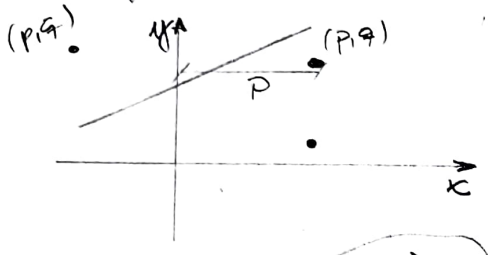
31/07/19

Data Scientist (Udacity)

→ linear regression
- Absolute Trick
- Square Trick

→ Absolute Trick

$y = w_1 \cdot x + (w_2) \longrightarrow y = (w_1 + p) \cdot x + (w_2 + 1)$



! We need small steps, let's add a learning rate (α)

$y = (w_1 + \alpha p)x + (w_2 + \alpha)$

---

what if the point is on the bottom of the line?

$y = (w_1 - \alpha p)x + (w_2 - \alpha)$

$y = (w_1 \pm p\alpha)x + (w_2 \pm \alpha)$

+ for negative slop
- for positive slop

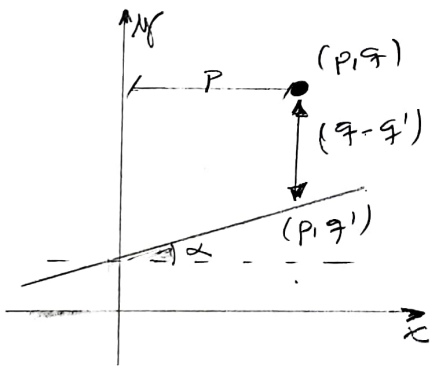
why we have p?

→ IF the point is in the left, p is negative and will make the slop negative, turning clockwise

→ IF p is small, the line will rotate less, IF is big, will rotate more to get closer.

! Does not consider the vertical distance

→ Square Trick



$y = w_1 x + w_2$

$y = [w_1 + p\alpha(q - q')]x + [w_2 + \alpha(q - q')]$

! Consider the distance of horizontal (p) and vertical (q-q').

Quiz

① $y = -0,6x + 4$

$(x, y) = (-5, 3)$

$\alpha = 0,1$

Absolute trick

$y = w_1 x + w_2$

$y = (w_1 + \alpha p)x + w_2 + \alpha$

$y = (w_1 - \alpha p)x + w_2 + \alpha$

$y = [-0,6 - 0,1 \cdot (-5)]x + 4 - 0,1$

$y = -0,6 + 0,5x + 3,9$

$y = -0,1x + 3,9$ //

(2) Square trick

$y = w_1 x + w_2$

$y = -0,6x + 4 \Rightarrow y = 7$

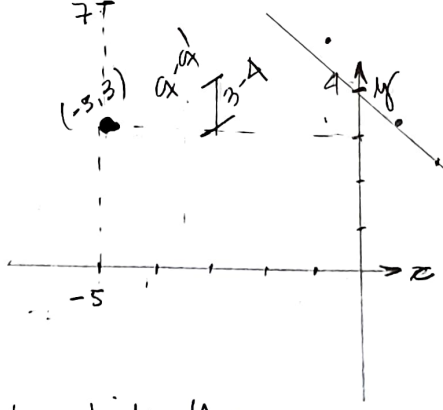
$(x, y) = (-5, 3)$

$y = [w_1 + \alpha(q - q')]x + [w_2 + \alpha(q - q')]$

$y = [-0,6 + 5 \cdot 0,01(3 - 7)]x + [4 + 0,01(3 - 7)]$

$y = (-0,6 + 0,2)x + 4 - 0,04$

$y = -0,4x + 3,96$



(2)

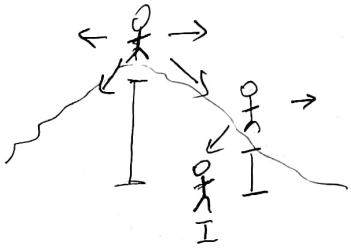
∇ the magnitude by which the intercept and slope change is dependent on how large the error prediction is.

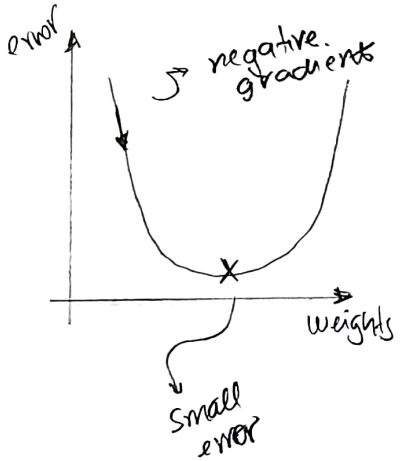
error → 7 - 3 = 4

y

→ linear regression

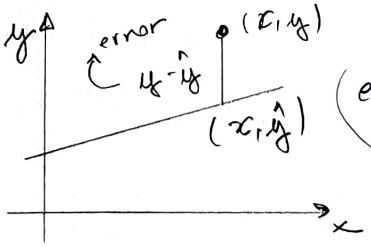
- gradient descent





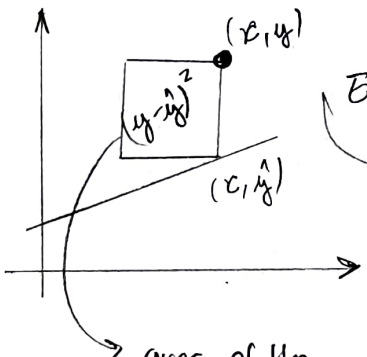
always (+) errors

- mean absolute error



$\text{error} = \frac{1}{m} \sum_{i=1}^{m} |y - \hat{y}|$

- mean squared error



$\text{Error} = \frac{1}{2m} \sum_{i=1}^{m} (y - \hat{y})^2$

Quiz

1) MAE

$y = 1.2x + 2$

(2, -2) (8, 14)

(5, 6)

(-4, -4)

(-7, 1)

$\frac{1}{5}[6,4 + 2 + 11,2 + 7,4 + 2,4] = 3,88$

2) MSE

$y = 1.2x + 2$

$(2,-2)\ (5,6)\ (-4,-4)\ (-7,1)\ (8,14)$

$$MSE = \frac{1}{2m} \cdot \sum_{i=1}^{m} (y - \hat{y})^2$$

$$= \frac{1}{10}\left[6,4^2 + 2^2 + 1,2^2 + 7,4^2 + 2,4^2\right]$$

$$= 10,892$$

01/08/19

2 algorithms → tricks → minimize erros) Are the same ?

$$MSE = \frac{1}{2}(y - \hat{y})^2$$

Gradient step $\Rightarrow$ $w_i \rightarrow w_i - \alpha \cdot \frac{\partial}{\partial w_i} \cdot error$

$$-(y - \hat{y}).x$$

$$\therefore w_i \rightarrow w_i + (y - \hat{y}).x.\alpha$$

Square trick

$$y = w_1 + p\alpha(q - \hat{q})x + w_2 + \alpha(q - \hat{q})$$

minimizing the mean square is equal to use the absolute or square trick

Exercise

Calculate the derivative of the error w.r.t w_2 and verify that it is precisely $-(y - \hat{y})$

$$Error = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y} = w_1 \cdot x + w_2$$

$$\frac{\partial Error}{\partial w_2} = \frac{\partial \frac{1}{2}\left(y - [w_1 x + w_2]\right)^2}{\partial w_2}$$

$$= 2 \cdot \frac{1}{2}(y - w_1 x - w_2).(-1)$$

$$= w_1 x + w_2 - y$$

$$= \hat{y} - y \quad ou \quad -(y - \hat{y})$$

→ Batch / Stochastic gradient descent (SGD)

- calculate the absolute / square trick for all the points, add them, and update the weights with the sum of the values.

→ mini batch gradient descent

- Split the data into small batches, then use each batch to update the weights.

Coding challenge

$$mse = \frac{1}{2}(y-\hat{y})^2 \qquad \hat{y} = w \cdot x + b$$

$$\frac{\partial mse}{w} = \frac{\partial}{\partial w}\frac{1}{2}(y-wx-b)^2$$

$$= (y-wx-b)(-x)$$

$$= (-y+wx+b)\cdot x \quad (wx+b = \hat{y})$$

$$= (-y+\hat{y})\cdot x \text{ ou } -(y-\hat{y})\cdot x$$

$$\frac{\partial mse}{b} = \frac{\partial}{\partial b}\frac{1}{2}(y-wx-b)^2$$

$$= (y-wx-b)(-1)$$

$$= -y+wx+b \quad (wx+b = \hat{y})$$

$$= -(y-\hat{y})$$

$$\therefore \boxed{\frac{\partial mse}{w} = -(y-\hat{y})\cdot x \qquad \frac{\partial mse}{b} = -(y-\hat{y})}$$

for the square trick:

$$y = w + \alpha p(y-\hat{y})x + b + \alpha(y-\hat{y})$$

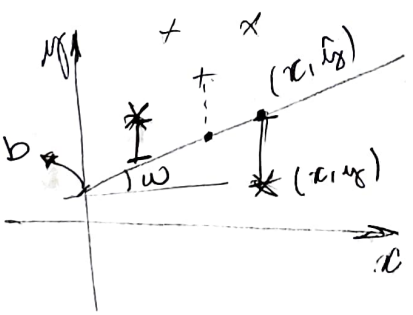
$$mse = \frac{1}{2m}\sum_{i=1}^{m}(y-\hat{y})^2 \quad (y: \text{points}, \hat{y}: \text{predicted } y)$$

$$= \frac{1}{2m}\sum_{i=1}^{m}(y-(w\cdot x+b))^2$$

$$y = \underbrace{w-\alpha x(y-\hat{y})}_{\text{new } w} \underbrace{-(y-\hat{y})\alpha}_{\text{new } b} + b$$

$$\hat{y} = wx + b \quad (x: \text{points})$$

$$\begin{bmatrix}\hat{y}_1\\ \hat{y}_2\\ \vdots\end{bmatrix} = w\cdot\begin{bmatrix}x_1\\ x_2\\ \vdots\end{bmatrix} + b$$



$\hat{y} = w \cdot x + b \Rightarrow \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}_{n\times 1} \begin{bmatrix} w_0 \end{bmatrix}_{1\times 1} + \begin{bmatrix} b \end{bmatrix}_{n\times 1}$

$n\times 1 + (1\times 1) \rightarrow n\times 1$

$error = y - \hat{y} \rightarrow \begin{bmatrix} error_0 \\ error_1 \\ \vdots \\ error_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix}$

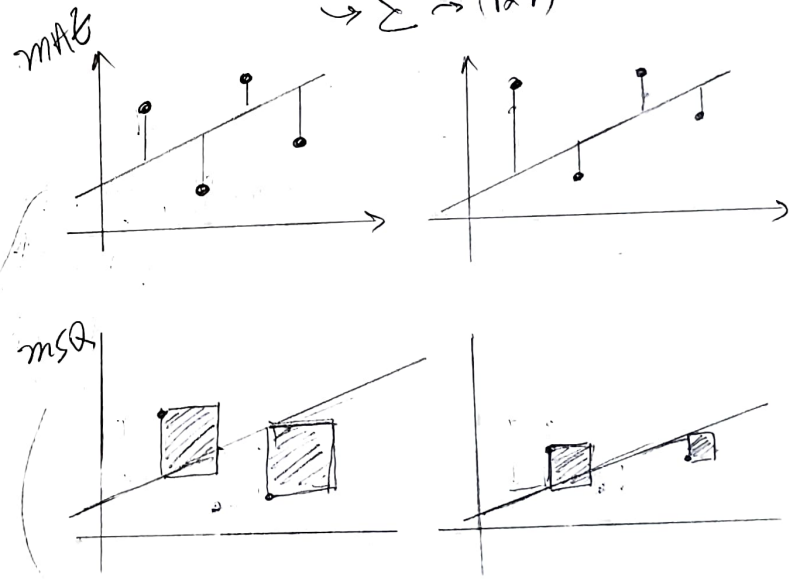
$\hat{w} = w + \alpha . x . error \Rightarrow [\hat{w}_0]_{1\times 1} = [w_1] + [\alpha] . \begin{bmatrix} error_0 \\ error_1 \\ \vdots \\ error_n \end{bmatrix}_{1\times 20} \begin{bmatrix} x_0 \\ x_1 \\ x_n \end{bmatrix}_{20\times 1}$

$\hat{b} = b + \alpha . error \Rightarrow [\hat{b}] = [b] + \alpha . \begin{bmatrix} error_0 \\ error_1 \\ \vdots \\ error_n \end{bmatrix}_{1\times 20} \rightarrow \sum \rightarrow (1\times 1)$

Quiz 2

$MAE = \frac{1}{2m} \cdot \sum_{i=1}^{m} |y - \hat{y}|$ (3-5 = 2, 3-2 = 1)

$MSE = \frac{1}{2m} \cdot \sum_{i=1}^{m} (y - \hat{y})^2$



Same MAE if the line goes up / down

→ MSE the error is minimal when the line is in the middle.

Polynomial Regression

$y = w_1 x^2 + w_2 x + w_0$

→ Scikit learn

- Polynomial features

? Predictor features

⇒ w and b for linear regression

! outcome feature

⇒ result y

→ Regularization

Adds model complexity into the error

Regularization L1. → sum the absolute coefficient values.

Regularization L2 → sum the squared coefficient values.

λ is used to give importance to the complexity error

- simpler models tend to generalize better than complex models.
- complex models has smaller errors, but tend to overfit.

$\uparrow\lambda \Rightarrow \uparrow$ complexity error

$\downarrow\lambda \Rightarrow \downarrow$ complexity error

error = model error + model complexity error

Regularization (L1) assigns zero to those coefficients with less importance

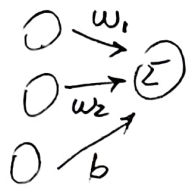
→ feature scaling

important for algorithms that use distance comparison (k-nn, svm, etc)

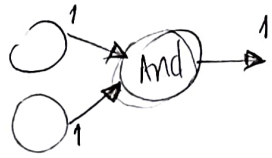
use standard scaler to fit and transform the coefficients

→ Classification

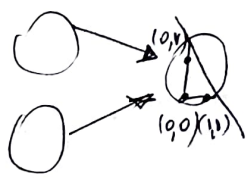
- perception



→ logical operators.



in	in	out
1	1	1
1	0	0
0	1	0
0	0	0



→ for and

$A \cdot w_0 + B w_1 = C$

$1.1 + 1.1 - 2 = 0$

$1.0 + 1.1 - 2 = 0$

$1.0 + 1.0 - 2 = -2$

$\therefore w_0 = 1$, $w_1 = 1$, $b = -2$

→ for or

$1.1 + 1.1 - 1 = 1$

$1.0 + 1.1 - 1 = 0$

$1.0 + 1.0 - 1 = -1$

→ not

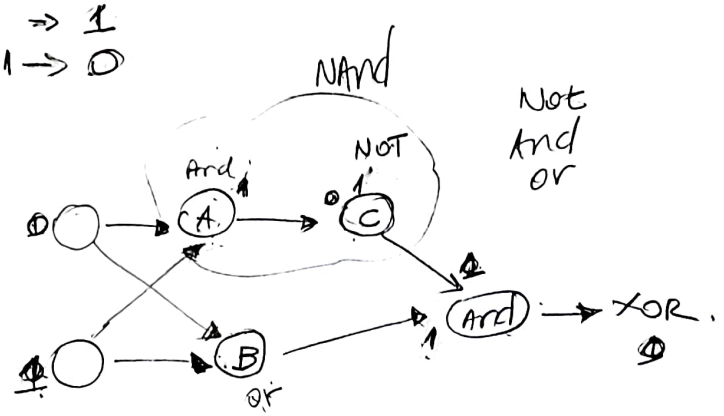
$A \cdot w_1 + B \cdot w_2 + b =$

$0 \cdot (-1) + 0 = 0 \Rightarrow 1$

$1 \cdot (-1) + 0 = -1 \rightarrow 0$

→ xor

1	1	0
1	0	1
0	1	1
0	0	0



$3x_1 + 4x_2 - 10 = 0$

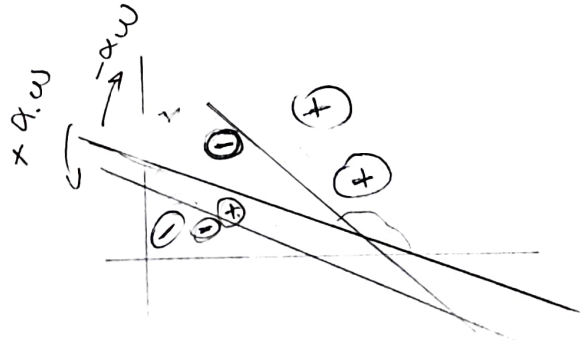
$P = (1,1)$

3	4	-10	
+ 1.0,1	1.0,1	1.0,1	
3,1	4,1	-9,9	≥ 0 ? não
0,1	0,1	0,1	
3,2	4,2	-9,8	≥ 0 não

3	4	-10	
+ 1	1	1	
4	5	-9	≥ 0 ? Sim

4,2 9,8
3,2 -7,4
~~-9,8~~ (2,4)
7,4

2,4 (3
8



(1x1) 1x2

$\hat{y} = x \cdot w + b$

$\hat{y} = x_1 w_1 + x_2 w_2 + b$

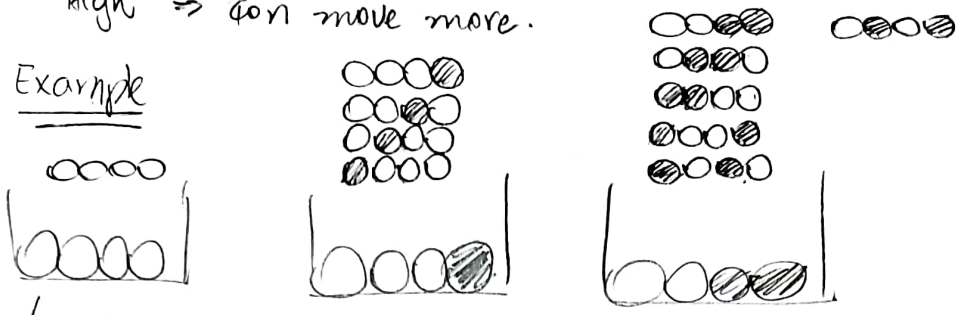
$1 \times 1 = 1 \times 2 \cdot 2 \times 1 + 1 \times 1$

Decisions trees.

→ Entropy ⇒ How much freedom has to move around.

High ⇒ can move more.

Example



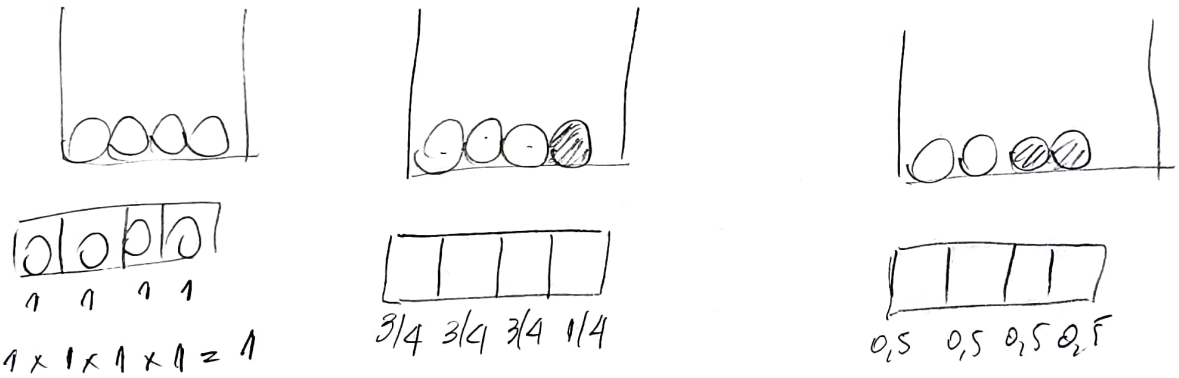
High knowledge Low

middle knowledge

High knowledge.

As more homogenius less entropy

As more knowledge, less entropy



1 1 1 1

$1 \times 1 \times 1 \times 1 = 1$

3/4 3/4 3/4 1/4

$0,75 \times 0,75 \times 0,75 \times 0,25 = 0,105$

0,5 0,5 0,5 0,5

$4 \cdot 0,5 = 0,0625$

Products are bad, many elements, numbers get too small

~ Sum is better

Use Logarithms

$$\log(a \cdot b) = \log a + \log b$$

$$3 \cdot \log(0,75) + \log(0,25) = -3,245.$$

$$\text{Entropy} = -\frac{m}{m+n} \cdot \log\left(\frac{m}{m+n}\right) - \frac{n}{m+n} \cdot \log\left(\frac{n}{m+n}\right)$$

Quiz

entropy

4 red balls

10 blue balls.

14 balls

$$\text{Entropy} = \underbrace{-\frac{4}{14} \cdot \log\left(\frac{4}{14}\right)}_{0,516} - \underbrace{\frac{10}{14} \cdot \log\left(\frac{10}{14}\right)}_{0,104} =$$

entropy = $-p_1 \cdot \log(p_1) - p_2 \cdot \log(p_2)$

Quiz

entropy = $-p_1 \cdot \log_2(p_1) - p_2 \cdot \log(p_2 - p_3 \cdot \log(p_3)$

5 blue
2 yellow
8 red.

entropy = $-\frac{3}{13} \cdot \log\left(\frac{3}{13}\right) - \frac{2}{13} \cdot \log\left(\frac{2}{13}\right) - \frac{8}{13} \cdot \log\left(\frac{8}{13}\right)$

$\underbrace{\frac{-3}{13} \cdot (-2,1139)}_{0,2307} - \underbrace{\frac{2}{13} \cdot (-2,7008)}_{0,1538} - 0,6153 \cdot (-0,7806)$

$= \underline{1,334}$

▷ Information gain

information gain = parent's entropy − 0,5 · (children 01 + children 02)

▷ Decision trees.
→ hyperparameters. { maximum depth.

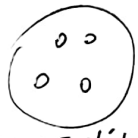
→ maximum depth

longest possible length between the root to a leaf.
A tree of lenght k can have at most 2^k leaves.

→ minimum number of samples to slip

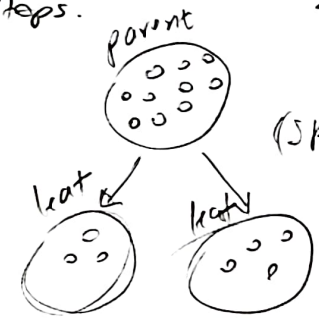
node ⇒ have at least min_samples_split samples to be large enough to split.

▽ If node has fewer than min_samples_split, the splitting process stops.



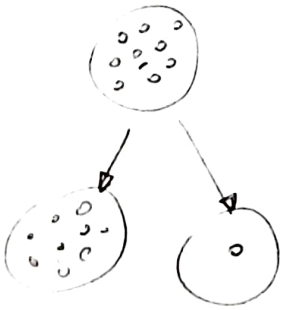
(no split)

min_samples_split = 5

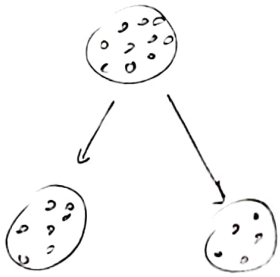


→ less than min_samples_split no control the minimum size of leaves.

→ minimal number of samples per leaf



min samples per leaf = 1



min samples per leaf = 5

▽ Avoid 99 samples in one leaf and 1 in another.

int : # of samples in a leaf

float : % of samples in a leaf

Quiz

Underfitting / overfitting vs Feature size

Small max depth → underfitting → too simple model.

Large max depth → overfitting → too deep trees can memorize the data

Small min samples per split → overfitting → complicated, highly branched tree, model memorized the data.

Large min samples per split → underfitting → not enough flexibility to build the tree.

Sklearn

Decision tree classifier

▽ Fitting the tree means finding the best tree that fits the data

→ titanic kaggle
using decision trees.

One-Hot-encoding ⇒ pandas.get_dummies(features).

! ↑training accuracy ↓ test accuracy = Overfitting

- reduce depth of the tree. ↓ → reduce memory and model complexity.
- increase min_samples_split ↑ → reduce number of branches.
- increase min_samples_leaf ↑ → reduce splitting, since it will stop splitting if the leaf has less than min_samples_leaf (will try different thresholds, though)

Naive Bayes

→ Bayes theorem: use more information to estimate a probability (a priori, a posteriori)

know	inferred
P(A)	P(A|R)
P(R|A)	

P(A|R) → event has occured.

→ probability that someone that wear red to be Alex / Brenda

→ probability Alex / Brenda wear red.

Alex: 3 day/week
Brenda: 1 day/week

w_1	w_2	w_3	w_4
A	A	A	B
A	A	A	B
A	A	A	B
A	A	A	B
A	A	A	B

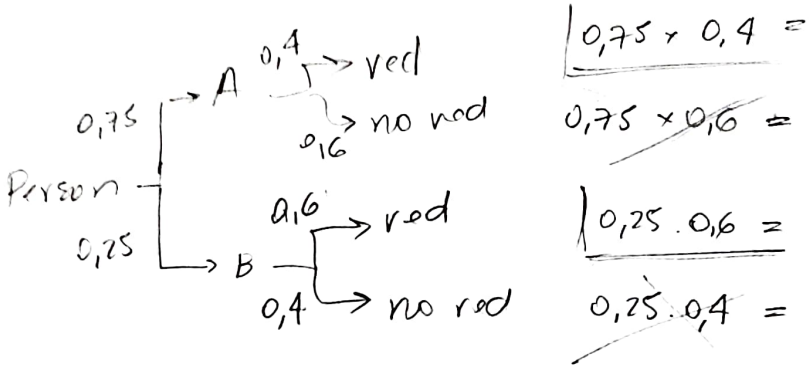
/// ⇒ red sweater.

Prior: P(Alex) = 0,75 (3/4)
P(Brenda) = 0,25 → 1/4

Person had a red sweater.
Alex wears 2x a week.
Brenda wears 3x a week.

Posterior: $P(Alex) = \frac{6}{9} \cdot \frac{2}{3}$

$P(Brenda) = \frac{3}{9} = \frac{1}{3}$



$0,75 \times 0,4 =$

$0,75 \times 0,6 =$

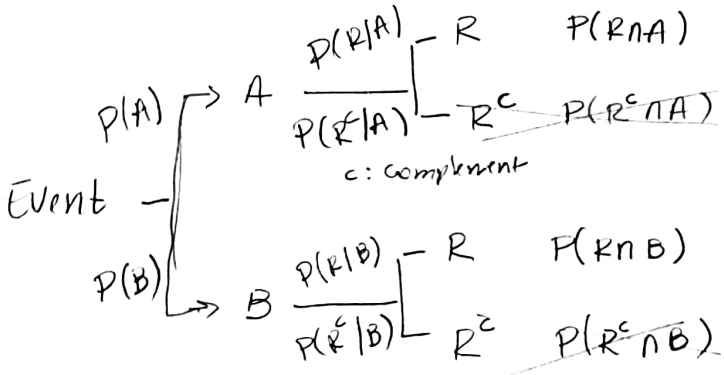
$0,25 \cdot 0,6 =$

$0,25 \cdot 0,4 =$

$$P(A|R) = \frac{0,75 \times 0,4}{0,75 \times 0,4 + 0,25 \times 0,6} = 0,67$$ → normalize

$$P(B|R) = \frac{0,25 \times 0,6}{0,75 \times 0,4 + 0,25 \times 0,6} = 0,33$$ → normalize

Bayes



$P(R \cap A) = P(A) \cdot P(R|A)$

$P(R \cap B) = P(B) \cdot P(R|B)$

should normalize to sum 1

$$P(A|R) = \frac{P(A) \cdot P(R|A)}{P(A) \cdot P(R|A) + P(B) \cdot P(R|B)}$$

$$P(B|R) = \frac{P(B) \cdot P(R|B)}{P(B) \cdot P(R|B) + P(A) \cdot P(R|A)}$$

posterior

Quiz

99 % accuracy

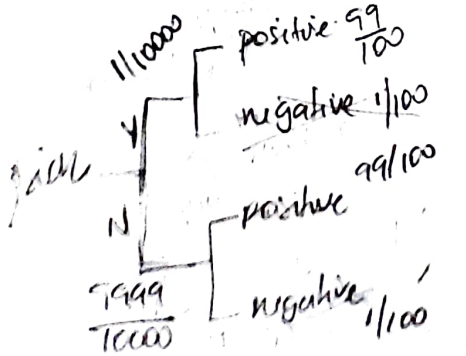
1 out of 10.000 ⇒ $P(\text{people sick}) = \frac{1}{10.000}$

$P(\text{sick}) = 0,0001$

$P(\text{positive} \mid \text{sick}) = 0,99$

$P(\text{positive} \mid \text{not sick}) = 0,01$

$P(\text{not sick}) = 0,9999$



$$\frac{\frac{1}{10.000} \cdot \frac{99}{100}}{\frac{1}{10.000} \cdot \frac{99}{100} + \frac{9999}{10000} \cdot \frac{1}{100}} = 0,98\ \%$$

$$P(\text{sick} \mid \text{positive}) = \frac{P(\text{sick}) \cdot P(\text{positive} \mid \text{sick})}{P(\text{sick}) \cdot P(\text{positive} \mid \text{sick}) + P(\text{not sick}) \cdot P(\text{positive} \mid \text{not sick})}$$

Quiz

$P(easy) = \frac{2}{5}$

$P(money) = \frac{3}{5}$

$P(easy \mid spam) = ?$

$$\frac{P(easy) \cdot P(spam \mid easy)}{P(easy) \cdot P(spam \mid easy) + P(\overset{not}{spam}) \cdot P(easy \mid not\ spam)}$$

NAIVE = ingenuo

➢ conditional probability

$$P(A \& B) = P(A \cap B) = P(A) \cdot P(B)$$

$$P(A \mid B) = P(A) \cdot P(B \mid A)$$

↳ we are considering the variables are independent

$P(spam \mid easy, money)$

$= P(spam) \cdot P(easy, money \mid spam)$

$P(easy \mid spam) \cdot P(money \mid spam)$ ⚠ Hot and cold are dependent

Naive assuption, since it might be dependents.

$P(spam \mid easy, money) = P(spam) \cdot P(easy \mid spam) \cdot P(money \mid spam)$

$= \frac{3}{8} \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{3}{36} = \frac{1}{12}$

$P(ham \mid easy, money) = P(ham) \cdot P(easy \mid ham) \cdot P(money \mid ham)$

$= \frac{5}{8} \cdot \frac{1}{5} \cdot \frac{1}{5} = \frac{1}{40}$

not the probability, are proportional because it is not normalized.

For 1/12 :

1/40 : ⊕ = 1

$$\frac{\frac{1}{40}}{\frac{1}{12} + \frac{1}{40}} = \frac{\frac{1}{40}}{\frac{10+3}{120}} = \frac{1}{40} \cdot \frac{3}{13} = \frac{3}{13}$$ ↳ not spam

$$\frac{\frac{1}{12}}{\frac{1}{12} + \frac{1}{40}} = \frac{\frac{1}{12}}{\frac{10+3}{120}} = \frac{1}{12} \cdot \frac{120}{13} = \frac{10}{13}$$ ↳ spam

- what if we have many features?

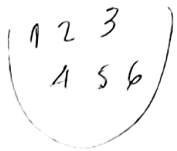
① $P(\text{spam} \mid \text{easy}, \text{money}, \text{cheap}, \dots) \propto P(\text{spam}) \cdot P(\text{easy} \mid \text{spam}) \cdot P(\text{money} \mid \text{spam}) \cdot P(\text{cheap} \mid \text{spam})$

② $P(\text{ham} \mid \text{easy}, \text{money}, \text{cheap}, \dots) \propto P(\text{ham}) \cdot P(\text{easy} \mid \text{ham}) \cdot P(\text{money} \mid \text{ham}) \cdot P(\text{cheap} \mid \text{ham})$

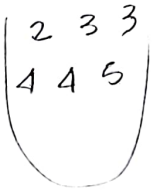
③ normalize

$\frac{P_1}{P_1+P_2}$ and $\frac{P_2}{P_1+P_2}$

Quiz



3 standard dices



2. no standard dices

1 die ⇒ was a 3.

$P(\text{standard} \mid 3) = ?$

$P(\text{std} \mid 3) = P(3) \cdot P(3 \mid \text{std})$ ①

$P(\text{not std} \mid 3) = P(3)\, P(3 \mid \text{not std})$ ②

① $\underbrace{\left(3 \cdot \frac{1}{6} + 2 \cdot \frac{2}{6}\right)}_{P(3)} \cdot \underbrace{\left(3 \cdot \frac{1}{6}\right)}_{P(3 \mid \text{std})} = \left(\frac{1}{2} + \frac{2}{3}\right) \cdot \frac{1}{2} = \left(\frac{3+4}{6}\right) \cdot \frac{1}{2} = \frac{7}{12}$

② $\left(3 \cdot \frac{1}{6} + 2 \cdot \frac{2}{6}\right) \cdot \left(2 \cdot \frac{2}{6}\right) = \left(\frac{1}{2} + \frac{2}{3}\right) \cdot \frac{2}{3} = \frac{7}{6_3} \cdot \frac{2}{3} = \frac{7}{9}$

③ $P(\text{std} \mid 3) = \frac{\frac{7}{12}}{\frac{7}{12} + \frac{7}{9}} = \frac{\frac{7}{12}}{\frac{21+28}{36}} = \frac{7}{12} \cdot \frac{36}{49_7} = \frac{3}{7} = 0{,}428$

1,2,4	3
4,3	3
4,1	2
2,1	2
1,1	

9 of 36

→ Hands-on

Naive Bayes

Example: P(threat // age, bag, nervous...)

considered independents

↳ Ex: Age and nervousness

- Spam / not spam based on joint probabilistic distributions of certain other events (presence of some words in this case).

nervous kid does not seems to be a threat, but a man has a bigger likelihood to be a threat.

→ Interesting points

→ pandas reads tab spaced table use read_table, params:
- sep = '\t'
- names = ['colA', 'colB'].
  ↓ indicate cols names.

→ transform categorical vars to integers

using map function

df:label.map({'ham': 0, 'spam': 1}).

assign a dict and transform the key to the value.

→ features using BoW

count tokenizer ⇒ separates the words into tokens (IDs)
count the some IDs
- transform to lowercase
- remove puctuation (uses regex)
- remove stop words (am, an, the...)

→ Removing punctuation from string ⇒ maps punctuations to none.

import string

string.translate(translator) → translator = str.maketrans('', '', string.punctuation)

→ Python Counter from Collections lib

Counter(_list) ⇒ {"hello": 1, "how": 1, "are": 1}.

→ Scikit learn Count Vectorizer

- sklearn.feature_extraction.text import CountVectorizer

→ count_vector.fit(documents) → transform to a document-term matrix

count_vector.get_feature_names() → feature names.

row	word 1	word 2	word 3...
doc #1			
doc #2			
doc #3			
doc #4			

→ frequency of word.

for this, use count_vector.transform(documents)

use toarray() to print

- pd.dataframe(doc_array, columns = get_feature_name

large dataset → BoW is not ideal

words like is, the, an, would appear a lot and mess with the features.

→ use tf idf instead.

> Bayes implementation

priors → probabilities we are aware / was given

posteriors → want to compute using priors

P(D) = probability of having diabetes = 0,01

P(pos) = probability of getting ⊕ test result

P(neg) = " " ⊖ test result

Sensitivity / True positive rate

P(pos/D) = probability of getting ⊕ on test, given you have diabetes = 0,9

P(neg/~D) = " " ⊖ on test given you don't have diabetes = 0,9

Specificity / True neg rate

Bayes

$$P(A/B) = \frac{P(A).P(B/A)}{P(B)}$$

(posterior) $P(D/Pos)$

given $P(B/A) = P(Pos/D)$

(prior) $P(A) = P(D)$ → given, $P(B) = P(Pos)$

$$\rightarrow P(D/Pos) = \frac{P(D).P(Pos/D)}{P(Pos)}$$ → sensitivity

$$P(pos) = [P(D).\ sensitivity] + [P(\sim D).(1 - specificity)]$$

$P(pos/D)$ $\quad$ $P(Neg/\sim D)$

$$\rightarrow P(\sim D/pos) = \frac{P(\sim D).P(pos/\sim D)}{P(pos)}$$ → $1 - P(Neg/\sim D)$ = specificity

---

▷ Naive Bayes in scikit-learn.

→ multinomial Naive Bayes ⇒ good for classifying discrete values

→ gaussian naive Bayes ⇒ better for continuous data as it assumes input data has a Gaussian (normal) distribution.

→ Evaluation model.

- Accuracy: how often the classifier makes the correct prediction

$$Acc = \frac{\#correct}{\#\ total\ pred.}$$

- Precision: what proportion of messages were classified as spam, actually were spam.

$$Pr = \frac{\#\ true\ pos \rightarrow spam.}{(\#\ true\ pos + \#\ false\ pos)}$$

(all classified as spam.)

- Recall (positivity): proportion of messages that actually were spam were classified by us as spam

$$\text{Recall} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false negatives}}$$

→ predicted as spam.

all words that are spam.

* Accuracy is not a good metric when the data is highly unbalanced or skewed.

⚠ Naive Bayes is very good on handling a large number of features

---

→ Support Vector machines (SVM)

- Increase boundary and reduce error.

$$\text{margin} = \frac{2}{|w|} \qquad |w| = \sqrt{x^2+y^2}$$

$$\text{Error} = |w|^2 = x^2+y^2$$

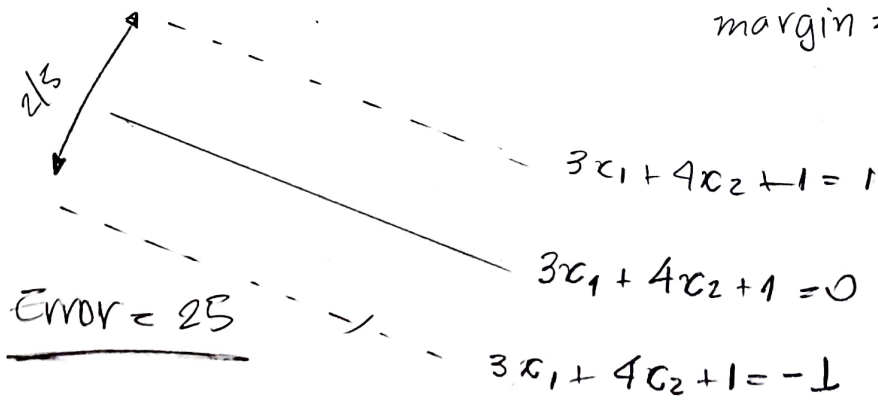
Ex: $w = (3,4)$ $b = 1$

$w_1x_1 + w_2x_2 + b = 0$

$3x_1 + 4x_2 + 1 = 0$

$|w|^2 = 3^2+4^2 = 25 = \text{Error}$

$\text{margin} = \frac{2}{\sqrt{25}} = \frac{2}{5}$

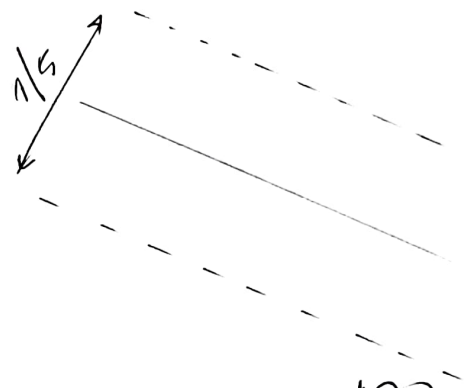


Error = 25

what if weights are :

$w = (6, 8)$

$b = 2$



$w_1 x_1 + w_2 x_2 + b = 0$

$6x_1 + 8x_2 + 2 = 0.$

$\text{Error} = |w|^2 = 6^2 + 8^2 = 100$

$\text{margin} = \frac{2}{\sqrt{100}} = \frac{1}{5}$

Error = 100

Summary:

$w = (3,4)$	Error = 25	$w = (6,8)$ Error = 100
$b = 1$ $\Rightarrow$	margin = 2/5	$b = 2$ Margin = 1/5

→ margin error is a norm of the vector w^2

> Error function

Error = C. Classification error + margin Error

How to minimize? → Gradient descent ∇

↓ C ⇒ large margin
↑ Classification Errors

↑ C ⇒ Small margin
classifies well.

Obs: $\text{margin} = \frac{2}{|w|}$ ∴ $\text{margin} = \frac{2}{\sqrt{\text{Error}}}$

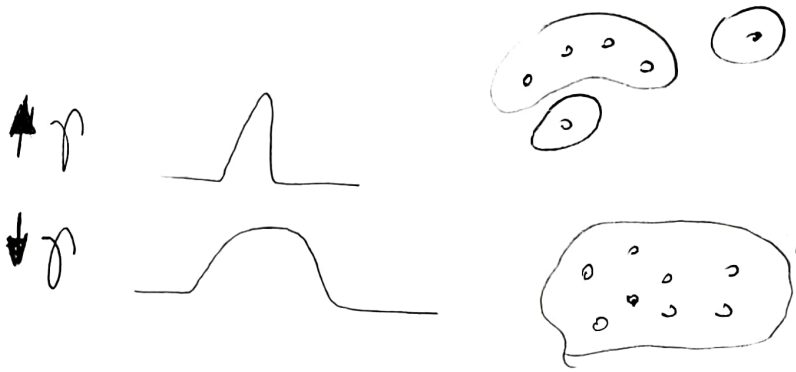
$\text{Error} = |w|^2$

C = hyperparameter, use grid search for optimal value

↑ error ↓ margin
↓ error ↑ margin

> Using Kernels.

→ Radial Basis Function (RBF)



~ overfit

~ underfit

Ensembles

1. Bagging → Get all friends and hear their opinion and combine (Average/ voting)
2. Boosting → Select best friend for each topic. (explore friend's strengths)

combine models for better prediction

Weak learner (friends) and Strong learner (super genius)

Bias and Variance are important.

curvar dobrar

→ High bias ⇒ does not do a good job of bending to the data.

→ High variance → model changes drastically to meet the needs of every point.

linear model

decision tree

good bias + good variance = better model

- use central limit theorem to reduce bias / variance
- introduce randomness into ensembles
  - ↳ Avoid overfit
    1. Sample data with replacement
    2. Use features only from a subset

Random forest → is a ensamble algorithm

- Deasions tree → memorize the data tends to overfit a lot

- Random forest is a set of deasion tree models and a voting

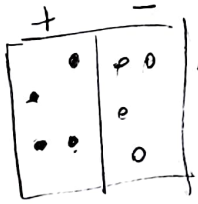
→ Bagging

1. Split the data
2. Train in different models
3. Combine them

→ Boosting (Ada Boost)

$$y = \ln\left(\frac{x}{1-x}\right)$$ x = accuracy.

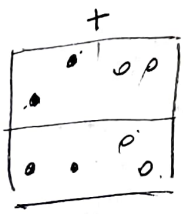
(→ weight)



Acc = 1
(Truthful)

$$w = \ln\left(\frac{1}{1-1}\right)$$

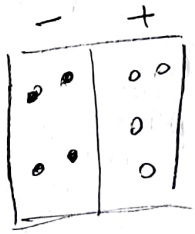
$w = \infty$



Acc = 0,5
(Random)

$$w = \ln\left(\frac{0,5}{0,5}\right) =$$

$w \to 0$

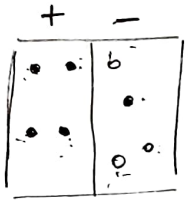


Acc = 0
(liar)

$$w = \ln\left(\frac{0}{1}\right)$$

$w = 0$

Quiz • ⊕ ○ ⊖

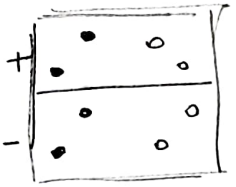


$Acc = \frac{7}{8}$

$w = \ln\left(\frac{x}{1-x}\right)$

$= \ln\left(\frac{718}{1-718}\right)$

$= 1.95$

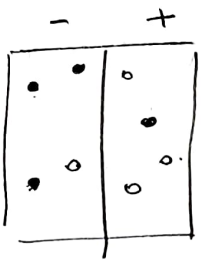


$Acc = \frac{4}{8} = \frac{1}{2}$

$w = \ln\left(\frac{x}{1-x}\right)$

$= \ln\left(\frac{0,5}{0,5}\right)$

$= 0$



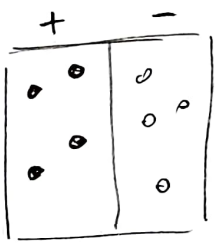
$Acc = \frac{2}{8} = \frac{1}{4}$

$w = \ln\left(\frac{x}{1-x}\right)$

$= \ln\left(\frac{0,25}{0,75}\right)$

$= -1.1$

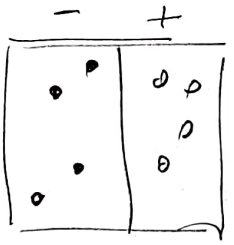
what about this?



$Acc = \frac{8}{8} = 1$

$w = \ln\left(\frac{1}{1-1}\right) =$

$\ln\left(\frac{1}{0}\right) \sim \infty$



$Acc. = 0$

$w = \ln\left(\frac{0}{1}\right)$

$\ln\left(\frac{0}{1}\right) \sim -\infty$ → do the complete opposite

→ Project (01) - Finding donors.

I chose the following classifiers:

- Logistic Regression
- Random Forest
- Adaboost

) Ensembles

layman's terms
- describe complex things with words to make everybody understand.

classifier	Strenghts	weakness
Logistic Regression	• linear model • light • easy to expand data using SVG	• hard to capture non-linear data relationship
Random Forest	• can model non-linear boundaries • parallel training each decision tree • reduce overfitting	• hard to interpretate
Addboost	• use diffirent classifiers as weak classifiers • simple to implement • resistant to overfitting	• training time is consuming • imbalanced class problems. • needs a termination condition

> STEPS to train a classifier

① Data exploration (analyse features, categoricals, numericals)

② Preparing data

→ skewed features → Logarithmic transformation (reduce distance between max and min values)

→ Normalization → MinmaxScaler make feature values between 0-1. make the features treated equally by supervised learners. (keep the distribution shape).

→ Treat non-numeric features. → pandas.get_dummies to transform categorical data into numerical data using one hot encoding

→ Shuffle / split data → scikit train_test_split helps to split the data.

③ Naive Predictor Performance

→ check a Naive Predictor performance → consider the model always predicts 1 calculates Accuracy and Fscore. (No 0, so FP=0, FN=0)

④ Using / selecting Supervised models.

→ create a training and prediction pipeline → generic function to receive a classifier and the train/test data. Fit the model, predict and calculate accuracy, fscore, training time, inference time

⑤ Analyse classifier results

→ understand the metrics for each classifier and test/train step. → IF train accuracy is ↑ and test accuracy is ↓ = OVERFITTING

Check the best Fscore for the specific problem. (some need high recall, others high precision)

⑥ Model tuning

→ Find best parameters for the model → use grid search to try the model with different parameters to find the best.

Use fbeta score as grid search optimization target