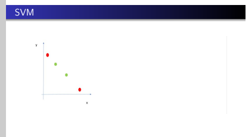
Page 78 :

SVM



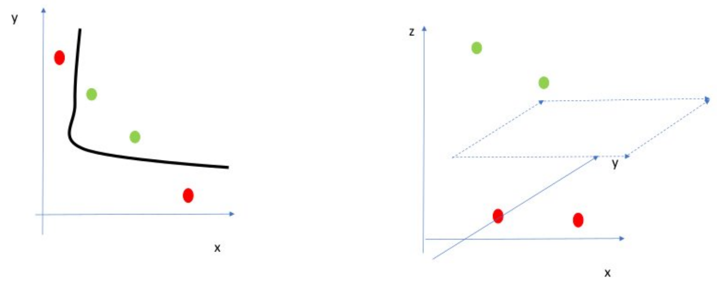
Page 79 :

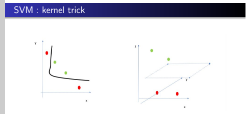
SVM



Page 80 :

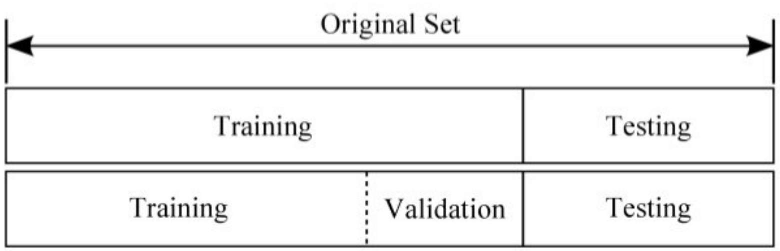
SVM : kernel trick

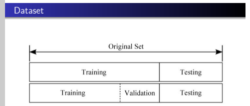




Page 81 :

Dataset





Page 82 :

```
def split_data(data,prob):
        """split␣data␣into␣fractions␣[prob,␣1␣-␣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 83 :

- **Supervised** : given a set of feature/label pairs, find a rule that predicts the label associated with a previously unseen input
- **Unsupervised** : given a features vectors (without labels) group them into "natural clusters" (or create labels for groups)

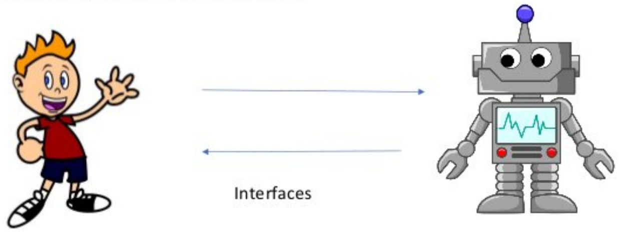
Page 84 :

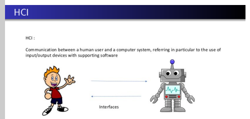
Page 85 :

HCI

HCI :

Communication between a human user and a computer system, referring in particular to the use of input/output devices with supporting software

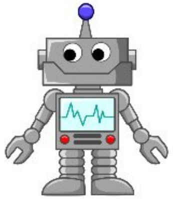




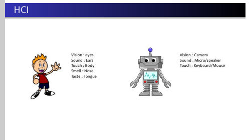
Page 86 :

HCI

Vision : eyes
Sound : Ears
Touch : Body
Smell : Nose
Taste : Tongue



Vision : Camera
Sound : Micro/speaker
Touch : Keyboard/Mouse



Page 87 :

Page 88 :

IML

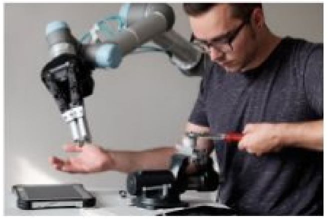
- ▷ Autonomous machine learning systems : often require intense engineering effort to be effective
- ▷ How machines can interact with people to solve problems more efficiently than autonomous systems ?
  - ◇ Humans interacting with robots to teach them to perform tasks
  - ◇ Humans helping virtual agents play video games given feedback on their performance
  - ◇ ...

IML
▷ Autonomous machine learning systems : often require intense engineering effort to be effective
▷ How machines can interact with people to solve problems more efficiently than autonomous systems ?
◇ Humans interacting with robots to teach them to perform tasks
◇ Humans helping virtual agents play video games given feedback on their performance
◇ ...

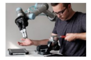
Page 89 :

IML

- ▷ Domain :
  - ◇ Machine Learning
  - ◇ Artificial intelligence
  - ◇ Human-computer interaction
  - ◇ Cognitive science
  - ◇ Robotics



IML
▷ Domain :
◇ Machine Learning
◇ Artificial intelligence
◇ Human-computer interaction
◇ Cognitive science
◇ Robotics



Page 90 :

Introduction

IML

Cédric Buche

ENIB

6 juillet 2018

Page 91 :