KNN with one independent variable.

Steps

which bucket → if dropped at 300px

① * Drop at random. at point (p)
② * abs(p - 300px) → abs.
③ * sort results from least to greatest
④ * look at top K records. & see which is most common
↓ sort this.

eg.

drop pos.	abs(pos - 300)	bucket
0	300	#1
150	150	#2
200	100	#3

after sort.

325	25	#4	if K=3
275	25	#3	K=3
375	75	#4	4 → common
		#3	

Lodash //

(300 - pt)

```
[
  [10, 0.5, 16, 1],
  [280, 0.5, 16, 4],
  [350, 0.5, 16, 4],
  [600, 0.5, 16, 5],
]
```

→

```
[
  [290, 1]
  [100, 4]
  [50, 4]
  [300, 5]
]
```

sort →

50	4
100	4
290	1
300	5

to get closest

slice(0, K) K=3

50	4
100	4
290	1

most common number ←

{ 4: 2, 1: 1 }

to pairs → [4: 2], [1: 1]

sort by

[1: 1], [4: 2]

Prediction was Bad.

① Adjust parameters. (K value alter ~~ke~~)
② Add more features to explain analysis (like bounciness)
③ change prediction point.
④ Maybe drop position doesn't matter.

① Finding ideal K.

Record a bunch of data points

split (at Random)

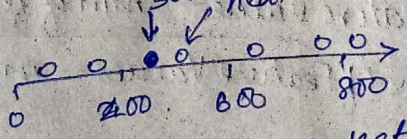
↙ test ↘ training.

for each test, run KNN using training data

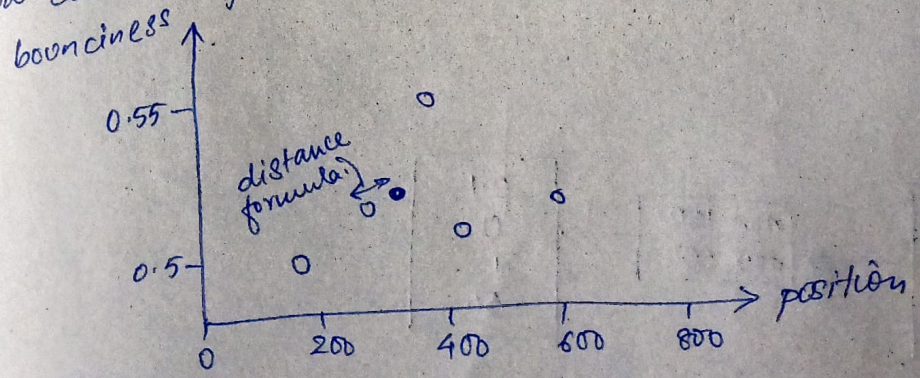
Does the result of KNN equal the 'test' ?

② Adding more features.

initially with only 1 feature it was like ~~x = a~~ one-D



Now considering more than 1 feature. (say 2)



Now in this method of using distance formula

position range 0 – 780 → $(300-400)^2 = 10000$

bounciness range 0.5 – 0.55 $(0.55-0.5) = (0.05)^2$

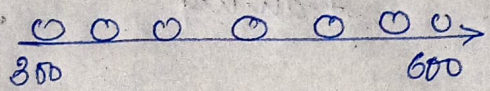
$= 0.0025$

doesn't cause any significant change

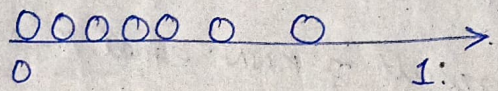
∴ we need to normalise our data

all numbers range from zero to 1.

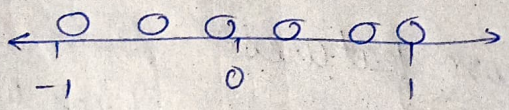
original



normalization



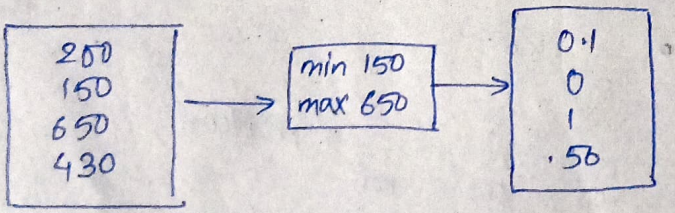
Standardization



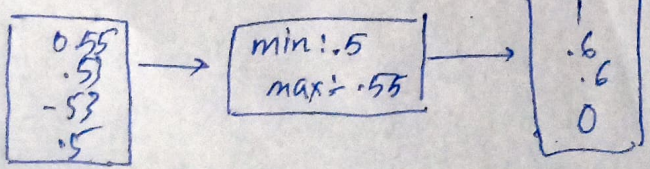
$$\text{Normalised dataset} = \frac{\text{feature value} - \text{min of feature Value}}{\text{max of Feature Values} - \text{min of Feature Value}}$$

Example.

Drop position



Bounciness



Changes to drop position → Predictable changes to output.

Ball bouncines → changes o/p but not predictably.

Feature Selection

Deciding which features to include in analysis

How to select ?

```
[
  [300, 4]
  [350, 5]
  [416, 9]
]
```

with drop → KNN → 30%.

```
[
  [.5, 4]
  [.52, 5]
  [.53, 4]
]
```

bouncines → KNN → 10%.

Tensorflow 's # Job (For Us)

* make working with numbers in array of arrays simpler terms.

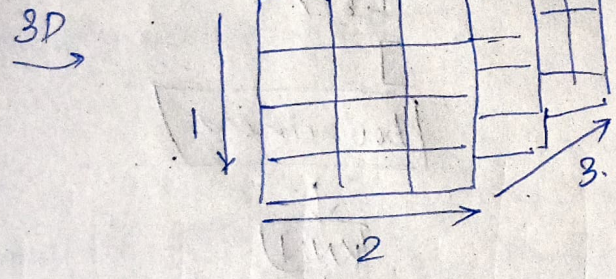
tensors — is object that wraps a collection of numbers.

eg.
```
[
  [300, 0.5, 16, 4],
  [350, 0.53, 16, 3],
]
```

```
[
  [10, 2, 6],
]
```

```
[
  [
    [5, 10, 15]
  ]
]
```

1D 2D 3D



Shape — How many records in each dimension?
basically length considered from out to inwards.

[5, 10, 17]
= [3]

```
[
  [5, 10, 17],
  [18, 4, 2].length
].length
= [2, 3]
```

```
[
  [
    [5, 10, 17].length
  ].length
].length
= [1, 1, 3]
```

2D being most common that we will work with.

```
[
  [300, 0.4, 16, 4],
  [350, 0.4, 25, 5],
]
```

[2, 4]
- 2 → rows
- 4 → columns

tensorflow examples:

```
const data = tf.tensor([1, 2, 3]);
const otherData = tf.tensor([4, 5, 6]);
data.add(otherData); // [5, 7, 9]
```

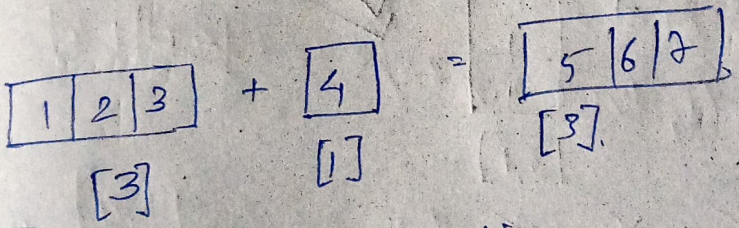
identical indices operate.

creates a new data & not alter the original.

```
1 2 3
4 5 6
+ ------
6, 7, 9
```

data.sub(otherData) for subtraction
.mul(1)
.div()

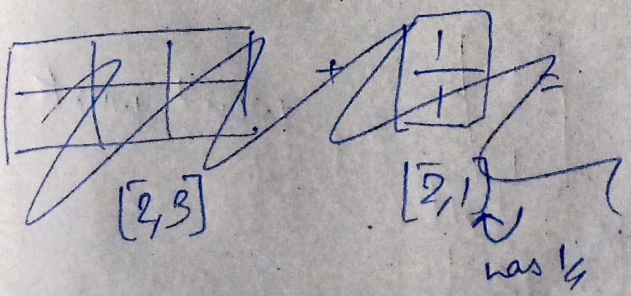
"shape needs to match"

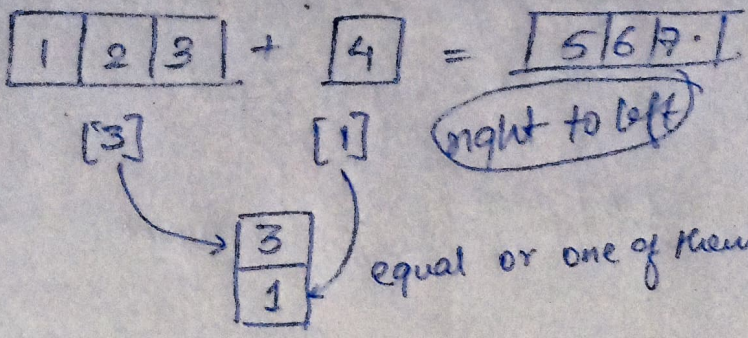


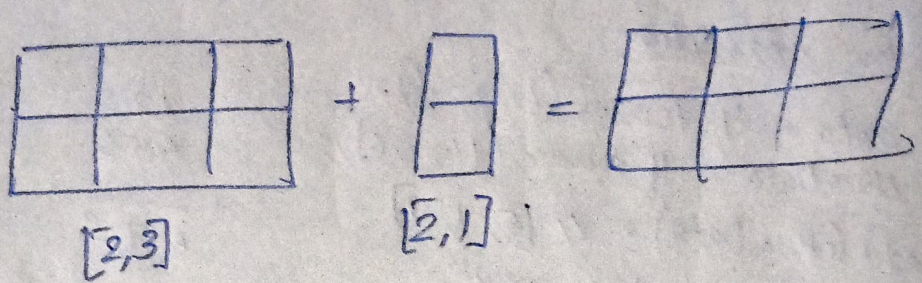
this is called broadcasting.

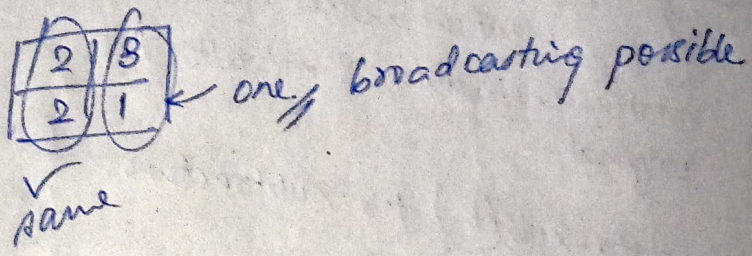
condition :-

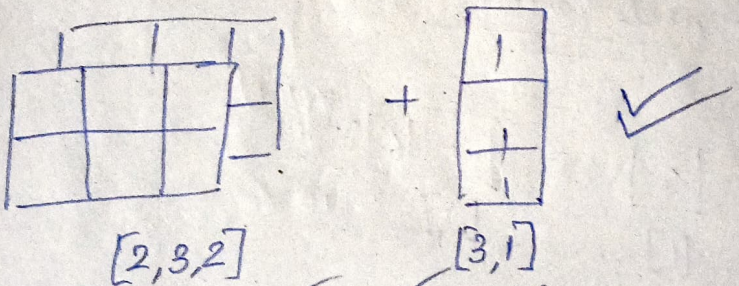
from right to left the shapes are equal or one has '1'

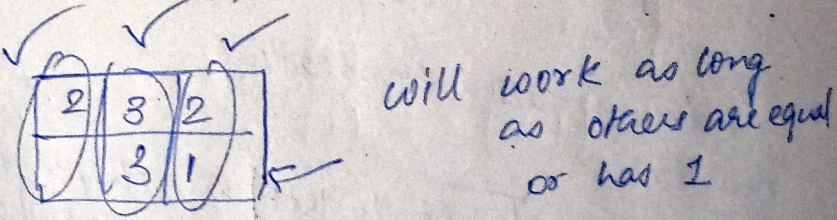



1 2 3 + 4 = 5 6 7
[3] [1]
right to left
3
1
equal or one of them is equal to 1


[2,3]
[2,1]


2 3
2 1
one // broadcasting possible
same


[2,3,2]
[3,1]


2 3 2
3 1
will work as long as others are equal or has 1

+
= won't work
[2,1]
2 3 2
2 1
X

initialising a tensor

```
const data = tf.tensor([
    [1, 2, 3]
]);
data.shape  // [1,3]
```

to print the data in tensor

clg(data.print()) ✗ no need to clg.

data.print() ✓

data[0] ✗ data.get(0) 1D. ↳ index

data.get(0,0) 2D

↓ ↓ ↳ c

to get a specific column only.

data.slice(startIndex, size) ↳ not zero-indexed

[0, 1] [6, 1]

↓ ↓ ↳ width of slice like no. of rows.

c no of rows.

data.shape → [row, col]

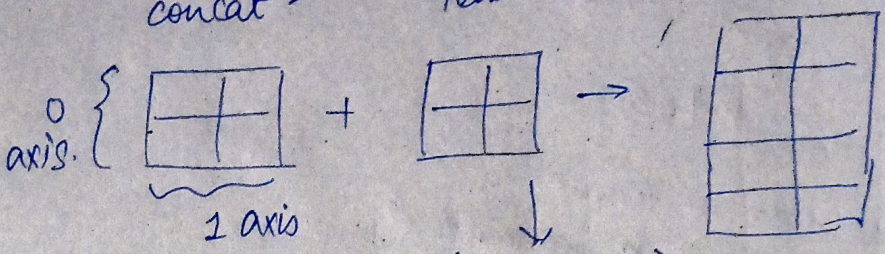
so data.slice([0,1], [data.shape[0], 1])

or data.slice([0,1], [-1, 1])

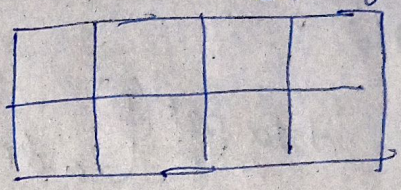
-1 ~~not~~ gives all.

joining tensors.

concat → (identical dimensions)
tensorA.concat(tensorB)



tensorA.concat(tensorB, 1) → default = 0



8.

[4,3]

data.sum(); sums all elements

data.sum(1); sum of each row → [4]

(0); columns.

gives 1D data

[4,3]

So, data.sum(1).concat(data2) ✗ error

second args of sum, to keep dimensions same

data.sum(1, true).shape → ~~[4,1]~~ [1,4]

Alternate way

data.sum(1).expandDims(1) → [1,4]

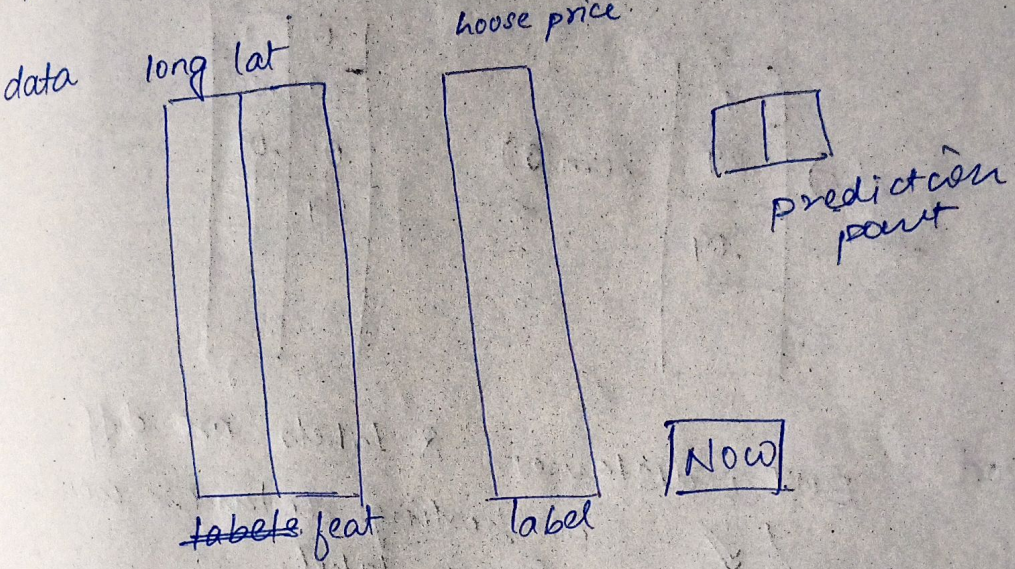
(1) → [4,1]

Plinko → Classification

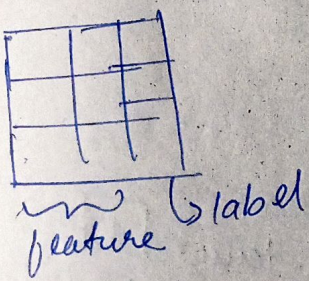
House price → Regression.

KNN algorithm

* distance b/w features and prediction point
* sort from closest point to greatest
* take the top K records
* average the label value of those top K records.



64



Change in data str

$$dist = \sqrt{(lat - lat)^2 + (long - long)^2}$$

lat	long
-121	47
-121.2	46.3

sub | -121 | 46 |

0	1
.2	.5
0	.3

↓ pow(2)

0	1
.04	.25
0	.09
1.29	

sum(a) ←

1
.29
.09

pow(.5) ←

1
.53
.03

↓ sort.

BUT ① distances & labels are diff.
concat ✓ so sorting distances won't change label.
② tensor can't be sorted.

↓

[10,2] ↓↓

10

	1	200
ten_sort	.53	210
	.03	250

unstack

[

tensor | 1 | 200 |

tensor | .53 | 210 |

]

↳ returns array

↓

oo use sort //

sort ((a,b) ⇒ a.arrsync()[0] - b.arrsync()[0])

top K-records.

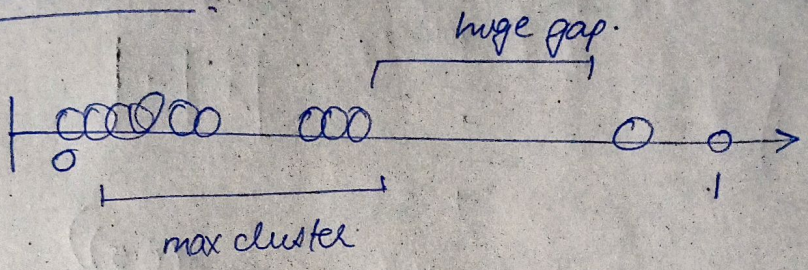
array →

- slice (0, K)
- reduce ((acc, ten) ⇒ acc + ten.get(1),0) / K//

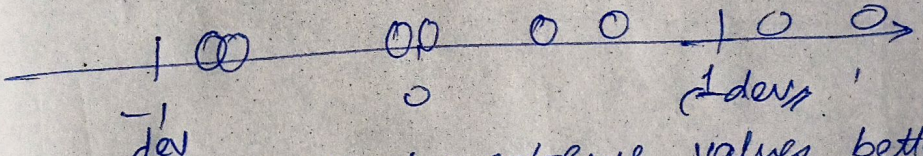
avg of prices

error %

Normalization



Standarization



→ as it accounts for extreme values better than normalization

Ham

Standardization

$$\frac{\text{value} - \text{Avg}}{\text{St.d Deviation}} = \sqrt{\text{variance}}$$

tf. moments (tensor to calc.) → { mean : variance :

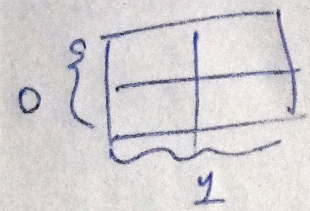
↓

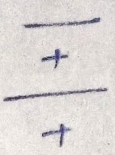
does total (all elements)

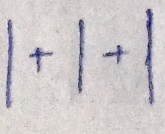
tf. moments (data, 0 or 1)

0 → row

1 → column







data.sub(mean).div(variance.pow(0.5))

Linear Regression

create a relation with independent variable & dependent variable

$$\text{mean squared error} = \frac{1}{n}\sum_{i=1}^{n}(\text{guess}_i - \text{actual}_i)^2$$

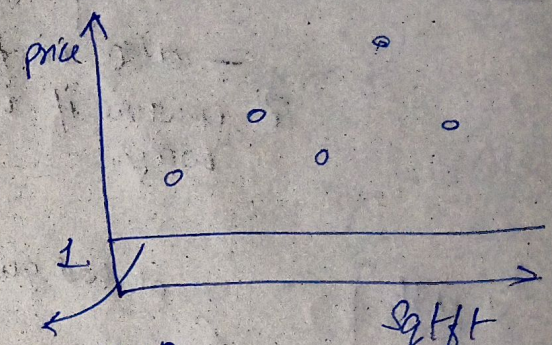
↓

tells you about how bad, your equation is

$y = mx + c$

$y = 0x + 1$

Sqft	Guess	Actual
800	1	200
950	1	230
1040	1	245
1120	1	274
1250	1	259
1350	1	262



$y = mx + c$

$y = 1$ //

$y = 0x + 1$

$$\text{mean sq error} = \frac{(1-200)^2 + (1-230)^2 + \cdots (1-262)^2}{6}$$

MSE

$= 60132$ //

New guess $= y = 0x + 200$.

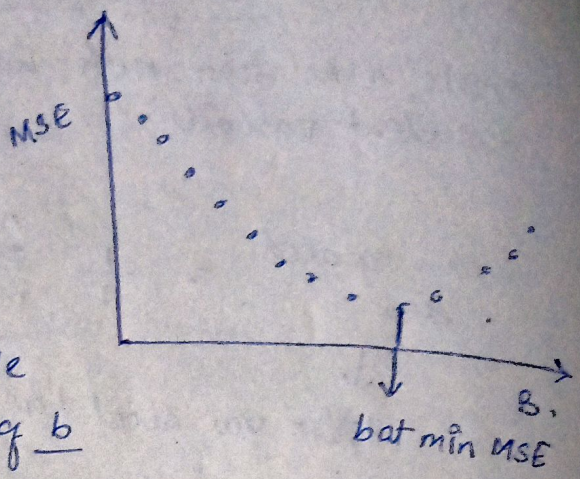
MSE $= 4287$.

$y = mx + b$.

we need to guess this better

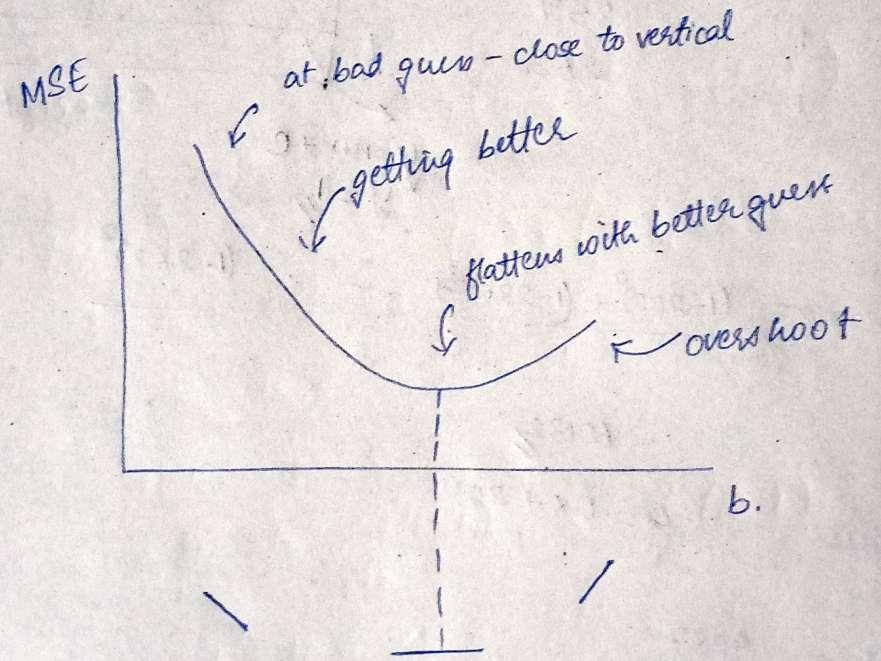
lets focus on guessing B now.

b	MSE
1.	60132
200	4287
⋮	⋮
⋮	⋮



now using for loop to provide MSE for range of value of b

Problems :- we DON'T KNOW ~~value of~~ range of b.
- also in for loop we do say b++;
? unsure if it is supposed to increment that way.



Slopes to left
'b' is less than optimum

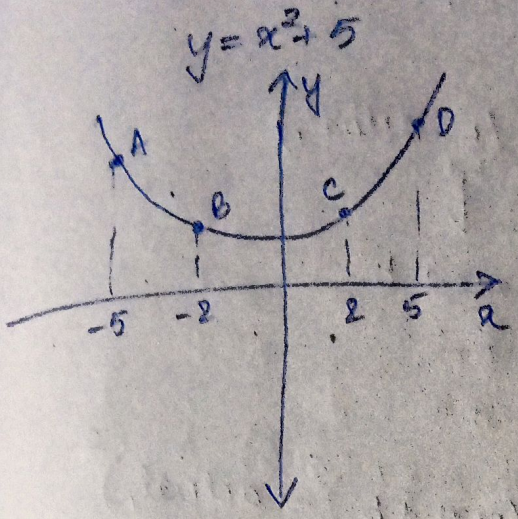
Slopes to right
'b' is more than optimum

more slope — farther
less slope - closer

Therfore slope/rate of change is valuable

Derivatives

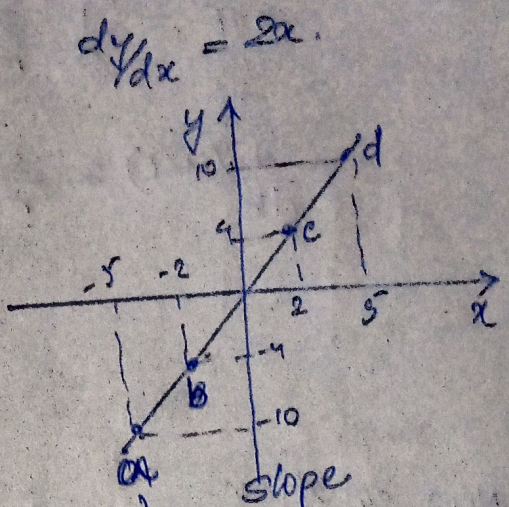
The derivative of an equation gives a new equation that tells us the slope at any location.



y – MSE

x – b

$\frac{d(MSE)}{db} = \frac{2}{n} \Sigma(b - actual)$



Slope at A

y-axis – slope $\frac{d(MSE)}{db}$

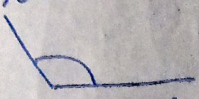
x-axis – a // b //

Gradient Descent

Steps –

- Pick 'b' any value.
- calculate slope of MSE vs b.
- if slope is very very small – work done!
- multiply slope with an arbitrary value called learning rate
- sub that from b.

a – learning rate



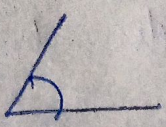
-ve slope (m)

$b - (a \times m)$

~~$b + a \times m$~~

$b + a|m|$

right



m = +ve

$b - a|m|$

~~b – at~~

towards left

Now to include m as well.

b alone =

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(guess_i - actual_i)^2$$

$$\frac{d}{db}(MSE) = \frac{2}{n}\sum_{i=1}^{n}(b - actual_i)$$

$b \rightarrow mx + b$

$\downarrow$

0.

New equation

$$MSE = \frac{1}{n}\sum_{i=1}^{n}((mx_i + b) - actual_i)^2$$

$$\frac{d}{db}(MSE) = \frac{2}{n}\sum_{i=1}^{n}(mx_i + b - actual) \times 1$$

$$\frac{d}{dm}(MSE) = \frac{2}{n}\sum_{i=1}^{n}(x_i) \times (mx_i + b - actual)$$

Gradient Descent

- Pick b & m
- calculate $\frac{d(MSE)}{db}$ & $\frac{d}{dm}(MSE)$
- if they are small — we are done
- multiply both with learning rate (α)
- sub results $b - \alpha\frac{d(MSE)}{db}$ & $m - \alpha\frac{d(MSE)}{dm}$.

goo.gl/f4KdZj

here say when we apply, there is unchanging value of b, but looks fine for m

meaning :- scaling is required.

Project. cars csv.

miles per gallon = $m * (\text{car horsepower}) + b$.

structure

class LinearRegression:
- gradientDescent() → run one iteration of GD and update m & b
- train() → run GD till we get good value of b & m
- test() → evaluate accuracy using testcases
- predict() → make prediction using our 'm' & 'b'

$$\frac{d(MSE)}{db} = \frac{2}{n} \sum_{i=1}^{n} ((mx_i + b) - \text{actual}_i)$$

$$\frac{d}{dm}(MSE) = \frac{2}{n} \sum d(x_i)((mx_i + b) - \text{actual}_i)$$

($mx_i + b$ = current guess)

Tensorflow implementation

P.T.O

Matrix Multiplication

Shape [4 | 2] · [2 | 3]

must be same

Now how is related to ML

$$[c \ d] \begin{bmatrix} a \\ b \end{bmatrix} = ie * a + - d * b.$$

Now implementation: here 2 is insignificant as it can be put in learning rate as well. as $mx_i + b$] we just need slope

Slope of MSE w.r.t both b & m.

$$\frac{\text{features}^T * ((\text{features} * \text{weights}) - \text{labels})}{n}$$

General eq.

weights: M & b in a tensor

$[Mx_i + b * 1]$

1 2 ; 1 2
3 4
1+b 2

horsepower

x_1
x_2
x_3
⋮ [6, 1]

weights

[m / b]

[2, 1]

Matrix multiply.

equal

some appended on left hand side

x_1 1
x_2 1
x_3 1
1
⋮

[6, 2]

[n / b]

[2, 1]

current Guesses.

$x_1 m + b$
$x_2 m + b$
$x_3 m + b$.

[6, 1]

$x_1 m+b$
$x_2 m+b$ sub actual → differences / diff − 2
$x_3 m+b$
[6,1]

(mx_i+b) − actual i

done

horsepower

x_1 1
x_2 1 × differences / [6,1]
[6,2]

transpose → actually $[x_1\ x_2\ \ldots]$ $\Sigma(-) = \Sigma x_i(\,)$

$\begin{bmatrix} x_1 & x_2 & x_3 & \ldots \\ 1 & 1 & 1 & \end{bmatrix}$ [2,6] [6,1] $\begin{bmatrix} \text{diff } 1 \\ d-2 \\ d-3 \\ \vdots \end{bmatrix}$

$\begin{bmatrix} x_1 d-1 + x_2 d-2 \ldots \\ d-1 + d-2 \ldots \end{bmatrix}$ · $\Sigma x_i(mx_i+b-\text{actual})$
$\Sigma(mx_i+b-\text{actual})$
↓ divide by n.
gradients or slopes

slopes * learning rate =
weights − =

Gauge accuracy using 'coefficient of determination' R^2.

$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$ → sum of squares residual
SS_{tot} → " " " total

higher – the better

$SS_{tot} = \sum_{i=1}^{n} (\text{actual} - \text{average})^2$ ↓ of test.

$SS_{res} = \sum_{i=1}^{n} (\text{actual} - \text{predicted})^2$

test()

1	x1
1	x2
1	x3
1	x4

b
m

$b + mx_4$
our prediction

prediction ↗

$SS_{res} = \sum (\text{label} - \text{predicted})^2$

Bad accuracy

Standarization * apply same mean & var. in test as well

$\frac{\text{value} - \text{mean}}{\text{std dev}}$ → $\sqrt{\text{variance}}$

Multivariate.

$MPG = b + (m1 * weight) + (m_2 * disp) + (m3 * horsep)$

$y = b + (m1 * x_1) + (m_2 * x_2) + (m_3 * x_3)$ ..

features weights disp horse

1.	x_1	x_2	x_3.
1	x_1	x_2	x_3
1	x_1	x_2	
1	x_1	x_2	
1	x_1	x_2	
1	x_1	x_3	

matrix

$\begin{bmatrix} b \\ m1 \\ m^2 \\ m3 \end{bmatrix}$

4	1

6	4

guies the prediction or given.

[6, 1]

feat T.

1	1	1	1	1	1
x_1	x_1	x_1	x_1	x_1	x_1
x_2	x_2	x_2	x_2	x_2	x_2
x_3					

4, 6

4, 1

$\begin{bmatrix} b + (m1 * x_1) \cdots \\ \\ \\ \\ 6, 1 \end{bmatrix}$

Optimising learning rate

methods:-

- Adam
- RMS Prop
- Adagrad
- Momentum.

155
145
300

Custom Learning Rate Optimization

Steps:-

- with every iteration of GD, calculate the exact value of MSE and store it
- 1 iteration later look at MSE_{old} & MSE_{new}.
- if $MSE_{new} > MSE_{old}$ learning Rate / 2.
- if $MSE_{new} < MSE_{old}$ ↑ the LR by 5%. i.e 1.05 LR

Vectorized MSE

mean sq error $= \frac{sum(((features * weights) - labels)^2)}{n}$

Batch Gradient Descent

- guess starting value of B & M
- calculate slope of MSE using a portion of observations in feature set & current M/B values
- multiply slope by learning rate
- update B & M

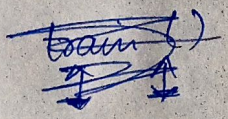
Stochastic Gradient Descent

- guess a starting value of B & M
- calculate slope of MSE using 1 obs. in feature set & current M/B values.
- multiply the slope by the learning rate
- update B & M

Gradient Descent	Batch Gradient Descent	Stochastic GD
uses entire feature set	couple observation	ONE observation

b4

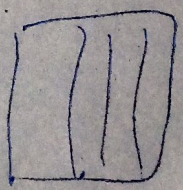
train() ⟶ gradientDescent()
↓ ↓
features labels



features ↓ train() ← labels ↓
batch of features / batch of labels → gradientDescent()

carve them to little batches

BGD — size in each batch
— total no. of batches.



~~here~~ GD → each iteration = entire data set is gone through
~~BGD → each iter~~ MSE is taken based on full then weights are adjusted
HUGE computation
slow.

BGD — calculates MSE in batches & weight adjustments.
it's like 100 pushups in 5 workout sessions
rather than 100 at one go.

Now predicting.

take an array of arrays

[
  [horse, weight, displac] } multi predict
  [" " "]
]

then matmul with weights gives the guesses.

Logistic Regression:

Predicts discrete value (classification)

Binary Logistic Regression - Binary classification.

Pass / Not Pass

Span / Not Span.

Customer accepts / Customer declines

Apple / Android

Q. Based on a person's age, do they prefer READ BOOKS (RB) or WATCH MOVIES (WM)

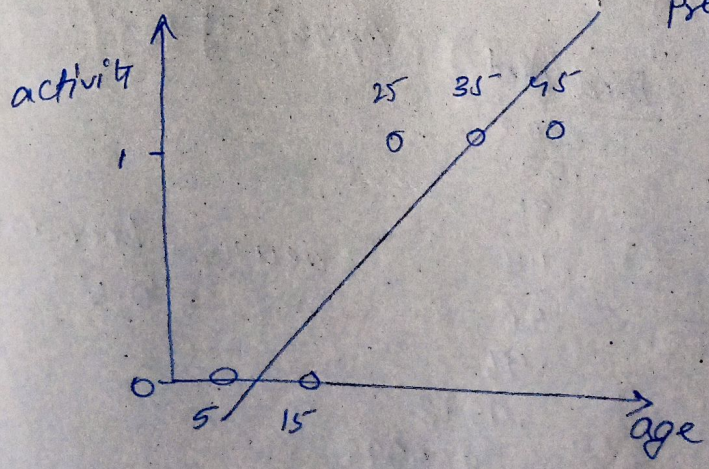
Sample

Age	Activity	Encoded activity
5	WM movies	0
15	WM movies	0
25	RB books	1
35	RB books	1
45	RB books	1

↳ convert this column to numerical data

Now we formulate an equation

Pref = 0.029 age - 0.15



If linear regression is used.

Age	actual	Bed
5	0	-0.0011 X
15	0	.2969
25	1	.5945
35	1	.8923
45	1	1.19 } X.
50	1(?)	1.48 }

results b/w 0 & 1 are OK
outside 0 & 1 X

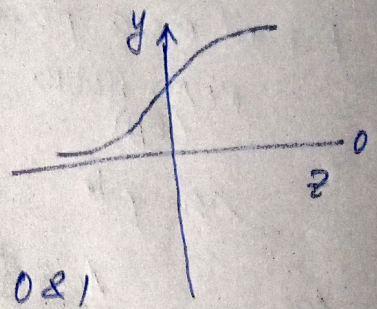
The eq. we will use.

Probability of preferring books.

$$\frac{1}{1+e^{-(mx+b)}}$$

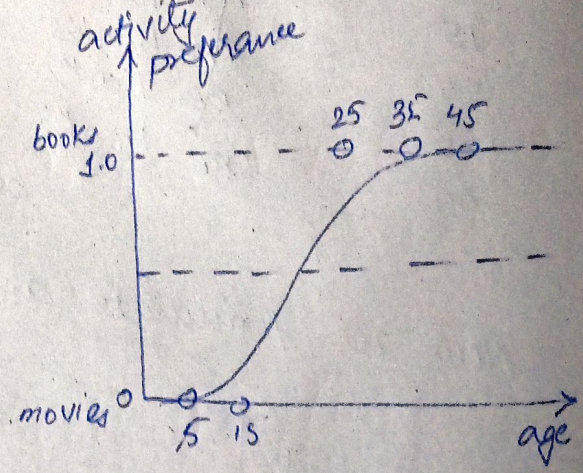
$e = 2.718$.

sigmoid eq. = $\frac{1}{1+e^{-z}}$



output is b/w 0 & 1

Age	predicted
5	.0002
15	.07
25	.96
35	.9998
45	.999999
55	.99999999



Age	Predicted.
5	.0002
15	.07
18	.81
19	.44
20	.58
25	.96
35	.9998
45	.999999
55	.99999999

decision boundary = 0.5.

Logistic Regression Gradient Descent

Steps:

Linear Regression:

* Encode label values as either '0' or '1'
* guess M, B.
* cal. slope of MSE using all obs. & ~~old~~ current M/B.
* multiply * learning rate
* update B and M.

Given a vehicle's weight, horsepower & engine displacement

PASS or NOT PASS.

smog emissions check

linear — MSE (metric of how bad we guessed)

For logistic — Cross Entropy

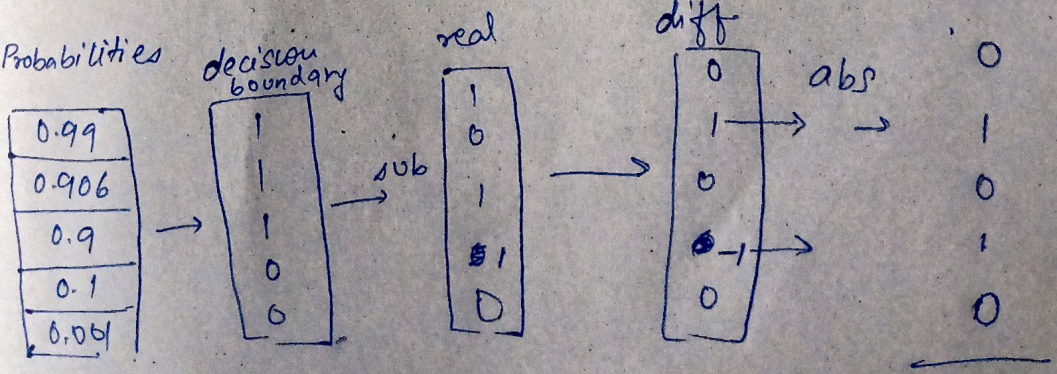
$$-\frac{1}{n}\sum_{i=0}^{n} \text{actual} \cdot \log(\text{guess}) + (1-\text{actual}) \cdot \log(1-\text{guess})$$

guess

$$\text{guess} = \text{sigmoid}(mx+b)$$

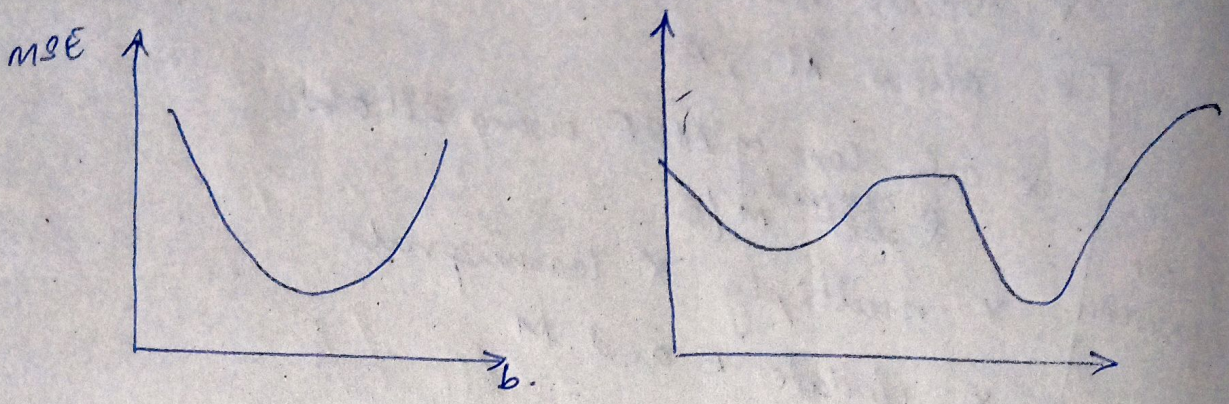
for logistic regress. $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

makes no sense



no. of errors = 2 times / total obs.

Why not MSE used with sigmoid ? although it still gives good accuracy.



not sigmoid

Convex function = ✓

└ one min. value

X.

has local minimas

- and hence, cross entropy

cost History → gives history of ~~~~ CEntropy

Cross Entropy.

$$\left(-\frac{1}{n}\right) \sum_{i=0}^{n} \text{actual} \cdot \log(\text{guess}) + (1-\text{actual}) \cdot \log(1-\text{guess})$$

Vectorized

$$-\frac{1}{m} \cdot \left(\text{actual}^{T} \cdot \log(\text{guess}) + (1-\text{actual})^{T} \log(1-\text{guess})\right)$$

Multinomial Logistic Regression.

* based on hrs driven what car — Luxury, Sedan, Truck, Compact

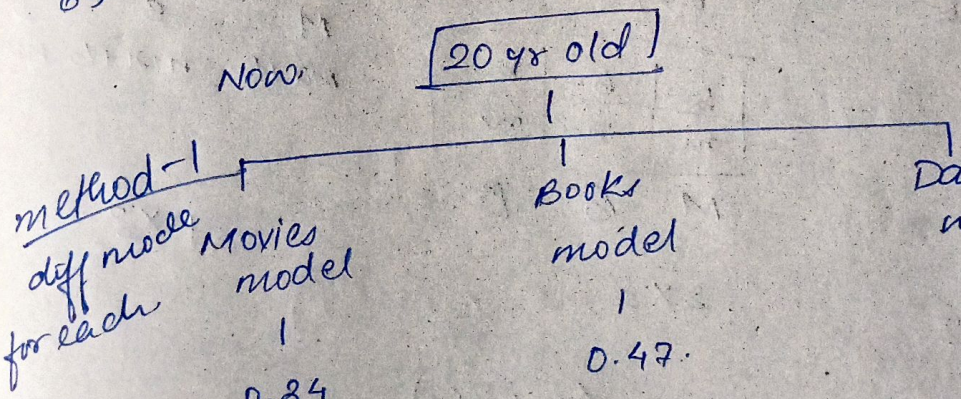
* salary basis phone — Apple, Android, Feature, Windows

Person's age :- Books, Dance, Movies.

~~Method-1~~.

age	Pref. Activity	encoded Movies	encoded Books	encoded Dance
5	movies	1	0	0
15	movies	1	0	0
25	books	0	1	0
85	books	0	1	0
45	books	0	1	0
55	dance	0	0	1
65	dance	0	0	1

Now, [20 yr old]



intense for multiple say 20 diff ~~to~~ options

method-2

combine weights

M	B	D
b	b	b
m	m	m

combine tensors.

Movie	Book	Dance
1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	0	1
0	0	1

Slope of Cross Entropy w.r.t M and B

= Features^T * sigmoid (Features * weights) - labels

Features^T : n × P

weights: 2 × 3 (M B D)

Predictions: P × 3 (M B D; $mx+b$, $mx+b$, $mx+b$)

Sigmoid (predic) . sub (labels) → [P × 3]

β^T

9 Given: horsepower, weight, displacement

fuel efficiency → high, medium or low.

mpg → somewhat same as fuel efficiency.

↳ this is given as ~~int~~ float values. so first convert it to.

0-15	15-30	30+
low	med.	high

low mid high

what if we get output [1 1 0]

conditional

marginal probability distribution → considers 1 possible output case in isolation.

conditional probability distribution → considers all possible output cases together

we need marginal probability

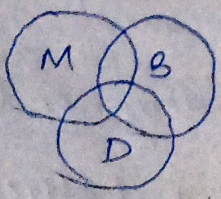
what we have

movies	books	dancing
0.34	.47	.67

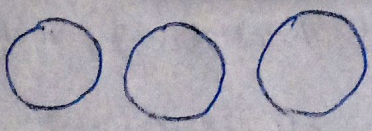
$0.34 + 0.47 + .67 = 1.48$ ← greater than 1

its like ~~A∪B~~ A∩B cases would have been needed to be reduced.

this indicates that say movies probability is not just ONLY movies but some intersection of movies & something



$M + B + D > 1$



$M + B + D = 1$
(like dice)

so it would be nice to have Σ prob $= 1$

like

ONLY –	M	B	D.
	.28	.32	.39

does this

Sigmoid → considers ONE possible output → Marginal Probability Distribution

Softmax → Conditional Probability Dist.
↳ considers all possible output cases together.

Softmax Equation

Probability of being '1' label rather than '0' label $= \dfrac{e^{mx+b}}{\sum_{K=0}^{K} \left(e^{mx+b_K}\right)}$

Individually:

	movies	books	dancing
	$mx+b = 15$	$mx+b = 23$	$mx+b = 22$
	$\dfrac{e^{15}}{e^{15}+e^{23}+e^{22}}$	$\dfrac{e^{23}}{e^{15}+e^{21}+e^{22}}$	$\dfrac{e^{22}}{e^{15}+e^{21}+e^{22}}$
Together	.002	.73	.26

so we need to use softmax on each row.

	M	S	D
softmax.	$m x + b$	$m x + b$	$m x + b$
	0.819	0.422	0.258

but here how do we take which one?

decision boundary = 0.5 (say)

when all = 0.

we need max so we use

argMax()

↳ returns the INDEX with highest value.

test():

L.	M	H
1	0	0
1	0	0
0	1	0
0	1	0
0	0	1
0	0	1

argMax

(Pred)

col. qmax
0
0
1
1
2
2

not Equal

label col qmax	Plain ≠
0	0
0	0
1 →	0
1	0
0	1
2	0

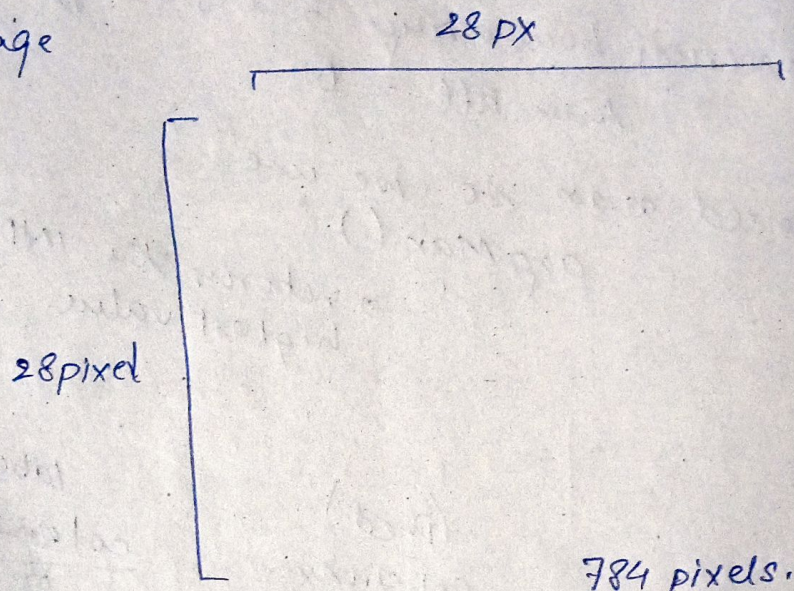
sum.

we get error

Handwritten Digit Recognition.

Given the pixel intensity values in an image, identify whether the character is a hand-written 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

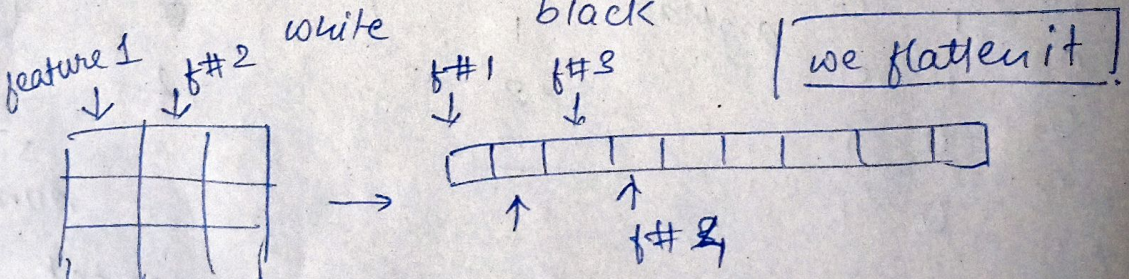
Each image



The input is grayscaled

0 — 255

white black



here we have multiple features

784 features [Px-1 Px 2 Px 3 . . .]

64 → [horsepower, displa, weight]

next for each label
we divide them into columns.

en→	0	1	2	3	4	5	6	7	8	9
7								1		
4					1					
2			1							
9										1

Now, the <u>problem</u>
for scaling, we are using standarization.
where formula = we get variance as zero in some cases.

$$\frac{?}{0} \quad \frac{value - avg}{\sqrt{variance}} \rightarrow NAN.$$

eg features [[0, 20, 40],
[0, 20.5, 80],
[0, 20.3, 90]].

variance [0, 0.0422, 466.667]

feat.sub(mean).div (vari.pow(0.5)) = E

[1, -1.2977, -1.3887]
[1, 1.1 ...]
[1, ...

$\frac{something}{0}$ = 1 → browser
↓
Nodejs = NAN

so what we do is

variance.cast(bool) → [0, 1, 1]
.logicalNot() → [1, 0, 0]
.cast('float32') → [1, 0, 0] → add above

so basically turns '0' → 1 in variance

another problem

Cross entropy formula has log( )

log(0) & log(−x) → NAN.

Trick to solve it

log(0). add (1e−7) → $10^{-7} \neq 0$

so solved

and such a small value won't make a diff in o/ks.