Machine Learning

IML

Cédric Buche

ENIB

27 août 2019

IML

2019-08-27

Page 1 :

1. Machine Learning
2. Supervised - Regression
   - Linear regression
   - Polynomial regression
3. Supervised - Classification
   - Naive Bayes
   - Decision Tree
   - Logistic regression
   - KNN
   - Neural network
   - SVM
4. Unsupervised - Clustering
   - k-means
   - Hierarchical clustering
   - Distance

IML
└─Machine Learning

2019-08-27

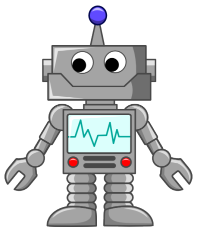
Page 2 :

1. Machine Learning
2. Supervised - Regression
   - Linear regression
   - Polynomial regression
3. Supervised - Classification
   - Naive Bayes
   - Decision Tree
   - Logistic regression
   - KNN
   - Neural network
   - SVM
4. Unsupervised - Clustering
   - k-means
   - Hierarchical clustering
   - Distance

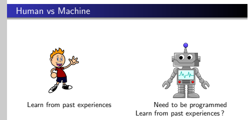
Page 3 :

Human vs Machine

Learn from past experiences

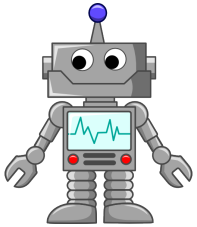


Need to be programmed
Learn from past experiences ?

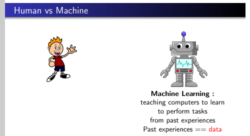


Page 4 :

Human vs Machine



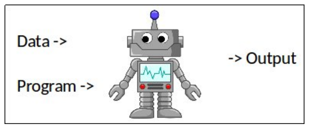
Machine Learning :
teaching computers to learn
to perform tasks
from past experiences
Past experiences == data



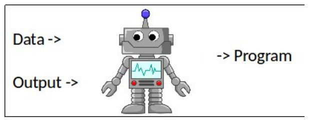
Page 5 :

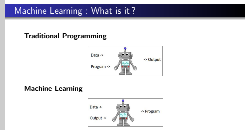
Machine Learning : What is it ?

Traditional Programming



Machine Learning





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 ?

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 :

1 Machine Learning
2 Supervised - Regression
- Linear regression
- Polynomial regression
3 Supervised - Classification
- Naive Bayes
- Decision Tree
- Logistic regression
- KNN
- Neural network
- SVM
4 Unsupervised - Clustering
- k-means
- Hierarchical clustering
- Distance

Machine Learning
Supervised - Regression
Linear regression
Polynomial regression
Supervised - Classification
Naive Bayes
Decision Tree
Logistic regression
KNN
Neural network
SVM
Unsupervised - Clustering
k-means
Hierarchical clustering
Distance

Page 8 :

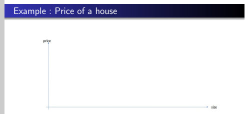
$20, 000

$300, 000

?

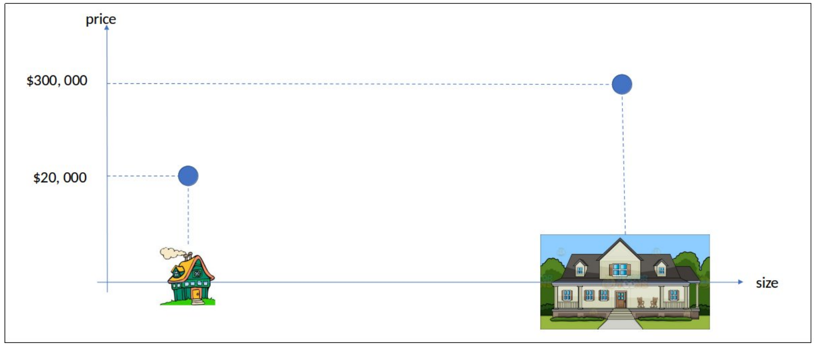
Page 9 :

Example : Price of a house



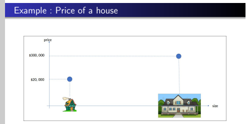
Page 10 :

Example : Price of a house

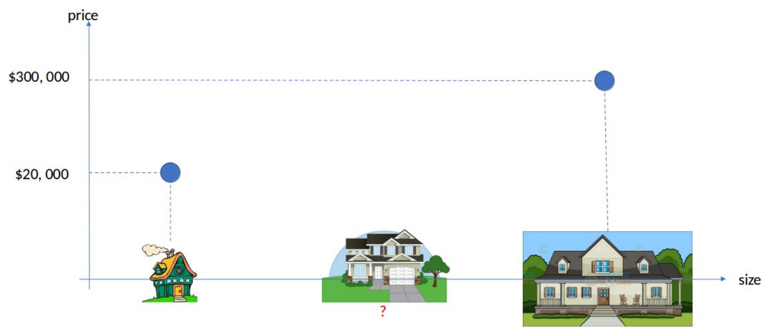


2019-08-27

IML
└Supervised - Regression
└Linear regression
└Example : Price of a house

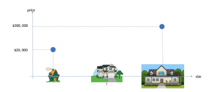


Page 11 :



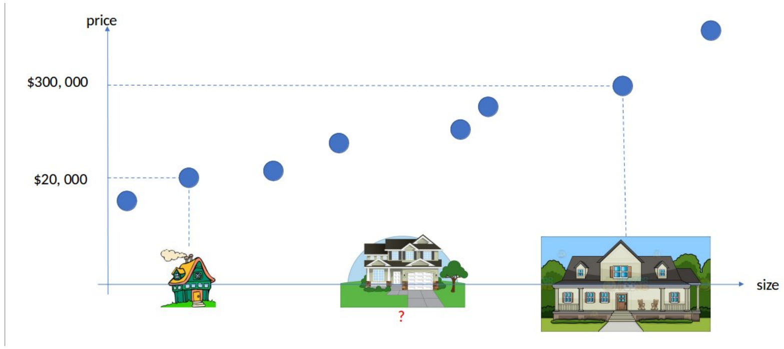
2019-08-27

IML
└Supervised - Regression
└Linear regression



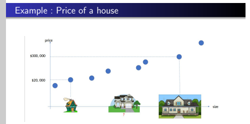
Page 12 :

Example : Price of a house



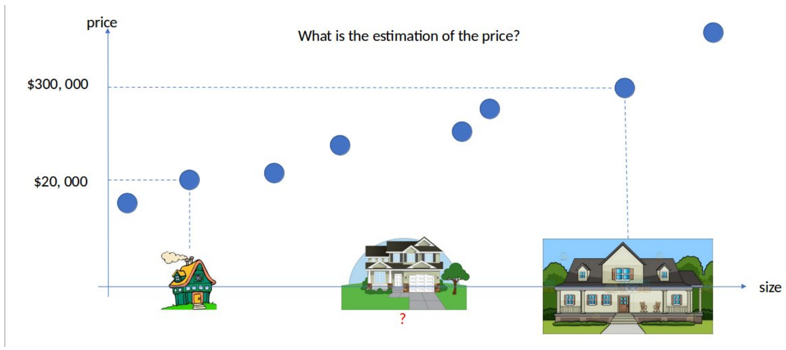
IML
└Supervised - Regression
 └Linear regression
  └Example : Price of a house

2019-08-27



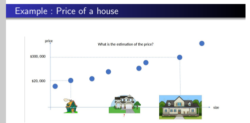
Page 13 :

Example : Price of a house



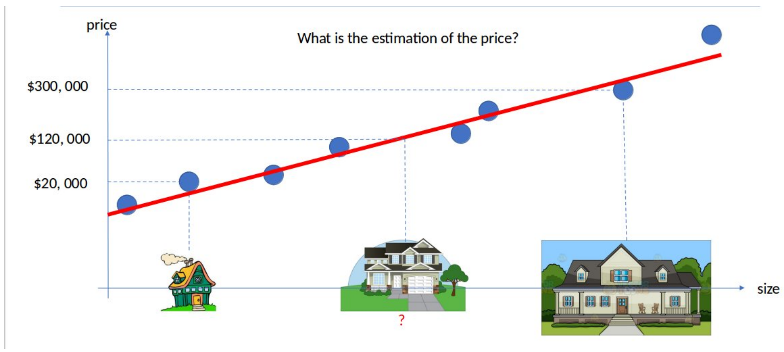
IML
└Supervised - Regression
 └Linear regression
  └Example : Price of a house

2019-08-27



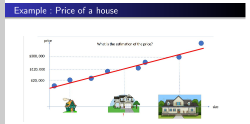
Page 14 :

Example : Price of a house



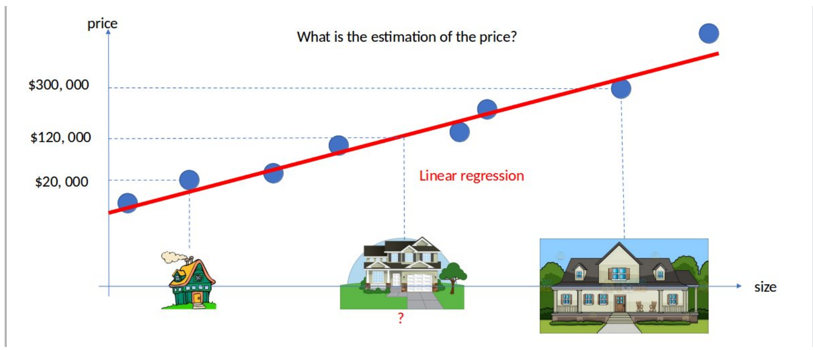
2019-08-27

IML
└Supervised - Regression
└Linear regression
└Example : Price of a house



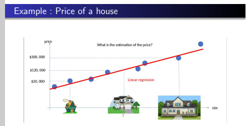
Page 15 :

Example : Price of a house



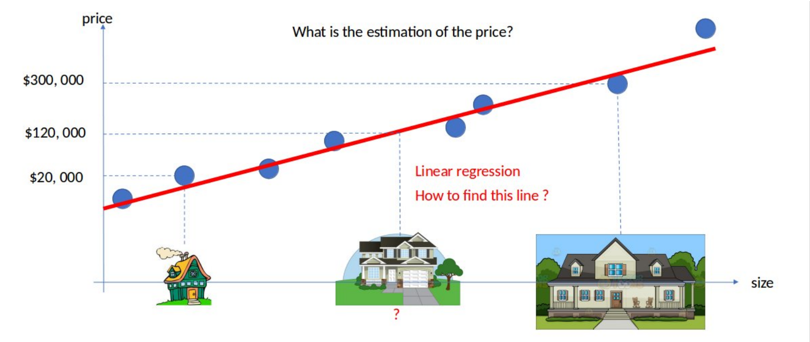
2019-08-27

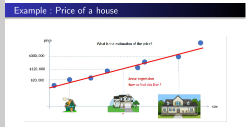
IML
└Supervised - Regression
└Linear regression
└Example : Price of a house



Page 16 :

Example : Price of a house



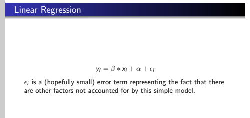


Page 17 :

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.



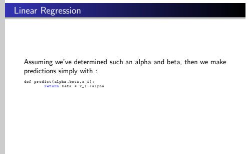
Page 18 :

The representation of linear regression is an equation that describes a line that best fits the relationship between the input variables (x) and the output variables (y), by finding specific weightings for the input variables called coefficients (B).
For example :
$y = B_0 + B_1 * x$
We will predict y given the input x and the goal of the linear regression learning algorithm is to find the values for the coefficients B_0 and B_1.
Different techniques can be used to learn the linear regression model from data, such as a linear algebra solution for ordinary least squares and gradient descent optimization.
Linear regression is a fast and simple technique and good first algorithm to try.

Linear Regression

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

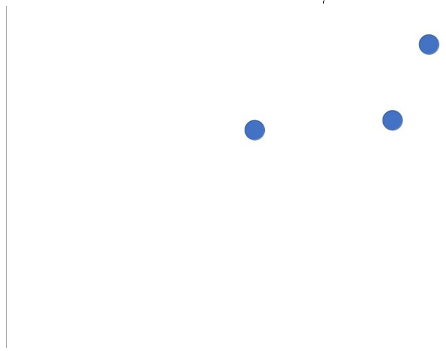
```
def predict(alpha,beta,x_i):
        return beta * x_i +alpha
```

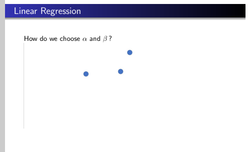



Page 19 :

Linear Regression

How do we choose α and β ?

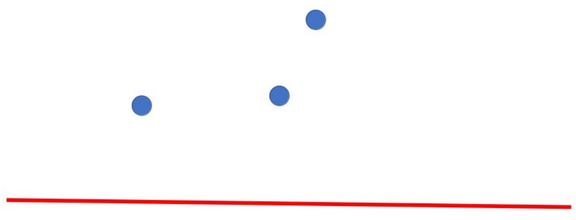




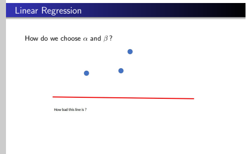
Page 20 :

Linear Regression

How do we choose α and β ?



How bad this line is ?

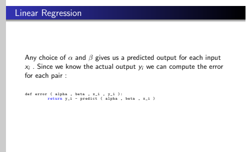


Page 21 :

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 22 :

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 23 :

Machine Learning
Supervised - Regression
Supervised - Classification
Unsupervised - Clustering

Linear regression
Polynomial regression

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 :

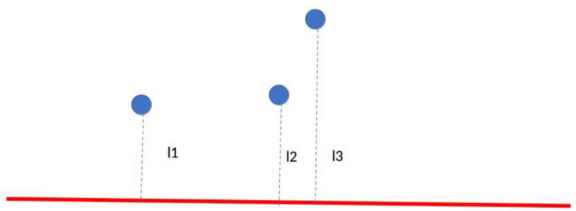
```
def least_squares_fit ( x , y ):
        beta = correlation ( x , y ) * standard_deviation ( y ) /
             standard_deviation ( x )
        alpha = mean ( y ) - beta * mean ( x )
        return alpha , beta
```


2019-08-27
IML
└─Supervised - Regression
└─Linear regression
└─Linear Regression

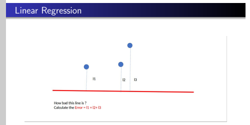
Page 24 :

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

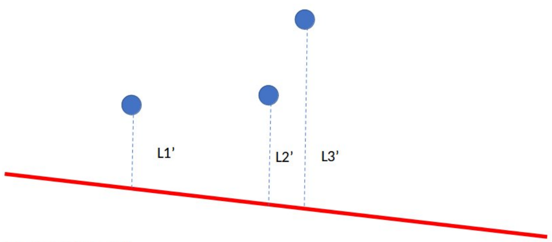


How bad this line is ?
Calculate the Error = l1 + l2+ l3

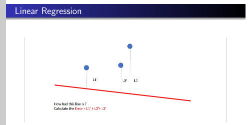


Page 25 :

Linear Regression



How bad this line is ?
Calculate the Error = L1' + L2'+ L3'

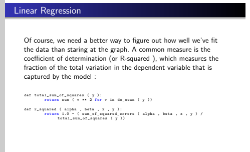


Page 26 :

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 :

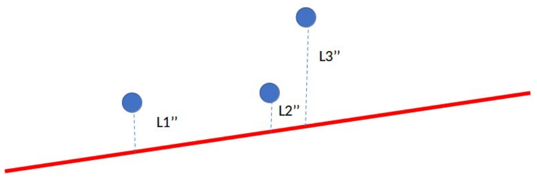
```
def total_sum_of_squares ( y ):
        return sum ( v ** 2 for v in de_mean ( y ))

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

Page 27 :

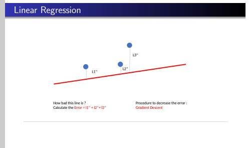
Now, we chose the alpha and beta that minimized the sum of the squared prediction errors. One linear model we could have chosen is "always predict mean(y) " (corresponding to alpha = mean(y) and beta = 0), whose sum of squared errors exactly equals its total sum of squares. This means an R-squared of 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



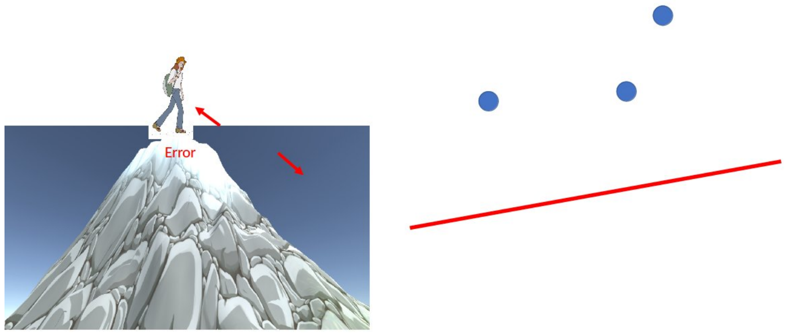
How bad this line is ?
Calculate the Error = l1'' + l2''+ l3''

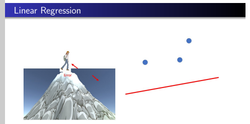
Procedure to decrease the error :
Gradient Descent



Page 28 :

Linear Regression

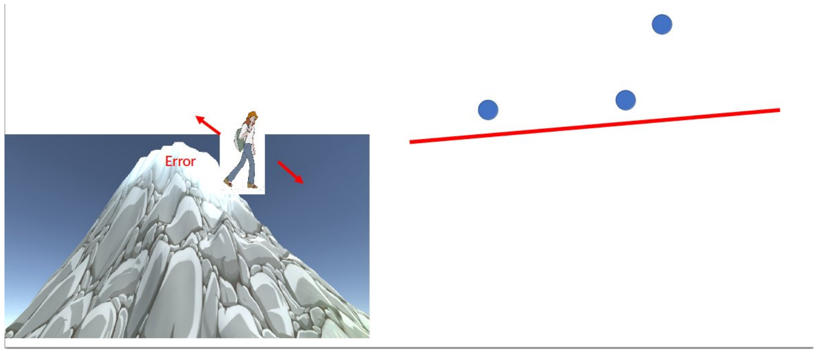




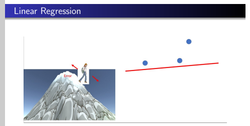
Page 29 :

Machine Learning
Supervised - Regression
Supervised - Classification
Unsupervised - Clustering
Linear regression
Polynomial regression

Linear Regression

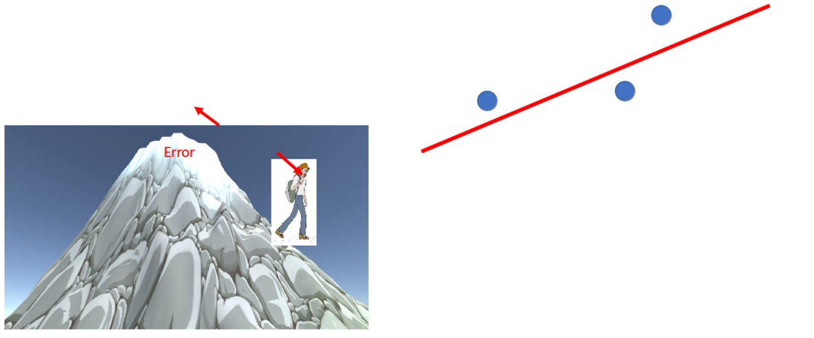


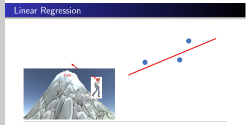
2019-08-27
IML
└Supervised - Regression
└Linear regression
└Linear Regression



Page 30 :

Linear Regression





Page 31 :

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 32 :

Linear Regression

Demo !

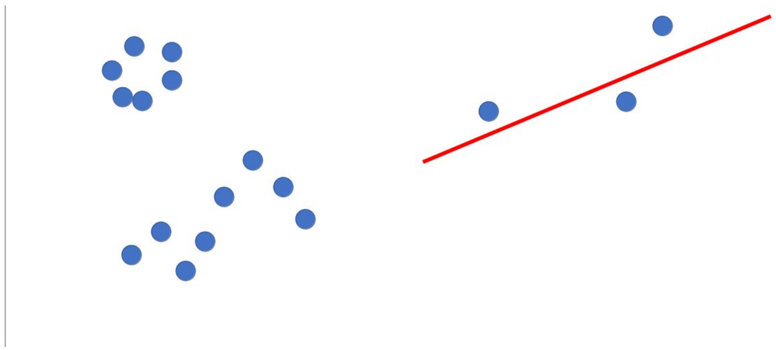
Linear Regression

Demo !

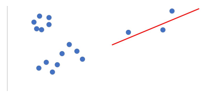
Page 33 :

list daily_minutes that show many minutes per day each user spends on a website, and you've ordered it so that its elements correspond to the elements of num_friends list. We'd like to investigate the relationship between these two metrics.

Polynomial Regression

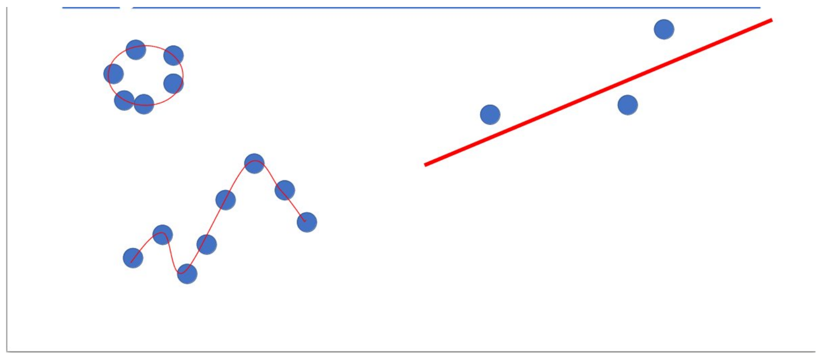


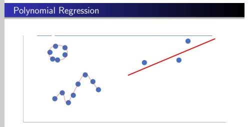
Polynomial Regression



Page 34 :

Polynomial Regression



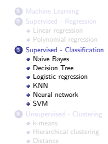


Page 35 :

1 Machine Learning
2 Supervised - Regression
- Linear regression
- Polynomial regression

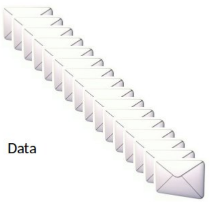
3 Supervised - Classification
- Naive Bayes
- Decision Tree
- Logistic regression
- KNN
- Neural network
- SVM

4 Unsupervised - Clustering
- k-means
- Hierarchical clustering
- Distance



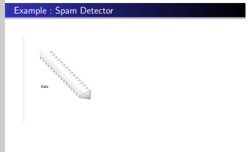
Page 36 :

Example : Spam Detector



2019-08-27

IML
└Supervised - Classification
  └Naive Bayes
    └Example : Spam Detector

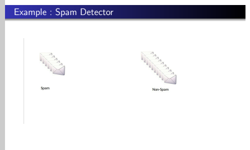


Page 37 :

Example : Spam Detector

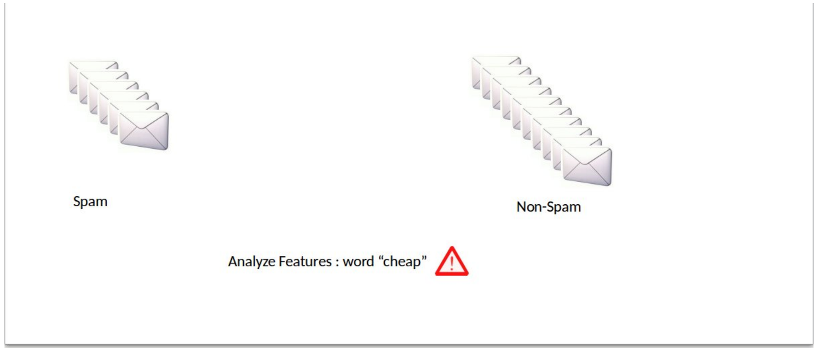
2019-08-27

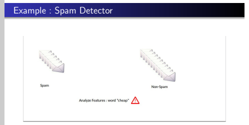
IML
└Supervised - Classification
  └Naive Bayes
    └Example : Spam Detector



Page 38 :

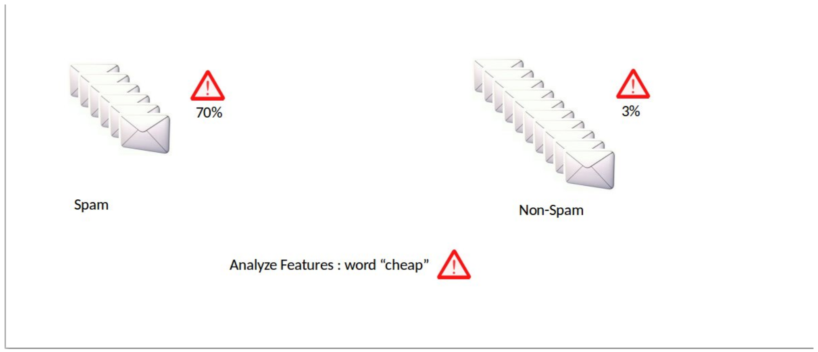
Example : Spam Detector

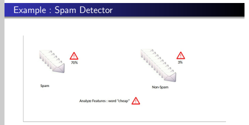




Page 39 :

Example : Spam Detector





Page 40 :

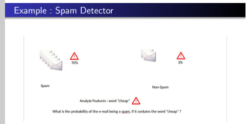
Example : Spam Detector

Spam

Non-Spam

Analyze Features : word “cheap”

What is the probability of the e-mail being a spam, if it contains the word “cheap” ?



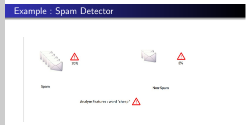
Page 41 :

Example : Spam Detector

Spam

Non-Spam

Analyze Features : word “cheap”



Page 42 :

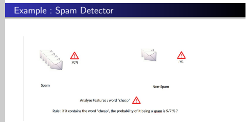
Example : Spam Detector

Spam

Non-Spam

Analyze Features : word "cheap"

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



Page 43 :

Example : Spam Detector

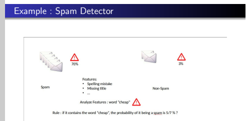
Spam

Features:
- Spelling mistake
- Missing title
- ...

Non-Spam

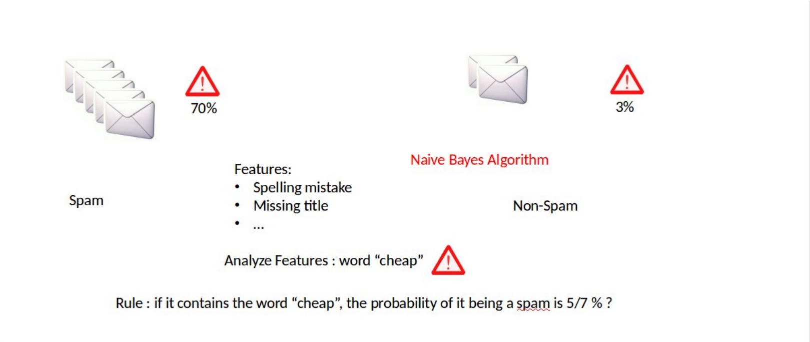
Analyze Features : word "cheap"

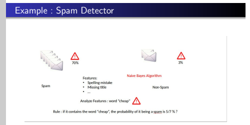
Rule : if it contains the word "cheap", the probability of it being a spam is 5/7 % ?



Page 44 :

Example : Spam Detector





Page 45 :

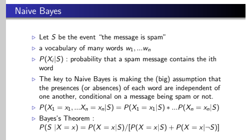
Naive Bayes is a simple but surprisingly powerful algorithm for predictive modeling.
The model is comprised of two types of probabilities that can be calculated directly from your training data :

1. The probability of each class.
2. The conditional probability for each class given each x value.

Once calculated, the probability model can be used to make predictions for new data using Bayes Theorem.
When your data is real-valued it is common to assume a Gaussian distribution (bell curve) so that you can easily estimate these probabilities.
Naive Bayes is called naive because it assumes that each input variable is independent. This is a strong assumption and unrealistic for real data, nevertheless, the technique is very effective on a large range of complex problems.

Naive Bayes

- Let S be the event "the message is spam"
- a vocabulary of many words $w_1, \ldots w_n$
- $P(X_i|S)$: 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.
- $P(X_1 = x_1, \ldots X_n = x_n|S) = P(X_1 = x_1|S) * \ldots P(X_n = x_n|S)$
- Bayes's Theorem : $P(S\ |X = x) = P(X = x|S)/[P(X = x|S) + P(X = x|\neg S)]$



Page 46 :

Naive Bayes

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

Page 47 :

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
```


Page 48 :

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 ()]
```


Page 49 :

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* does not 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 50 :

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 51 :

Naive Bayes

```
import glob , re
# modify the path with wherever you have put the files
path = .....
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 is left
                        subject = re.sub( r"^Subject:␣" , "" , line ).strip ()
                        data.append(( subject , is_spam ))
```


Page 52 :

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 )
```


Page 53 :

Machine Learning
Supervised - Regression
Supervised - Classification
Unsupervised - Clustering
Naive Bayes
Decision Tree
Logistic regression
KNN
Neural network
SVM

Naive Bayes

Demo !

2019-08-27
IML
└Supervised - Classification
└Naive Bayes
└Naive Bayes

Page 54 :

three folders : spam, easy_ham, and hard_ham. Each folder contains many emails, each contained in a singlefile. To keep things really simple, we will just look at the subject lines of each email.

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 55 :

Decision Trees are an important type of algorithm for predictive modeling machine learning.
The representation of the decision tree model is a binary tree. This is your binary tree from algorithms and data structures, nothing too fancy. Each node represents a single input variable (x) and a split point on that variable (assuming the variable is numeric).
The leaf nodes of the tree contain an output variable (y) which is used to make a prediction. Predictions are made by walking the splits of the tree until arriving at a leaf node and output the class value at that leaf node.
Trees are fast to learn and very fast for making predictions. They are also often accurate for a broad range of problems and do not require any special preparation for your data.
Decision trees have a high variance and can yield more accurate predictions when used in an ensemble.
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

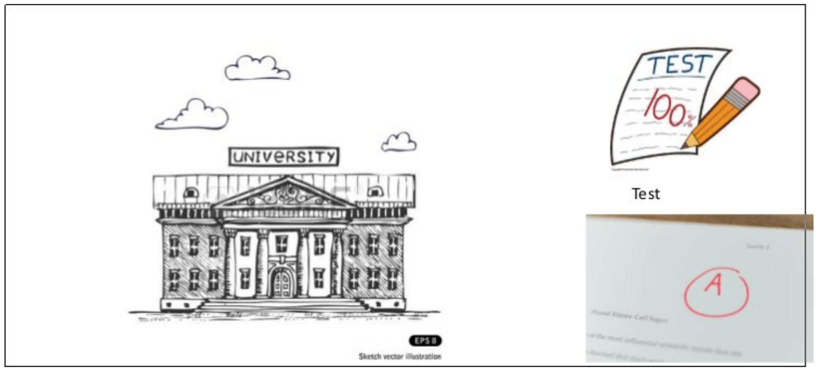
buche@enib.fr

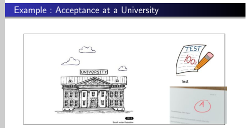
Page 56 :

Example : Acceptance at a University

Page 57 :

Example : Acceptance at a University





Page 58 :

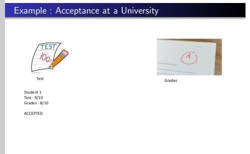
Example : Acceptance at a University

Test

Grades

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

ACCEPTED



Page 59 :

Example : Acceptance at a University

Test

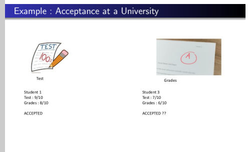
Grades

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

ACCEPTED

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

ACCEPTED ??



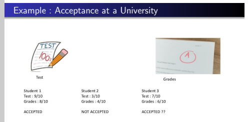
Page 60 :

Example : Acceptance at a University

Test

Grades

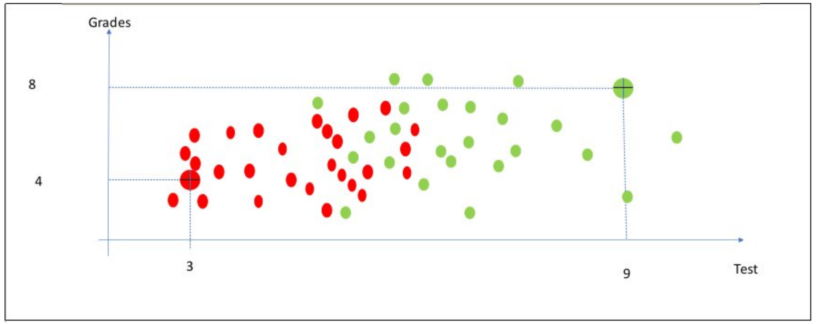
Student 1	Student 2	Student 3
Test : 9/10	Test : 3/10	Test : 7/10
Grades : 8/10	Grades : 4/10	Grades : 6/10
ACCEPTED	NOT ACCEPTED	ACCEPTED ??



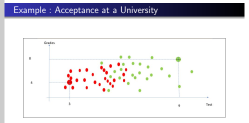
Page 61 :

Machine Learning
Supervised - Regression
Supervised - Classification
Unsupervised - Clustering
Naive Bayes
Decision Tree
Logistic regression
KNN
Neural network
SVM

Example : Acceptance at a University

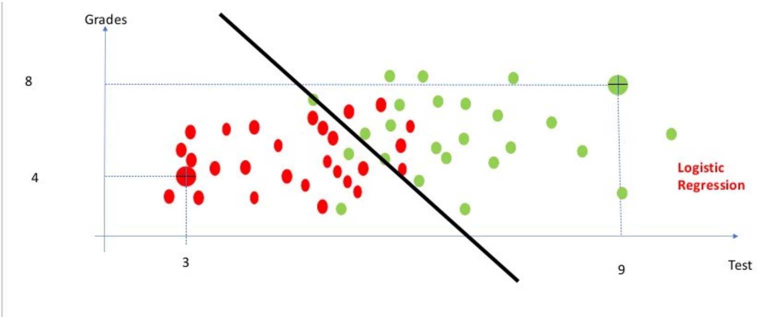


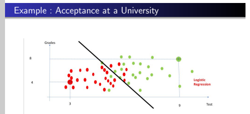
2019-08-27
IML
└Supervised - Classification
└Logistic regression
└Example : Acceptance at a University



Page 62 :

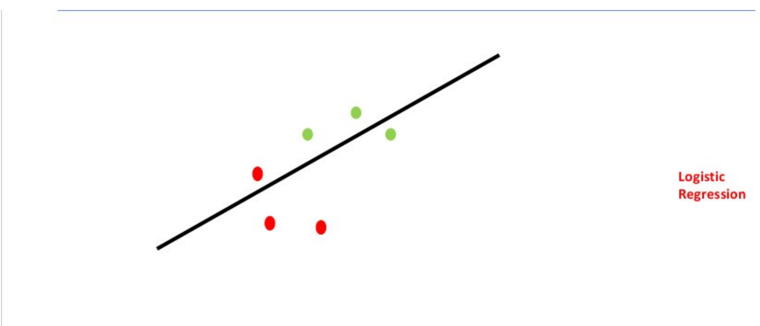
Example : Acceptance at a University

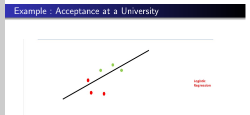




Page 63 :

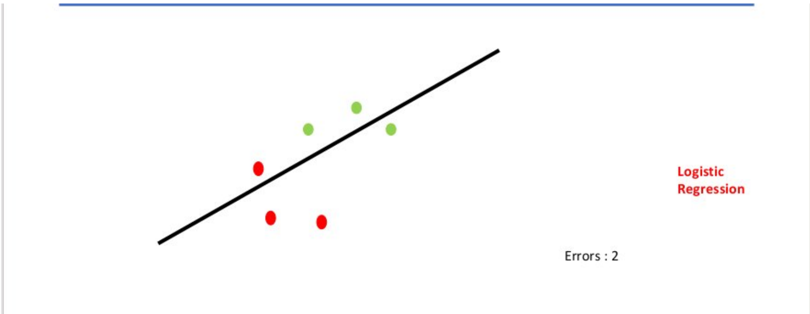
Example : Acceptance at a University





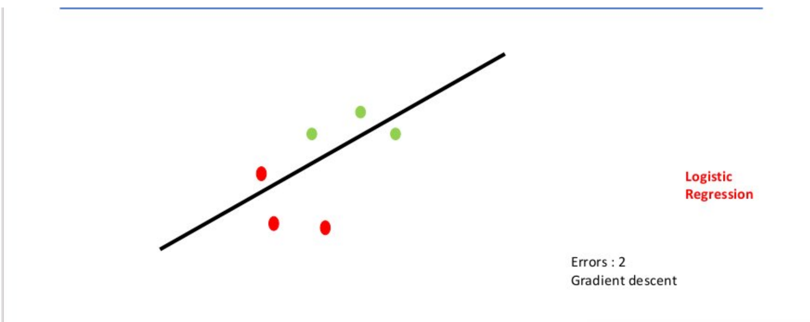
Page 64 :

Example : Acceptance at a University



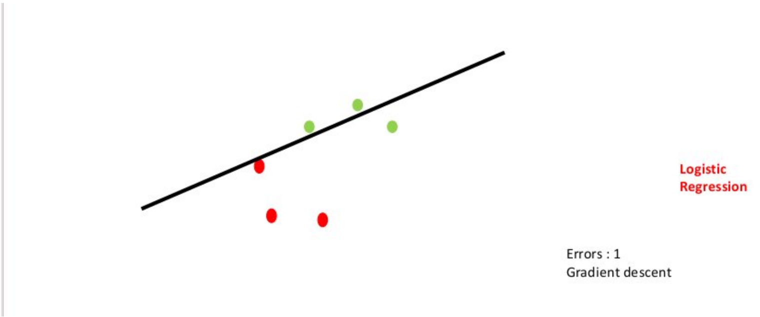
Page 65 :

Example : Acceptance at a University



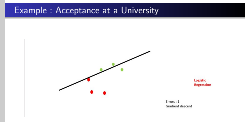
Page 66 :

Example : Acceptance at a University



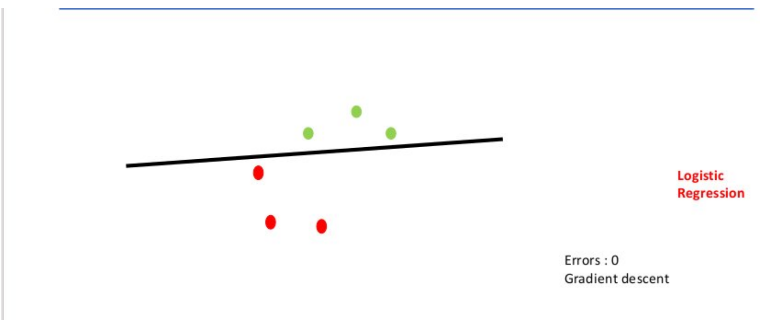
2019-08-27

IML
└Supervised - Classification
└Logistic regression
└Example : Acceptance at a University



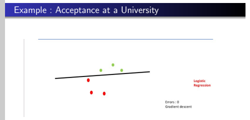
Page 67 :

Example : Acceptance at a University



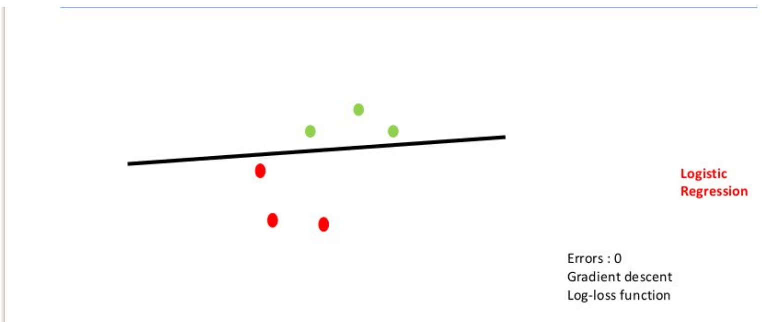
2019-08-27

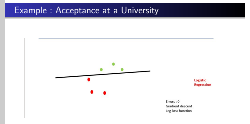
IML
└Supervised - Classification
└Logistic regression
└Example : Acceptance at a University



Page 68 :

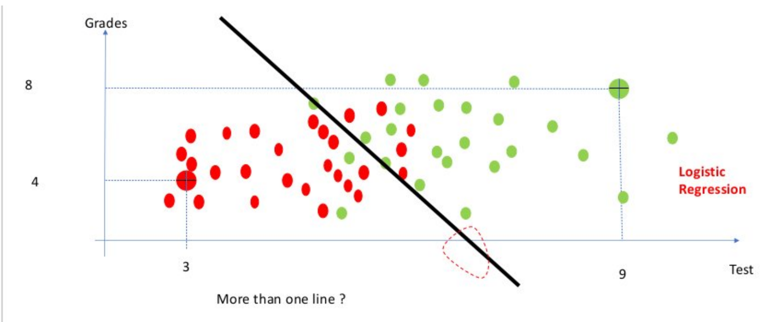
Example : Acceptance at a University

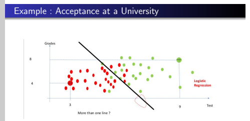




Page 69 :

Example : Acceptance at a University



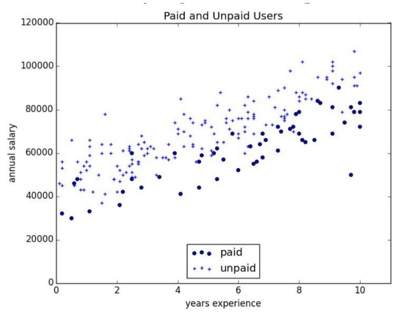


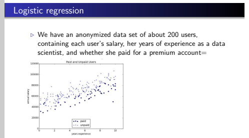
Page 70 :

Logistic regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values).
Logistic regression is like linear regression in that the goal is to find the values for the coefficients that weight each input variable.
Unlike linear regression, the prediction for the output is transformed using a non-linear function called the logistic function.
The logistic function looks like a big S and will transform any value into the range 0 to 1. This is useful because we can apply a rule to the output of the logistic function to snap values to 0 and 1 (e.g. IF less than 0.5 then output 1) and predict a class value.
Because of the way that the model is learned, the predictions made by logistic regression can also be used as the probability of a given data instance belonging to class 0 or class 1. This can be useful for problems where you need to give more rationale for a prediction.
Like linear regression, logistic regression does work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar (correlated) to each other.
It's a fast model to learn and effective on binary classification problems.

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=



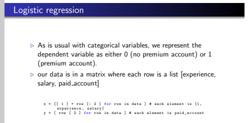


Page 71 :

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
```

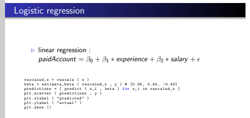



Page 72 :

Logistic regression

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

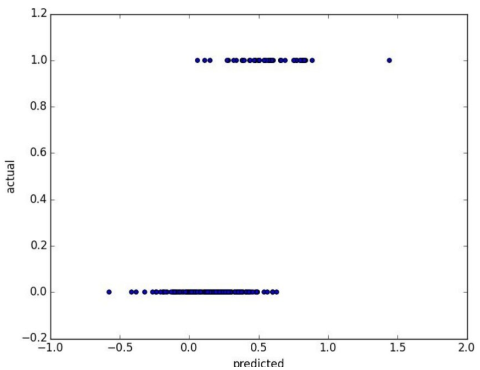
```
rescaled_x = rescale ( x )
beta = estimate_beta ( rescaled_x , y ) # [0.26, 0.43, -0.43]
predictions = [ predict ( x_i , beta ) for x_i in rescaled_x ]
plt.scatter ( predictions , y )
plt.xlabel ( "predicted" )
plt.ylabel ( "actual" )
plt.show ()
```

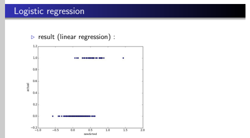



Page 73 :

Logistic regression

- result (linear regression) :



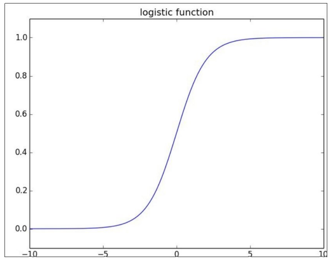


Page 74 :

Logistic regression

- logistic regression (logistic function) :

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



Page 75 :

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

Page 76 :

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 77 :

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 78 :

Naive Bayes

Demo !

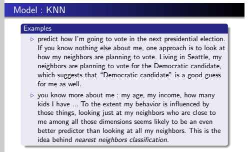
Page 79 :

the problem of trying to predict which users paid for premium accounts.

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*.



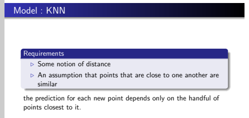
Page 80 :

Model : KNN

Requirements

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

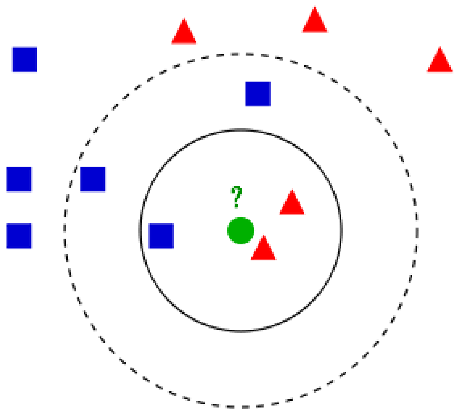
the prediction for each new point depends only on the handful of points closest to it.

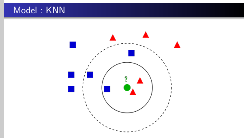


Page 81 :

Machine Learning
Supervised - Regression
Supervised - Classification
Unsupervised - Clustering
Naive Bayes
Decision Tree
Logistic regression
KNN
Neural network
SVM

Model : KNN





Page 82 :

The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.
If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

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 83 :

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)
```


Page 84 :

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
]
```


Page 85 :

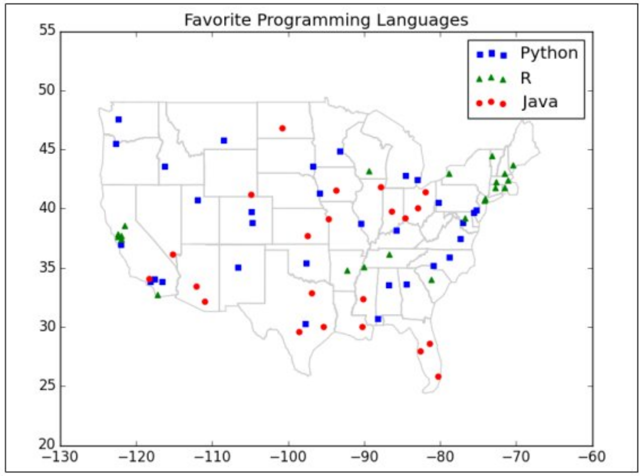
Example : Favorite Programming Languages

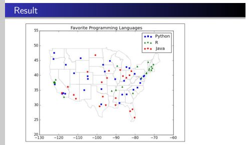
Plotting the data

```
# 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 86 :

Result



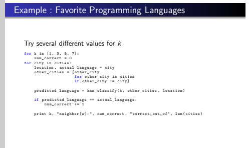


Page 87 :

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 88 :

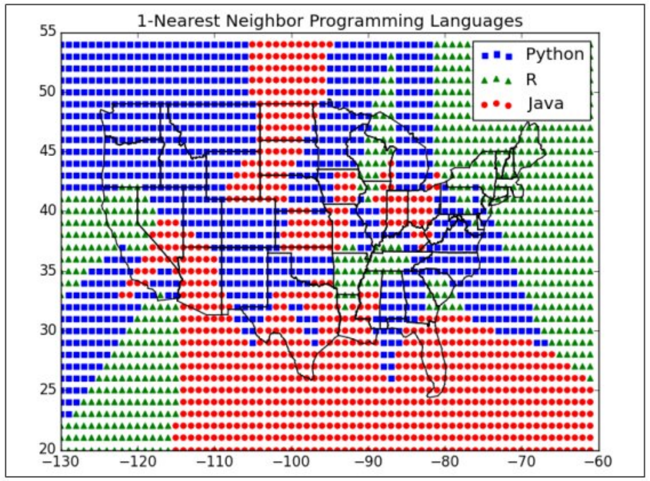
1 neighbor[s] : 40 correct out of 75 3 neighbor[s] : 44 correct out of 75 5 neighbor[s] : 41 correct out of 75 7 neighbor[s] : 35 correct out of 75

Example : Favorite Programming Languages

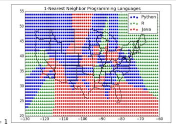
```
plots = { "Java" : ([], []), "Python" : ([], []), "R" : ([], []) }

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 89 :

Result



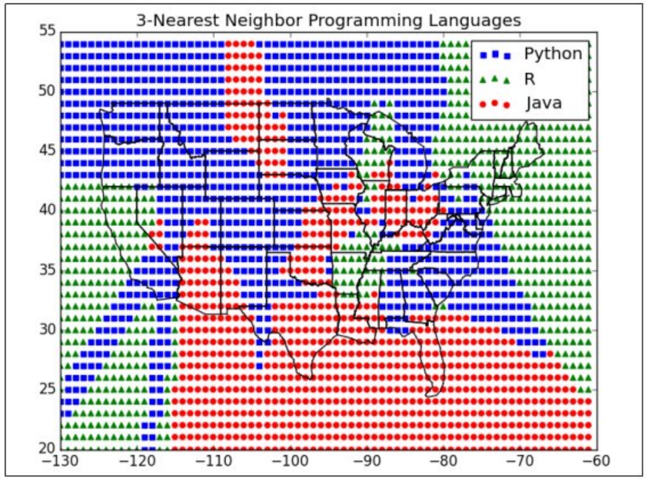
k = 1



Page 90 :

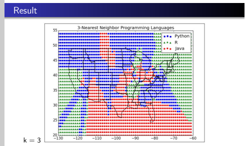
Result

k = 3



2019-08-27

IML
└Supervised - Classification
  └KNN
    └Result



Page 91 :

KNN

Demo !

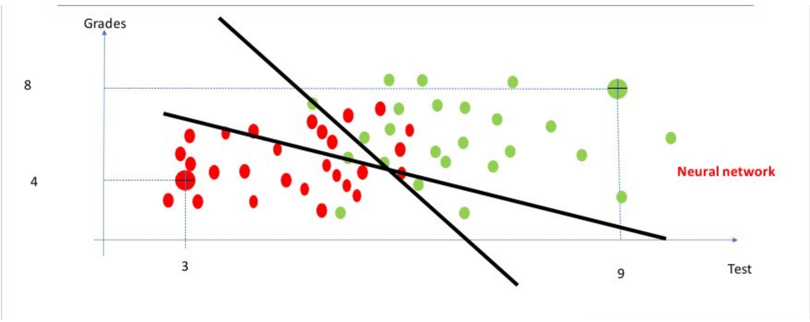
2019-08-27

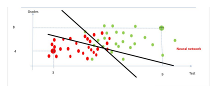
IML
└Supervised - Classification
  └KNN
    └KNN

KNN

Demo !

Page 92 :

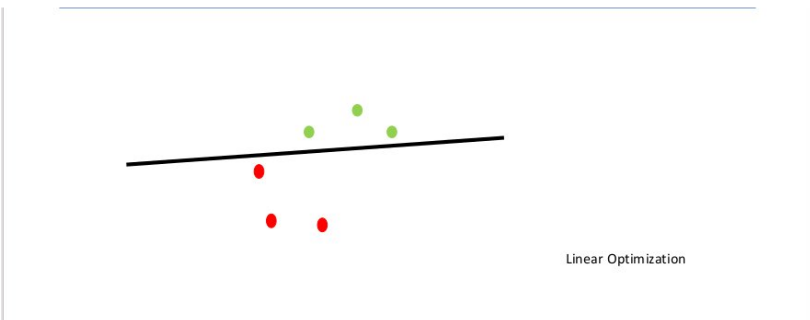


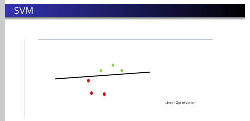


Page 93 :

What is a parametric machine learning algorithm and how is it different from a nonparametric machine learning algorithm ?
Assumptions can greatly simplify the learning process, but can also limit what can be learned. Algorithms that simplify the function to a known form are called parametric machine learning algorithms.
The algorithms involve two steps :
Select a form for the function. Learn the coefficients for the function from the training data.
Some examples of parametric machine learning algorithms are Linear Regression and Logistic Regression.
Algorithms that do not make strong assumptions about the form of the mapping function are called nonparametric machine learning algorithms. By not making assumptions, they are free to learn any functional form from the training data.
Non-parametric methods are often more flexible, achieve better accuracy but require a lot more data and training time.
Examples of nonparametric algorithms include Support Vector Machines, Neural Networks and Decision Trees.

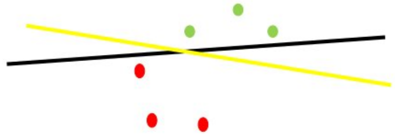
SVM



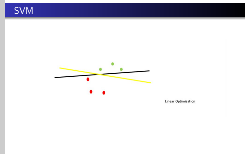


Page 94 :

SVM

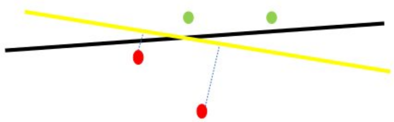


Linear Optimization



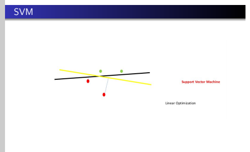
Page 95 :

SVM



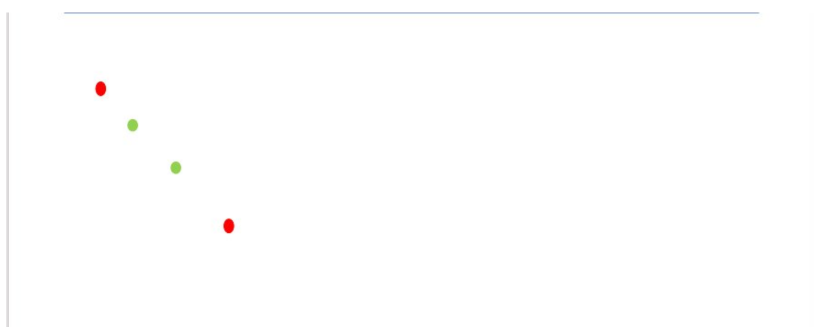
Support Vector Machine

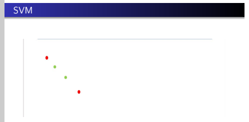
Linear Optimization



Page 96 :

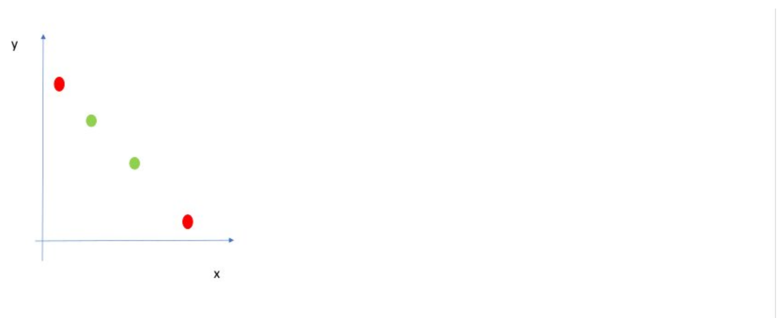
SVM

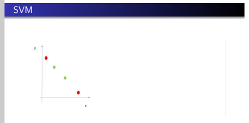




Page 97 :

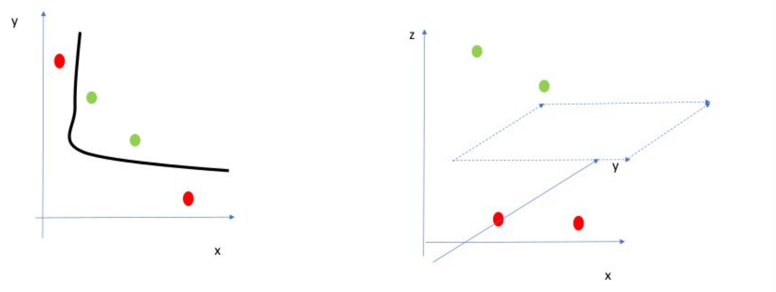
SVM

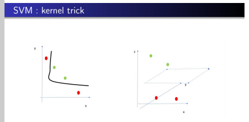




Page 98 :

SVM : kernel trick





Page 99 :

Support Vector Machines are perhaps one of the most popular and talked about machine learning algorithms.
A hyperplane is a line that splits the input variable space. In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1.
In two-dimensions, you can visualize this as a line and let's assume that all of our input points can be completely separated by this line.
The SVM learning algorithm finds the coefficients that results in the best separation of the classes by the hyperplane.
The distance between the hyperplane and the closest data points is referred to as the margin. The best or optimal hyperplane that can separate the two classes is the line that has the largest margin.
Only these points are relevant in defining the hyperplane and in the construction of the classifier.
These points are called the support vectors. They support or define the hyperplane.
In practice, an optimization algorithm is used to find the values for the coefficients that maximizes the margin.

1. Machine Learning
2. Supervised - Regression
   - Linear regression
   - Polynomial regression
3. Supervised - Classification
   - Naive Bayes
   - Decision Tree
   - Logistic regression
   - KNN
   - Neural network
   - SVM
4. Unsupervised - Clustering
   - k-means
   - Hierarchical clustering
   - Distance

Page 100 :

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*.

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.

Page 101 :

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.

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 102 :

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.

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 103 :

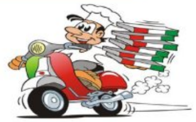
Example : pizza

Pizza chain

Optimal location ?

Page 104 :

Example : pizza

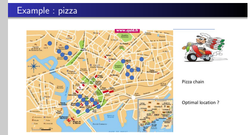


Pizza chain

Optimal location ?

2019-08-27

IML
└Unsupervised - Clustering
  └k-means
    └Example : pizza



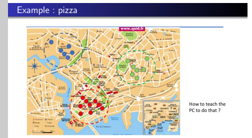
Page 105 :

Example : pizza

How to teach the PC to do that ?

2019-08-27

IML
└Unsupervised - Clustering
  └k-means
    └Example : pizza



Page 106 :

Example : pizza

Page 107 :

Example : pizza

Page 108 :

Example : pizza

Page 109 :

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:

  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 110 :

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 are 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 do not divide by 0
  if i_points:
    self.means[i] = vector_mean(i_points)
```


Page 111 :

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 112 :

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 113 :

K means

Demo !

Page 114 :

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.

k-means vs Hierarchical Clustering : HC do not need to specify k

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.

k-means vs Hierarchical Clustering : HC do not need to specify k

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	True	False	True	0	Yes
Chicken	True	True	False	False	2	No
Alligator	True	True	False	True	4	Yes
Frog	True	False	True	True	4	No
Salmon	True	True	False	True	0	No
Python	True	True	False	True	0	Yes

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

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	True	False	True	0	Yes
Chicken	True	True	False	False	2	No
Alligator	True	True	False	True	4	Yes
Frog	True	False	True	True	4	No
Salmon	True	True	False	True	0	No
Python	True	True	False	True	0	Yes

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

Distance : Euclidean

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

Distance : Euclidean

Page 117 :

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

Distance : Euclidean
Alligator is closer to a frog than a snake

Page 118 :

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 119 :

Machine Learning
Supervised - Regression
Supervised - Classification
Unsupervised - Clustering
k-means
Hierarchical clustering
Distance

Machine Learning

IML

Cédric Buche

ENIB

27 août 2019

Page 120 :