Idea — 1:

How about polarizer/Analyzer

Introduction

To

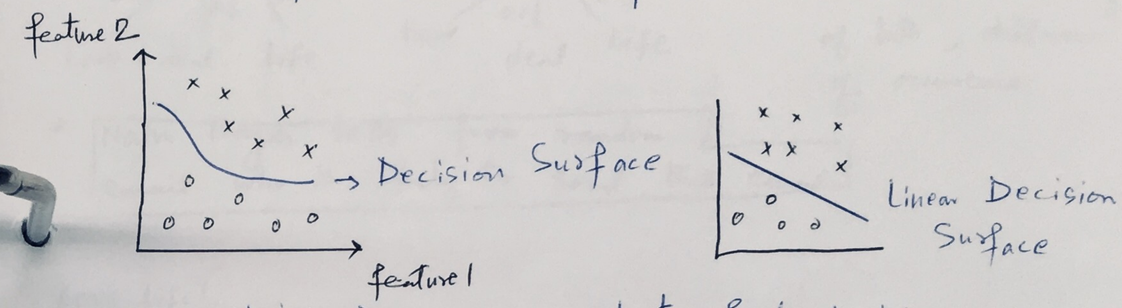
Machine Learning

→ Acerous → animals w/ no horns.

⇒ Non-acerous → animals w/ horns.

feature & attribute

* One of imp. things in ML is where to put (arrange) new data (or point) based on previous data.



* Naive Bayes is used to find decision surfaces.

* scikit-Learn.
  1- Create classifier.
  2- train it by fitting the features & Label data
  3- predict label using test feature.

* Accuracy $= \dfrac{\#\text{ of points correctly classified}}{\text{all points in test set}}$

⇒ Bayes Rule:

Question:

$P(c) = 1\%$

Sensitivity $P(POS|c) = 90\%$

Specificity $P(NEG|>c) = 90\%$

$P(c|POS) = ?$

$P(POS|>c) = 1 - P(NEG|>c) = 0.1$

$P(>c) = 1 - P(c) = 0.99$

Sol. $P(c|POS) = \dfrac{P(POS|c)P(c)}{P(POS)}$

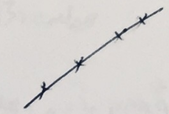
$P(c|POS) = \dfrac{P(POS|c)P(c)}{P(POS|c)P(c) + P(POS|>c)P(>c)}$

$P(c|POS) = \dfrac{0.9(0.01)}{0.9(0.01) + 0.1(0.99)}$

$\boxed{P(c|POS) = 8\tfrac{1}{3}\%}$ OR 8.33%

8% chance of cancer given positive test

$r_{xy} \neq 1$

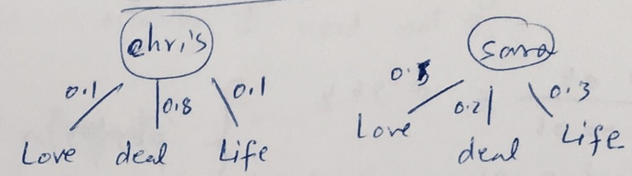


7 May (Night Shift)

(2)

→ Text learning:

Naive Bayes Rule:



+ love deal life are text in messages of both, difference of occurence

+ Naive Bayes tells from random email who is likely to send this email

msg: Love Life!

⇒ apriori:

$P(chris) = 0.5$

$P(Sara) = 0.5$

+ Sara is most likely to have sent this message

⇒ ~~apriori~~ New msg:

LIFE Deal

→ More likely from Chris. As:

$$\frac{0.1 \times 0.8 \times 0.5}{0.1 \times 0.8 \times 0.5 + 0.3 \times 0.2 \times 0.5}$$

Chris $= (0.1)(0.8)(0.5) = 0.04$ ← OR $\frac{0.04}{0.04+0.03}$ 0.07

Life deal apriori

Sara $= (0.3)(0.2)(0.5) = 0.03$ ← OR $\frac{0.03}{0.03+0.04}$

⇒ posterior:

$$P(chris \mid \text{"Life Deal"}) = \frac{P(\text{"Life Deal"} \mid chris)\,P(chris)}{P(\text{"Life Deal"} \mid chris)\,P(chris) + P(\text{"Life Deal"} \mid \neg chris)\,P(\neg chris)}$$

$$= \frac{(0.1)(0.8)(0.5)}{(0.1)(0.8)(0.5) + (0.3)(0.2)(0.5)} = 0.57$$

$$P(Sara \mid \text{"Life Deal"}) = \frac{P(\text{"L D"} \mid Sara)\,P(Sara)}{P(\text{"L D"} \mid Sara)\,P(Sara) + P(\text{"L D"} \mid \neg Sara)\,P(\neg Sara)}$$

$$= \frac{(0.3)(0.2)(0.5)}{(0.3)(0.2)(0.5) + (0.1)(0.8)(0.5)} = 0.428$$

→ New Msg: "Love Deal"

Method-1:

$P(\text{chris}, \text{"Love Deal"}) = (0.1)(0.8)(0.5) = 0.04$

$P(\text{Sara}, \text{"Love Deal"}) = (0.5)(0.2)(0.5) = 0.05$

} joint

And:

$P(\text{chris} \mid \text{"Love Deal"}) = \frac{0.04}{0.04+0.05} = 0.444$

$P(\text{Sara} \mid \text{"Love Deal"}) = \frac{0.05}{0.04+0.05} = 0.555$

} posterior

Method-2:

$$P(\text{chris} \mid \text{"Love Deal"}) = \frac{P(\text{"love Deal"} \mid \text{chris})\, P(\text{chris})}{P(\text{"Love Deal"} \mid \text{chris})\, P(\text{chris}) + P(\text{"love Deal"} \mid \text{>chris})\, P(\text{>chris})}$$

or P(Sara)

$$= \frac{(0.1)(0.8)(0.5)}{(0.1)(0.8)(0.5) + (0.5)(0.2)(0.5)} = 0.444$$

* Actually ~~Bay~~ Method 1 & Method 2 both are same!

⇒ Why Naive Bayes is <u>Naive</u>?

They are hidden, always in Supervised Learning

* Every word gives us evidence either its person A or person B
* We multiply probabilities of all the words appearing in msg.

A .....
B .....

Ratio

Prior probabilities

* Called Naive, b/c it ignores the <u>order of the words in message</u>

⇒ Where ~~Bay~~ Naive Bayes Breaks:

In case of word order, so it won't differentiate b/w "Bull Chicago" & "Chicago Bull" although both have totally diff. meanings & "Chicago Bull" is actually 1 word not 2.

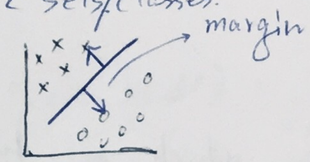
⇒ Project:

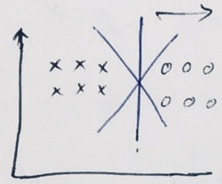
→ Based on text of email, classify which of two persons.

* Naive Bayes is very good for text classification as it treats each word as a feature & where features are independent.

→ SVM [Support vector Machine] → Invented by Russian

* It is a supervised learning method like Naive Bayes Rule.
* It takes data & creates/forms boundary b/w 2 sets/classes.

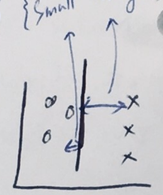


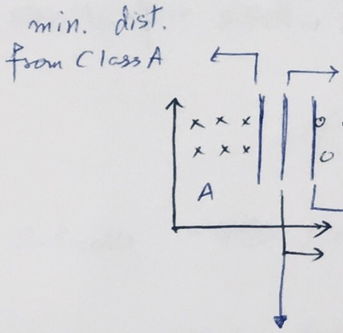


→ Best b/c of Margins

* What makes a good seperator?

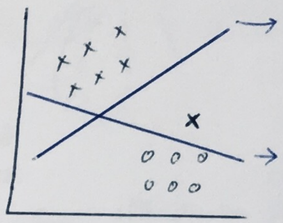
Margin → max. distance to the nearest point from either of the two classes.





Best as it maximizes the margin to nearest point from either (both) of ~~the~~ classes

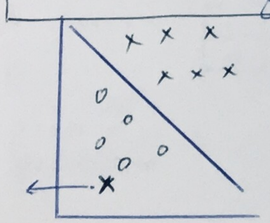
* Max. margin on either side reduces classification error.



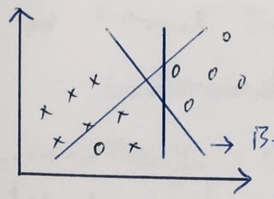
→ provides max. margin but doesn't classify data. — Contains classification error.

→ Best (It seperates data into 2 classes & also provides margin)

* So first correct classification is performed & only then the margins are maximized.



* SVM is good enough to classify, maximize margins as well as neglect outliers.



→ Best (* classifies correctly, * maximizes margins, * ignores outliers)

Adv. :

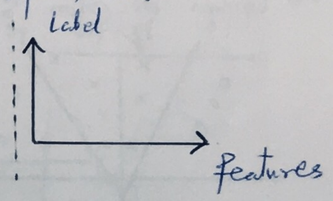
* SVM is efficient in high dimensional spaces.
* Effective when dimensions are more than number of samples.
* Diff. Kernel functions can be used for decision function.

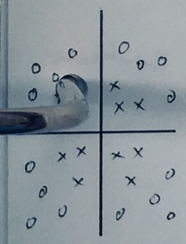
Disadv. :

* If # of features is much greater than # of samples, it performs poorly — Hughes Phenomenon – Curse of Dimensions
* doesn't provide probability estimates. 5-fold cross validation is used for probability estimates which is expensive.

---

* Margins make sure that there is as much seperation among classes as possible.



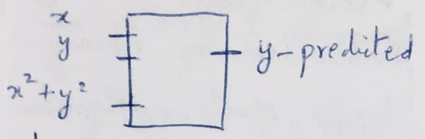


* Linear SVM will not work on this kind of data.
* There is no Linear Hyperplane (seperator) b/w two classes.

* input for training → features & labels
* input for predict → test features & output → predicted labels
* accuracy score → compare test labels with predicted labels

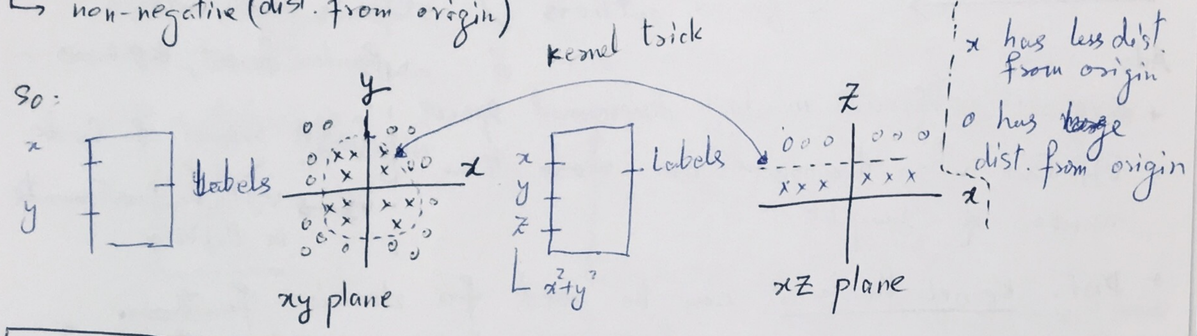
* scikit-learn
* Machine learning library.
classific, clustering, Regression
Naive Bayes, SVM, Random forest, KMeans ...
* Google Summer of Code
VI → Virtual Instruments in LabView

x → train-features.
y → train-Labels.
y-predicted → predicted Labels



↳ non-negative (dist. from origin)

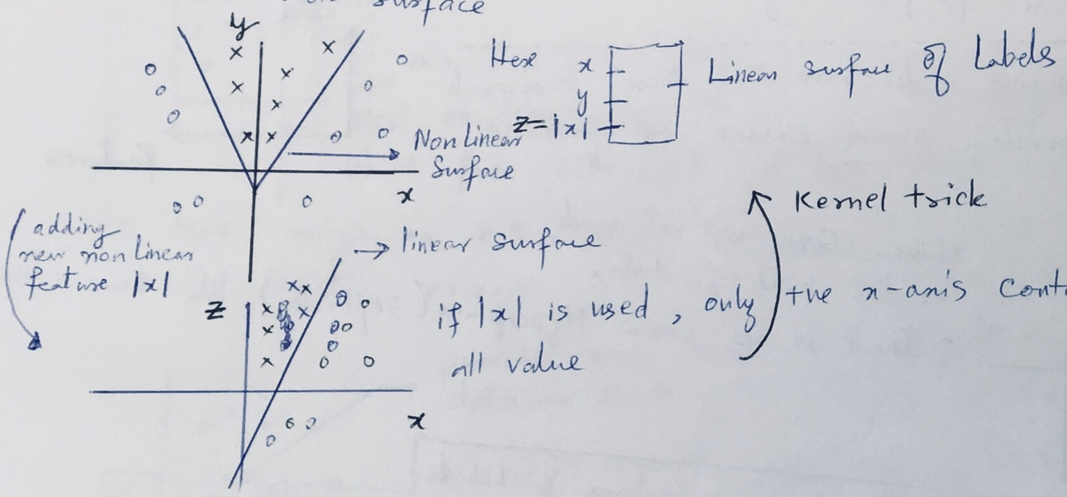
So:



This data is non Linearly seperable

* Line in xz plane corresponds to circle in xy plane

xy → here its non Linear decision Surface
xz → Linear decision Surface



if $|x|$ is used, only +ve x-axis contains all value

Kernel trick

⑤

⇒ Kernel Trick:

Low dimensional
x, y
nonseperable
Non Linear Seperation

Kernels

high dimensional
$x1$ $x2$ $x3$ $x4$ $x5$
Seperable
Linear Seperable Solution

* we use kernel to go from low dimension to high dimensional & find Linear Seperation, using info of linear seperation we can find nonlinear seperation in original data

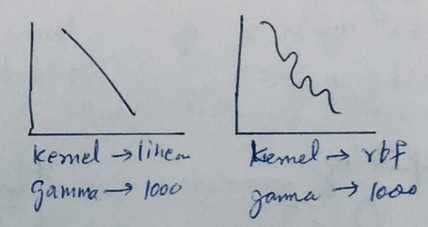
* There are several SVMs in scikit-learn, we are using SVC (supp. vec. classifier)

* SVC supports these kernels:
  1- Linear
  2- poly
  3- rbf
  4- sigmoid
  5- precomputed

* default is rbf

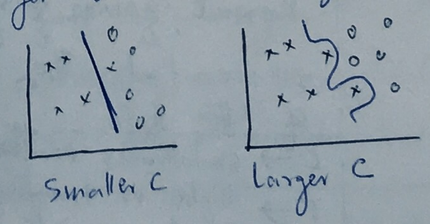
* parameters → passed when creating classifier before fit

params for SVM
→ Kernel
→ C
→ gamma



→ controls tradeoff b/w smooth decision boundary & classifying training points correctly.

* smaller C → smooth.
* larger C → correctly classifying but less smooth.

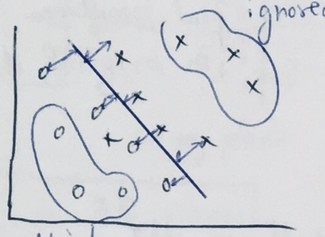


⇒ Gamma in SVM :

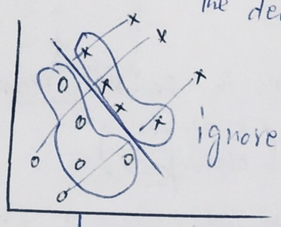
* defines how far the influence of a single training reaches

* Low — far reach
* high — close reach (decision boundary will be dependent on just the points closest to the decision boundary.

ignored

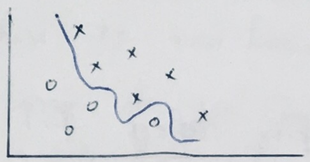


High γ



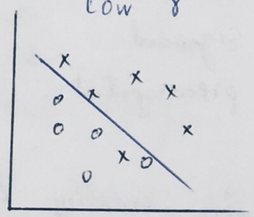
ignored

Low γ



High γ

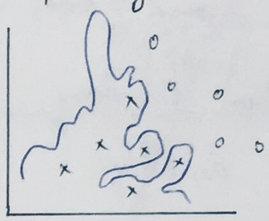
Higher influence of nearby points



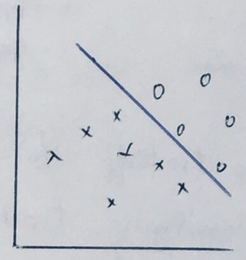
Low γ

lower influence of nearby points

⇒ Overfitting



vs.



better fitting

Overfitting

* when classifier takes the data too literally.
* classifier is trained on same kind of training data again & again w/o variation. So its now overfit for that data & not generalized

Kernel, gamma & C

all controls the overfitting of the data.

C → smoothness vs. best boundary

γ → higher vs. lower influence of nearby points

Kernel → linear, poly .....

Adv. of SVM:

Disadv. of SVM:

* Noisy data, overlapping data → doesn't work good.
* With very large dataset & lots of feature, SVM can be slow. Naive Bayes can be a much better classifier for these situation as feature as independent & order doesn't matter in Naive Bayes.
* Need to count "independent evidence" when classes are too overlapping i.e noisy data

---

* The more training data we have, the more is accuracy but slower is the data fitting — most of the time is consumed in training the classifier — Prediction takes very smaller time.
* In realtime usage like voice recognition & transactions blockage etc, we prefer speed over accuracy.
* naive bayes is usually way faster than SVM for large data sets.

* Using combination of high C & rbf kernel can give very good accuracy on smaller data set but on the expense of very complicated decision surface & the classifier will be much less generalized which is very bad.

↳ But if increase the data set size, classifier will be slow in training. — so right balance needs to be achieved.

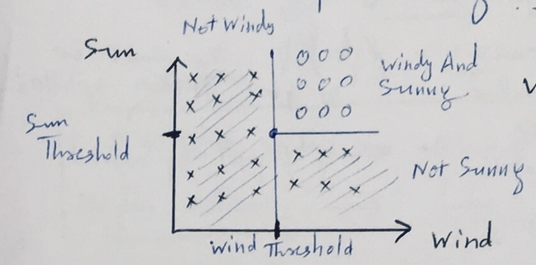
svm.predict(test_~~labels~~ feature)

* It provides us predicted class for each data point of test_~~labels~~ features.

* gridCV → sklearn tool that finds optimal parameters.

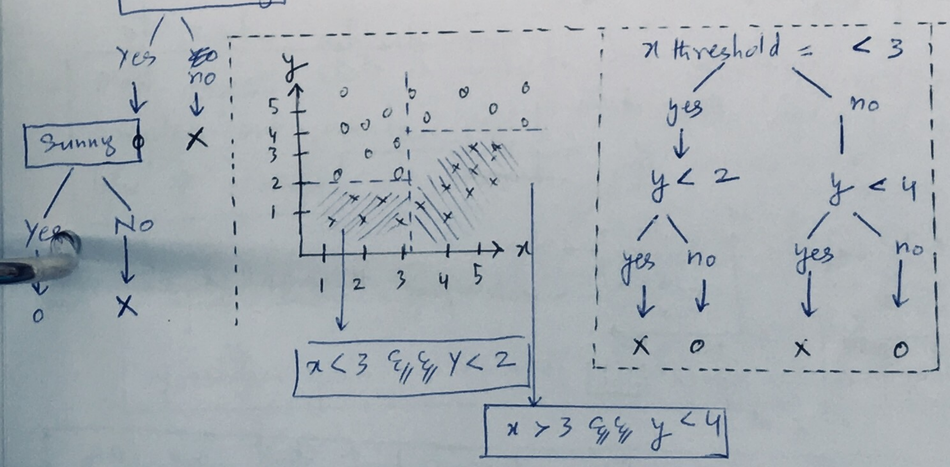
→ Decision Trees:

* In SVM, we were able to change from Linear to Non Linear Decision Surface using Kernel Tricks with the help of Margins



windsurfing → you enjoy surfing when its Sunny & windy

* ooo → suitable condition for wind surfing
* Its not Linearly seperable
* Decision Tree allows to ask Multiple linear Question one after another.
* Question 1: Is it windy (with the threshold)

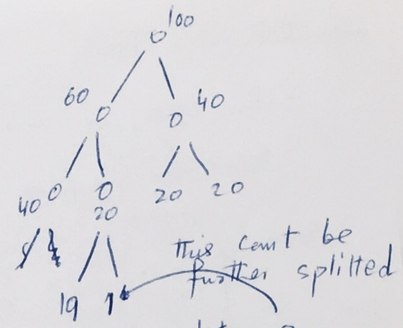


* So we classify data using data structure called Tree.
* job is to seperate o & x

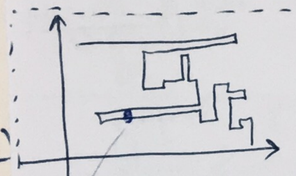
* Cost of running tree is Logarithmic in the number of data point in training.
* Uses white box model → any observable situation can be explained by boolean logic unlike artificial neural networks.
* Can have overfit by creating over-complex decision trees.

⇒ Decision Tree Parameters.

* min-sample-split → * min. value of subsample node allowed to be further splitted.
  * So 4 means node with value 4 wont be splitted further.
* default is 2.



* min-sample-split = 2

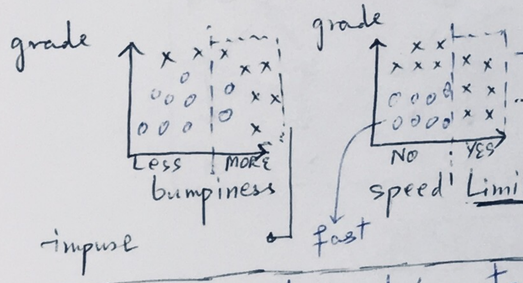


B/c of several nested/deep decision trees with overfit

if min-sample-split = 2 Then it'll create this whole decision surface just to grab this point

⇒ Data Impurity & Entropy:

* entropy → Controls how a DT decides where to split the data.
  → measure of impurity



* So decision tree tries to make subset that are as pure as possible & thats how DT actually makes its decision

$$\text{entropy} = \sum_i -p_i \log_2(p_i)$$

pure ← $0 \le \text{entropy} \le 1$ → impure

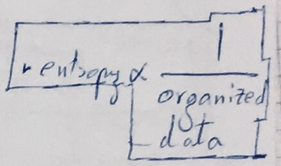
$\sum_i$ → sum over all classes available

p_i → fraction of examples in a given class.

$\log_2$ → 2 class; Can be 10, e ....

Cases for Entropy:

* all examples are same class:
  entropy = 0 → pure
* examples are evenly split b/w classes:
  entropy = 1.0 → impure



Lower entropy	→	More Organized data
Higher entropy	→	Less Organized data

grade	bumpiness	speed Limit?	Speed
steep	bumpy	yes	slow
steep	smooth	yes	slow
flat	bumpy	no	fast
steep	Smooth	no	fast

* $p_{slow} = 0.5$ → fraction of slow
* $p_{fast} = 0.5$ → fraction of fast

So

$$\text{entropy} = \cancel{0.5 + 0.5}\ 1.0 \Rightarrow -0.5\log_2(0.5) + -0.5\log_2(0.5)$$

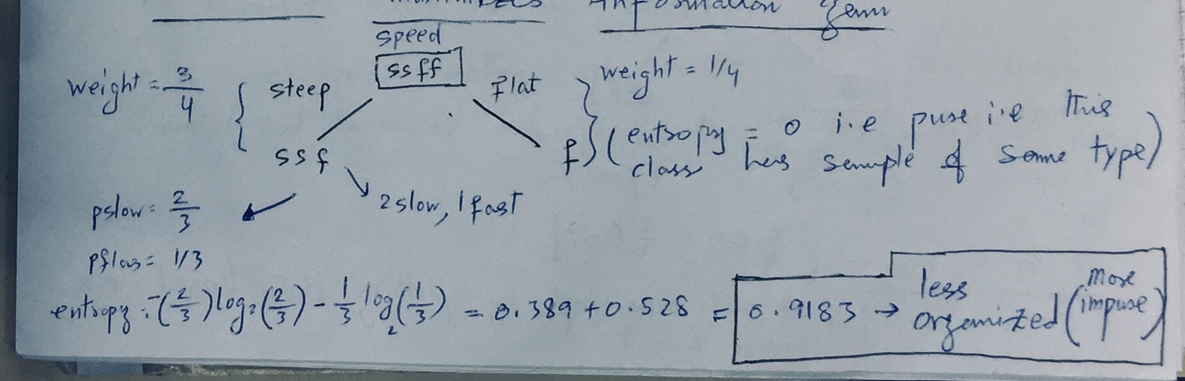
impure state

* Most impure state is when samples are evenly split b/w 2 classes.
* entropy of parent First(Start) node (max impure) 1.0

⇒ Information Gain: (IG)

information gain = entropy(parent) − weighted average of entropy(children)

* decision tree maximizes information gain



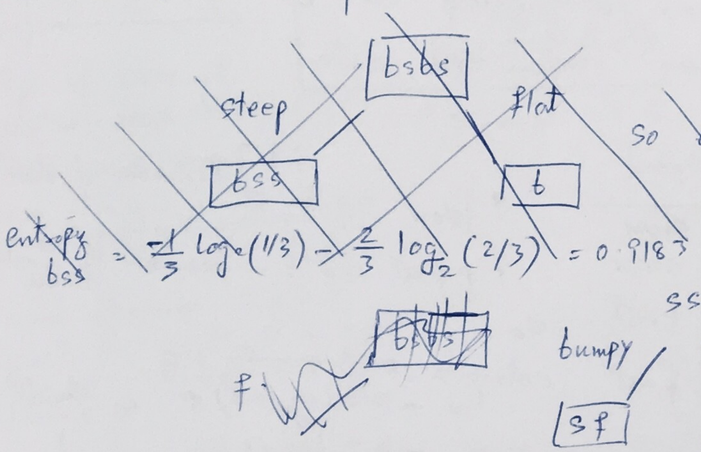
So

$$IG = \begin{pmatrix}\text{parent}\\ \text{entropy}\end{pmatrix} - \begin{bmatrix}\text{weight}\\ \text{Average}\end{bmatrix}\begin{pmatrix}\text{children}\\ \text{entropy}\end{pmatrix}$$

$$IG = 1.0 - \left[\frac{3}{4}(0.9183) + \frac{1}{4}(\cancel{0.9183}\ 0)\right]$$

$$\boxed{IG = \cancel{0.0817}\ 0.311275}$$

⇒ Consider bumpiness



And

$$I.G = 1.0 - \left[\frac{2}{4}(1.0) + \frac{2}{4}(1.0)\right]$$

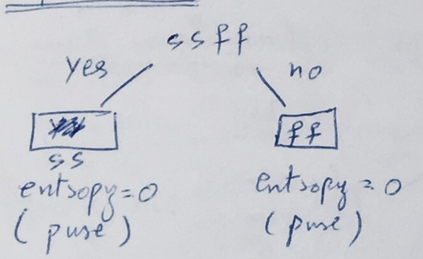
entropy parent

↳ weighted Sum of entropy children

$$\boxed{I.G = 0}$$

we don't gain any information by splitting data in bumpiness for this example.

→ Consider Speed Limit:



So:

$$I.G = 1.0 - \left[\frac{2}{4}(0) + \frac{2}{4}(0)\right]$$

$\boxed{I.G = 1.0}$ → We get max. info by splitting data based on Speed Limit

* in scikit learn, in DT, criterion param → gini / entropy

(no Info) Worst $0 \le$ Inform. Gain ≤ 1 Best (Full Info)

(pure) Best $0 \le$ entropy ≤ 1 Worst (Impure)

$$\boxed{\text{Information Gain} \propto \frac{1}{\text{entropy}}}$$

* for impure classification you get no information
* for pure classification, you get max. information

pure classification

class A	class B
Apple Apple	Banana Banana

* max Info
* low entropy

⇒ Impure Classif.

class A	class B
Apple Banana	Apple Banana

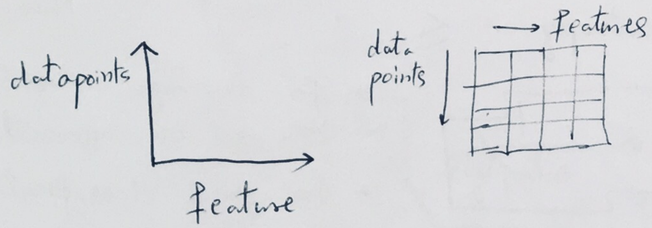
* no Info
* max entropy

⇒ Bias-Variance Dilemma:

* Bias classifier is the one on which there is no effect of data. So whichever may you train it, it'll never do things differently.
* Variance classifier is one which is very sensitive of the data, it can only replicate the stuff it has seen before. It'll react poorly for new data as it has very little bias to generalize itself to new stuff.
* So we need something in middle so it can learn from (good variance) but could also be generalized (good bias).

⇒ Tradeoffs with Decision Tree.

* Can be easily interpretted, whitebox – explained using boolean logic decessable situations can be
* Prone to overfitting b/c of overly-complex decision tree with lots of features.
* Decision Tree classifer can be used to new classifier (~~Unsampled~~ ensemble Classifier)



ensemble classifier
Several small & weak classifiers are combined together to form a single robust classifier.

# of feature ∝ complex fit

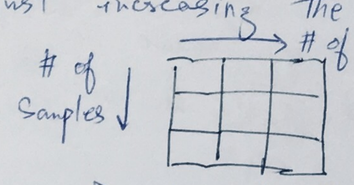
assuming sample size is constant

⇒ Adaboost.

* Adaptive boost + meta algorithm.
* ensemble → combines (weighted sum) several weak learners (decision trees) to form a robust classifier.
* adaptive → subsequent weak learners are favored over those instances misclassified by previous classifiers.
* Hughes phenomenon → with fixed number of training samples, prediction power ∝ $\frac{1}{\text{dimensions}}$

Curse of Dimension

→ Means if you have lots of features but very small training sample, then prediction is weak as features are just increasing the dimension.



→ So, rows & cols should both grow to have good prediction.

* pruning → process of removing poorly performing weak classifiers

* Bias ∝ $\frac{1}{\text{Data Dependence}}$; Variance ∝ Data Dependence

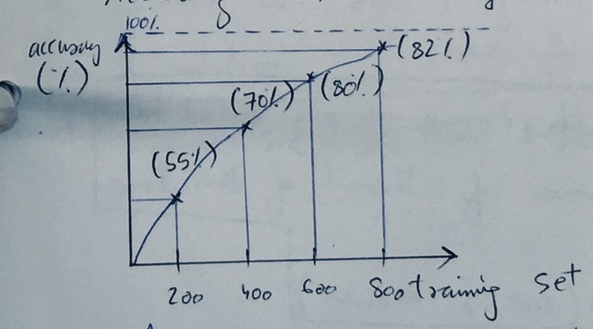
⇒ Enron fraud:

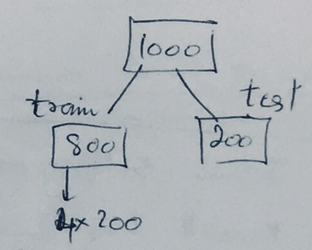
* Enron was energy trading company with 101 bn $ revenue.
* went bankrupt b/c of clever economic & financial practice.
* 10,000 of emails (real) were released as corpus & we will use these emails to train our classifiers & get insight.

⇒ Datasets & Questions:

* Person of Interest (POI): → indicted
  → settle w/o admitting guilty.
  → testified for immunity.

* Accuracy ⟺ Training set size





* Accuracy ∝ Training set size
* But we shouldn't have lots of features b/c it'll create Hughes phenomenon that too much features w/o increasing training set so just increasing dimensions & reducing prediction accuracy.

** | More Data > fine tuned Algorithm |

Types of Data:

Numerical – numerical values (feature → number)

Categorical – Limited # of discrete value (feature → category)

time series – temporal value (date, timestamp)
↳ features

text – words → feature
↳ we convert words feature to number using Bag of words formula etc.

* salary → numerical feature
* job title → categorical feature
* timestamp on email → time series feature
* email contents → text feature
* # of emails sent → numerical feature
* to/from field → can be categorical feature but fits better as text feature where text is email addresses. text wrangling tools can be used.

* Jeffrey Skilling → Ceo of Enron during fraud time
* Kenneth Lay → Chairman of Board of Directors.
* Andrew Fastow → CFO
* Unfilled feature → NaN (Not a Number)

→ features

# of people ↓

POI	Salary	Expenses	...

salary	to_messages	deferral_payments	total_payments	exercised_stock_options
bonus	restricted_stock	shared_receipt with POI	restricted_stock deferred	total_stock_value
expenses	loan_advances	from_messages	other	from_this_person_to_poi
poi	director_fees	deferred_income	long_term incentive	email
from_poi_to_this_person				

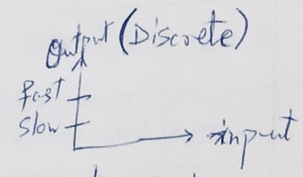
↑ features

* most non-POIs come from financial statements with complete data.
* POIs were added from Enron data with very incomplete data
* classifier learner can be tricked into thinking that POIs are those which have several NaN fields – which is obviously not the case.
* So we should be very careful when introducing new feature while using data from several sources
* 1 solution is to use features common to all sources.

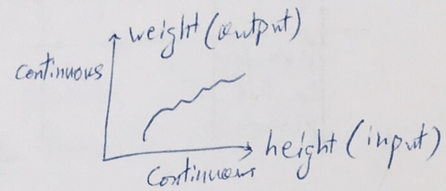
↳ feature selection from Multiple sources.

⇒ Regression.

* Continuous Supervised learning:
  ↳ output.
  * input is normally continuous but o/p can be discrete (binary) or continuous.



↳ Continuous

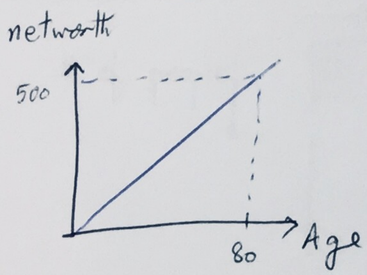


* Age → Continuous
* weather → discrete (Sunny or rainy)
* But most things we regard as discrete could be modelled as continuous. e.g. weather could mean the amount of light and this is continuous.
* person wrote email → discrete
  * even though we can have thousands of author but they are not continuous in time So #56 & #59 has no relation in time.
* phone number → discrete
* income → continuous

Without order	→ discrete
With order	→ Continuous

Regression → predicting relation among data & finding best fit line.

* Discrete	* Continuous
fast/slow	speed in mph



net worth = m (Age) + b

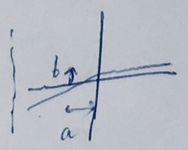
$m = \frac{500-0}{80-0} = 0.25$.

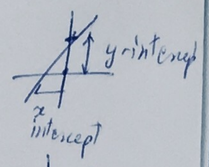
$b = 0$

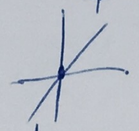
So:

networth = 0.25 (Age)

y-intercept → height when line intersects y axis







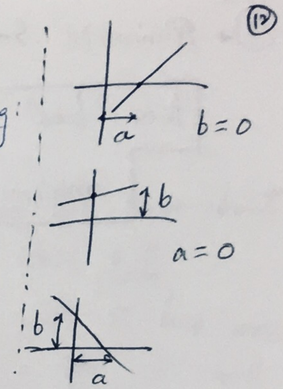
b = 0 (y intersc.)
a = 0 (x intersc.)

Target (out) = Slope (Input) + y-intercept

* Regression → predict output from input using:

$$\boxed{y = mx + b}$$

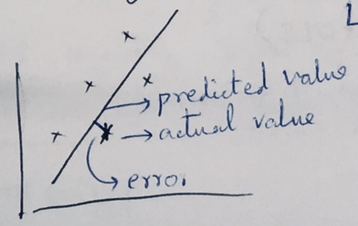
↳ Linear Regression



* Create
* fit
* predict

* We need to find $\boxed{m}$ & $\boxed{b}$

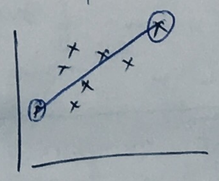
⇒ Linear Regression Errors:
↳ actual − predicted



error → −ve means actual point is below predicted point.

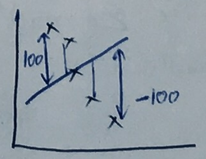
error → +ve means actual point is above predicted point.

* error on first & last data point:



Not a good fit

* Σ error on all data points



* Not a good fit
* both errors cancel each other out & result is straight line.

Σ error = 0
even its not fitting at all.

* Minimize Sum of Square Error — $\Sigma(\text{error})^2$;

$\boxed{\text{the best regression is: minimizes } \sum_{\text{all training points}} (\text{actual} - \text{prediction})^2}$

prediction from regression

training points

$$y = mx + b$$

*Methods to minimize $\Sigma(\text{actual} - \text{prediction})^2$ OR SSE

① Ordinary Least Square (OLS):

→ used in sklearn Regression.

② Gradient Descent:

$$y_i = y_i + \alpha(x_i - y_i)$$

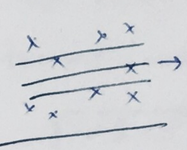
$$y_i = y_i + \beta(y_{i-1} + y_{i+1} - 2(y_i))$$

$y_i \to$ new

$x_i \to$ old

$\alpha \to$ data smooth

$\beta \to$ weight smooth



→ best regression result

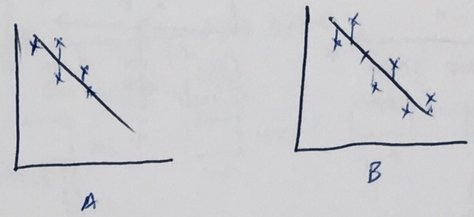
↳ in terms of SVM, it has Large margin on both side.

There can be multiple lines that minimizes Σ Error but only one line will minimize $\Sigma(\text{Error})^2$

* Another reason for using SSE i.e $\Sigma(\text{error})^2$ is that its much easier to code mathematically.

⇒ SSE is not perfect:

$\boxed{\text{larger SSE} \to \text{Worst fit}}$



* These both have about the same Fit
* But just B has more data points, it will have greater SSE.

In SSE, as you add more data, the SSE usually goes up even though the fit is not necessarily bad.

* SSE gives relation of test labels & predicted labels.

r^2 gives relation b/w input & output of test data

⇒ r^2 of a regression:

* how much Δoutput is explained by Δinput

$$0 < r^2 < 1$$

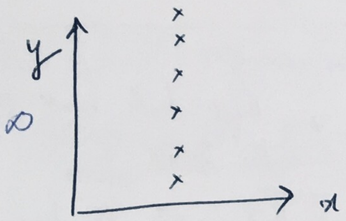
not doing good job of capturing trend in data

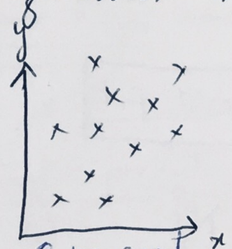
good description / relation b/w input & output

* r^2 can be in −ve when finding fit or regression for significantly different data set.
* r^2 is independent of # of training points.

⇒ A good Linear Regression Data

$m = \infty$



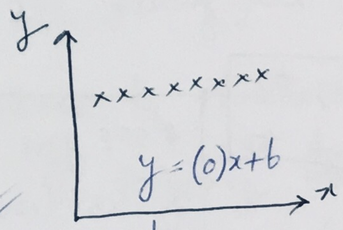


no good regression

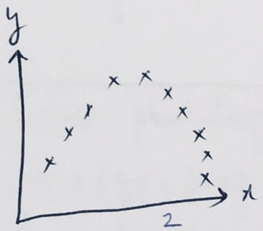
Can't be done using Single line

$y = mx + b$

$m = 0$



✓ good



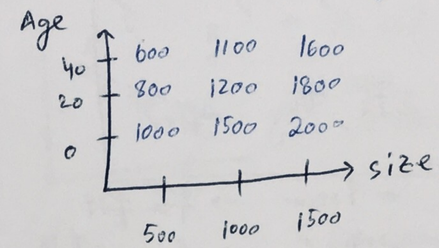
$y = ax^2$
parabola

⇒ Comparing Classification & Regression:

property	Supervised Classification	Regression
output type	Discrete	Continuous (Numbers)
what are we find	Decision Boundary	Best line of Line
evaluation	accuracy score	* Sum of Squared Error * R Squrase * gradient descent

⇒ Multi Variate Regression:

height, weight, educ; } → you



$y = \square\, x_1 + \square\, x_2 + \square$

$m_1 \quad m_2 \quad m_3$

Sol.

$$m_1 = \frac{1500 - 1000}{1000 - 500} = \boxed{1}$$

$$m_2 = \frac{1000 - 800}{0 - 20} = \boxed{-10}$$

And:

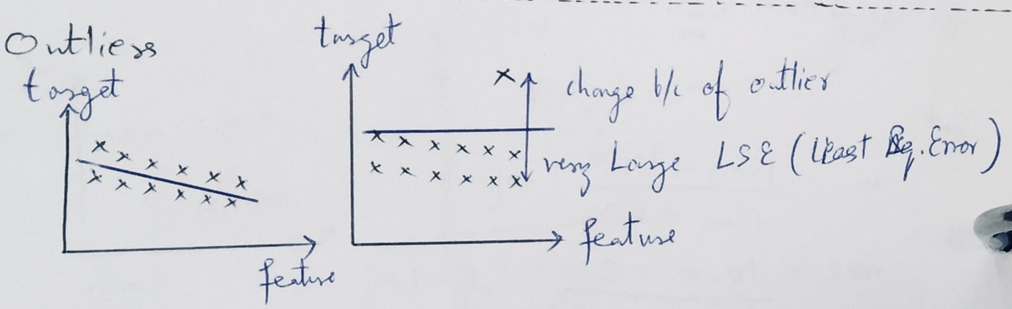
$$1000 = 1(500) + (-10)(0) + m_3$$

$$\boxed{m_3 = 500}$$

⇒ * If outliers lie in training data, they change slope & intercepts.
* If outliers lie in testing data, they affect negatively the regression score.

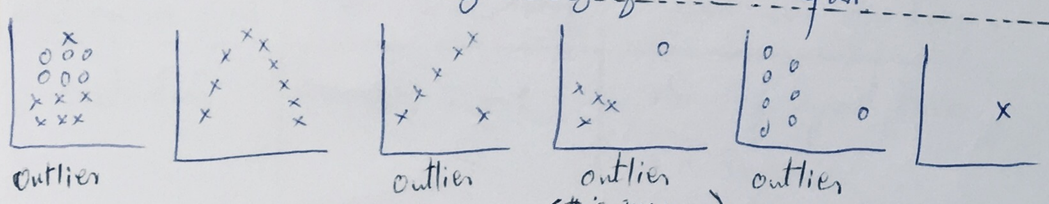
* train → for fit
* test → for predict

⇒ Outliers



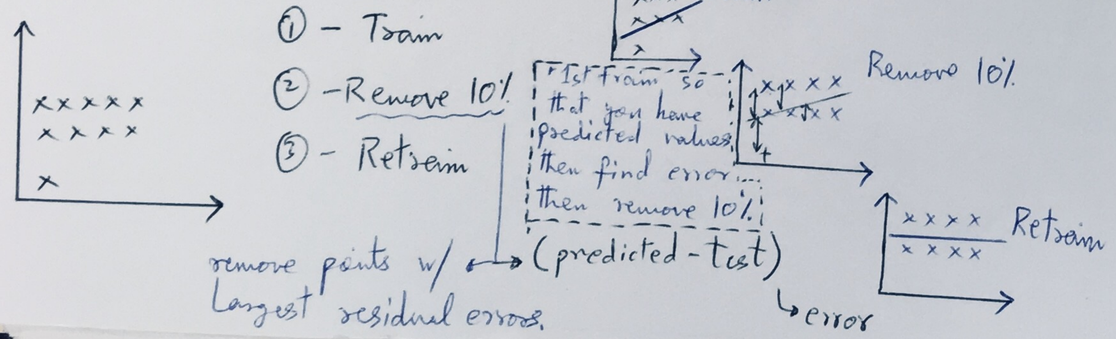
* Outliers can be caused by
  - → Sensor Malfunction
  - → Data Entry Error
  - → Special Event

* In robotics & SDC, we usually ignore outliers but in other fields like fraud detection, outliers are very highly looked for.



⇒ Outlier Removal:

① - Train
② - Remove 10%
③ - Retrain

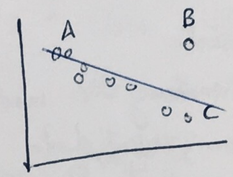


* 1st train so that you have predicted values then find error... then remove 10%.

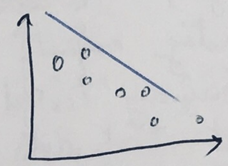
remove points w/ largest residual errors. ↔ (predicted - test)

Cost minimized SSE * high Variance

⇒ Residual Errors



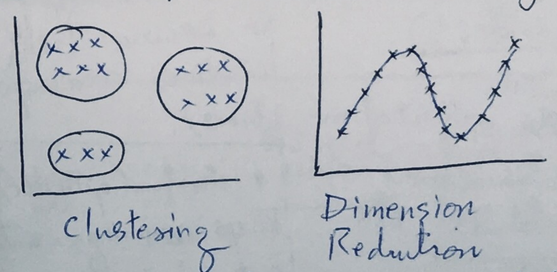
* D will have highest Residual Error after fit. i.e (predicted - actual)



* Steeper now, as we have no outlier
* In Regression, slope is also called Coefficient.

⇒ Unsupervised Learning:

* Structuring data without Labels.
* With labeled data, we dont generally need Unsupervised



* we didn't have classes but after looking at the data we can form the clusters & say them classes.

K_Means:

* Grouping a set of objects in clusters so that objects in same cluster are similar to each other than other clusters.

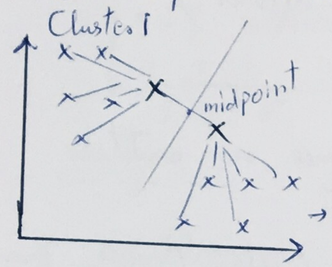
* $\boxed{\text{Kmeans uses Centroid model for Clustering}}$
* Kmeans (for Clustering) is different from K Nearest Neighbors (for classification)

$$\boxed{K \le n \quad \begin{array}{l} K \rightarrow \text{\# of clusters} \\ n \rightarrow \text{\# of data samples} \end{array}}$$

Algorithm for K Means:

① – Assignment.

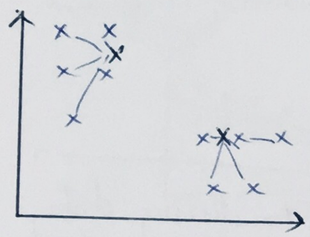
② – Optimize.



initial centroid

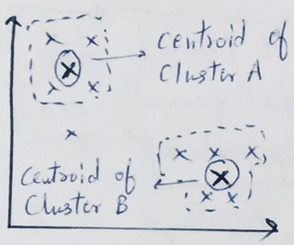
* Randomly place 2 points & form the line from their midpt.

* find difference b/w new & old centroids. Centroids change position of cluster point – rubber bands tries to be as short as possible.



* ~~further~~ iterative optimization.

* This is correct Centroid for both clusters.



centroids

* Initial random positions are very important & might end up with totally different clustering.

So: Kmeans input → n samples, how many clusters
output → K clusters

→ minimizes WCSS (Within Cluster Sum of Square)

→ in high dimension, Euclid distances becomes inflated due to Curse of Dimensions & Can be Solved by running reduction algorithms e.g PCA before clustering.

→ a.k.a Lloyd's algorithm.

Machine Learning:

* Algorithms which learn from data.

* By making a model based on input data (training features & ~~labels~~) & predicting output. with accuracy tested against test data.

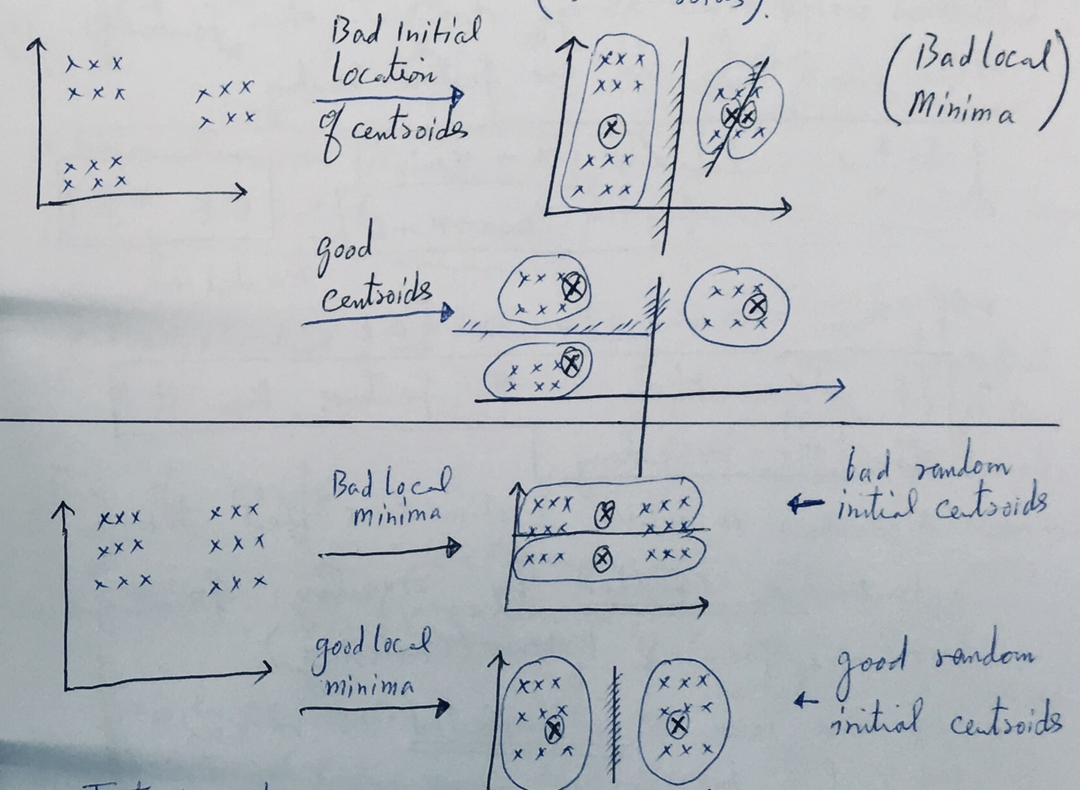
* Random forest & Adaptive Boost corrects the overfitting problem in Decision Tree (makes sometimes overcomplex trees).

* iteration (optimize) steps keeps running until centroid doesn't move very much.

[Local minimized SSE] * high variance

⇒ K Mean Challenges:

* Uses Hill Climbing technique for local minima. So the clusterings depends on where you put the initial cluster centre (or centroids).



* That is why we use several initial centroid positions & run the algorithm on each of them and then uses the best clustering out of them.

⇒ Feature Scaling :

$$\boxed{\text{Scaling} = \frac{x - x_{min}}{\text{Range}}}$$

feature-1 : [5, 7, 9, 6] → height

feature-2 : [170, 145, 150, 165] → weights

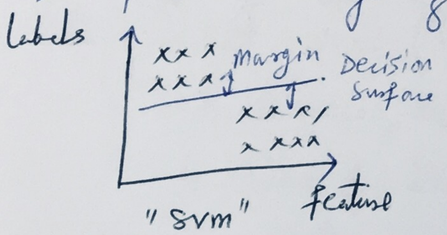
⇒ Now weight will always dominate other feature (height)

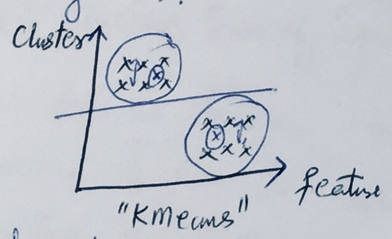
⇒ To avoid this, we use feature Scaling.

$$\boxed{\text{Feature-Scaling } x' = \frac{x - x_{min}}{x_{max} - x_{min}}}$$

$$\boxed{0 \le x' \le 1}$$ rescaled

* If $x_{min} = x_{max}$ then $x' = 0.5$
* If you've outliers in the features, then it'll effect the rescaling
* Algorithms in which 2 dimensions affect the output are effected by rescaling. for eg SVM (classifier) & KMeans (Clustering)
* Decision Tree is not affected as it uses vertical & horizontal Lines.
* Linear Regression is not affected as slope & feature always go together.



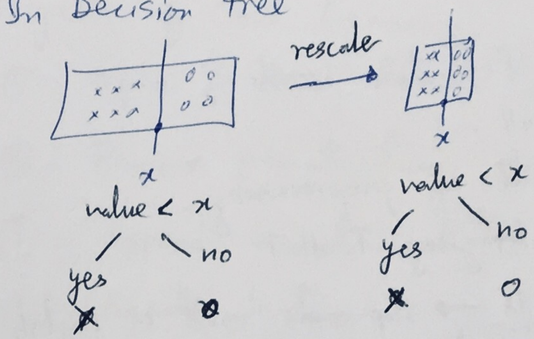


{SVM & KMeans both depend upon distance calc so are affected by rescaling}

* SVM relies on distance for Max. Margin
* KMeans relies on distance for Centroid Calculation → using Local Minima

Cost fn Minimized SSE → high Variance

* In Decision tree



* Rescale has no effect on Decision Tree
* Linear Regression is affected by outlier but not by rescaling

Linear Regression

- Input: * feature, * target
- Output: * slope, * intercept, * score, * predicted target

Classification

- Input: * features, * Labels
- Output: * features, * Labels, * accuracy, * Classes

Clustering

- * features
- * Clusters, * Cluster Centers

* .MinMaxScalar() → feature Scaling in Scikit Learn

⇒ Bag of words:

- CountVectorizer
- tfidf (term frequency, inverse document frequency)

vector.fit_transform(corpus)
vector.get_feature_name
vector.vocabulary
vector.transform(corpus).toarray()

* ~~Count~~ get_feature_name → List of features
  vocabulary → dictionary of features
  toarray → List of features count
* countvectorizer → 0 ..... 1 → frequency of feature
* tfidf uses Euclidean Dist. $\left(\frac{V}{\sqrt{V_1^2 + V_2^2 + \dots}}\right)$ → 2's norms.
  tfidf → (0 ..... 1)

Stop-words

- high frequency, less informative words.
- this, at, um, uh, is, will ...
- removing them is part of preprocessing.
- nltk → Natural Language Toolkit
- nltk.corpus.stopwords → stopwords used in english

Stemmesing:

- similar meaning words:

go, goes, going, gone $\xrightarrow{\text{stemmer}}$ go (stemmed word)

- nltk.stem.porter.PorterStemmer( ).
- stemmed word = ~~stem~~ PorterStemmer( ).stem(word)
- stemmesing is part of preprocessing.

> Stop words & Stemmer should be done <u>before</u> vectorization

Vectorization → forming a frequency vector (bag of words) of the text

'is, will, um, important' $\xrightarrow{\text{stopwords}}$ 'important'

receive, receving, received, receipt $\xrightarrow{\text{stemmer}}$ receiv, receipt

---

Code is minimized SSE → high variance

⇒ Project of Text Learning:

1. - open & Read Emails of Chris & Sara.
2. - Remove punctuation. Apply stemmer.
3. - Append email text to store in a list of emails text.
4. - Remove stopwords (* option1 → using nltk; * Option 2 → using tfidf param)
5. - form vector of email text i.e form Bag of Words using tfidf → TfidfVectorizer(min_df = 1, stop_word = "english")
6. - get frequency of features
7. - get <u>List</u> & <u>dictionary</u> of features names

> string $\xrightarrow{\text{split( )}}$ List
>
> List $\xrightarrow{\text{''.join( )}}$ string

> filter(Lambda condition: condition ≠ something, array)

→ removing the stopwords

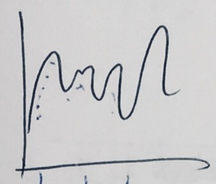
> tfidf → weights rare words higher than common words

→ term frequency Inverse document frequency

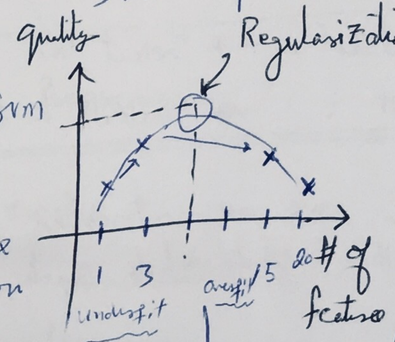
Carefully minimized SSE → * high Variance
* Complex Decision Boundary
* overfit

Large r^2 → regression Score
Low SSE → variation in test & predicted outputs

Sweet Spot



→ high gamma
→ high C } SVM
Overfit
→ high variance
→ high error on test data.
→ too many features



* high Variance ← too many features
* overfit
* high bias ← too few features
* low r^2 on training data
* Underfit

* Hughes Phenomenon
* Curse of Dimension
* too many features w/o increasing data set causes prediction power to be less OR low r^2 score (regression)

⇒ Regularization for Regression:
→ Penalizing extra features which are causing low regression score

LASSO ~~Expression~~ Regression: → Regular Linear Regression

Minimize SSE + $\lambda |B|$ → reduce # of feature used

reduce Error

$CI = a \sqrt{\frac{p-2}{n}}$ → quantiles

λ: penalty parameter

B: regression Coeff.

→ Adv. of high r^2 More features i.e

→ loss due to More feature i.e overfit

⇒ features ! = Information

↳ features attempt to access information.

Best → Minimum feature which can give max. information

⇒ Univariate feature Selection

1 → SelectPercentile() → Select x% of ~~select~~ features.

2 → SelectKBest () → Select k # of best features

3 → In text learning, we are interested in feature reduction as the data has such high dimension.

tfidf

(max-df)

→ 0 .... 1 [remove words which are occuring with this much frequency. So.

0.1 → occuring in 10% of document

0.8 → occuring in 80% of document.

max-df = 0.4 → words will be ignore if they are > 40%

&

min-df = 0.2 → words will be ignored if they have occurence < 20%

high Bias	high Variance
* pays little attention to training data	* pays too much attention to training data (doesn't generalize)
* oversimplified & underfit	* overfits
high error on training data	high error on test data
for regression low r, high SSE	

too few feature → Causes high Bias | Not paying attention to data.

tf idf → term freq. inv. document freq. → inverted freq. of words occurence

tf → bag of words, frequency of words.

idf → weighting by how often word occurs in corpus

tf idf → weights rare words higher than common words.

filter(lambda condition: condition != '', array)

Project of Preprocessing:

① - Read emails.

② - remove punctuation & apply stemming

③ - remove few given word.

④ - append this email text now to a list - storing a list of email text.

⑤ - fit tfidf on the list of preprocessed text, use stop_word parameter.

⑥ - get name_of_feature

Storing → list
split()

* remove stopwords.
* apply stemming.
* list → string
* apply contractions.
*

→ 38825

Feature Selection ①

- ~~Machine~~ learning algorithm prediction & training depends highly on features & data sample
  - → add new feature
  - → Select best feature.

Skilling — ceo
Kenneth — chairman
Fastow — cfo

- ~~Entitu~~ Intuition → POIs emails often POIs at higher rate than normal
- create a new feature
- visualize → does it discriminate POIs from non POIs.
- repeat → could we do better.

- Instead of ~~an~~ using absolute value, we can use fraction.

So: $\frac{\text{from POIs}}{\text{total from}}$ & $\frac{\text{to POIs}}{\text{total to}}$

- get emails
- → compare list to know POIs.
- → & return bool if author is POIs.
- → aggregate count over all emails to person.

⇒ Ignoring features:

- its noisy
- Causing overfit
- strongly correlated to a feature already present. (redundant features)
- ~~Slowselow~~ too many feature slowing down training process.

⇒ Bag of words: "Summary"

- 1. Use freq. of words in a text.

Remaining:
1. - Feature Selection
2. - Principal Component Analysis
3. - Validation
4. - Evaluation metrics

⇒ Bag of words

- Count Vectorizer
  - → vector.get_feature_name
  - → vector.vocabulary_
  - → vector.fit_transform(corpus).toarray()
- tfidf (term frequency inverse document frequency)
  - → list of features
  - → dictionary of features
  - → list of frequency
    - → CountVectorizer (0 ... n)
    - → tfidf (0 ... 1)
    - → Euclidean Normalization $\frac{v}{\sqrt{v_1^2 + v_2^2 + \dots}}$

⇒ Stop-words

- → high frequency, less important words.
- → nLkt → natural language toolkit
- → nlkt.corpus.stopwords → List of stopwords
- → removing stopwords from text is part of preprocessing.

⇒ Stemmerizing:

- → similar meaning words:
  - go, goes, going & do, does, doing, done.
- → nlkt.stem.porter.PorterStemmer
- → stemmed_word = PorterStemmer().stem(word)
- → stemming the words in the text is part of preprocessing

⇒ Stopwords & Stemmer should be done before vectorization

$x_1 \ldots x_4 \rightarrow$ feature

$m_1 \ldots m_4 \rightarrow$ Regression Coeff. for feature

$y = m_1 x_1 + m_2 x_2 + m_3 x_3 + m_4 x_4 + b$

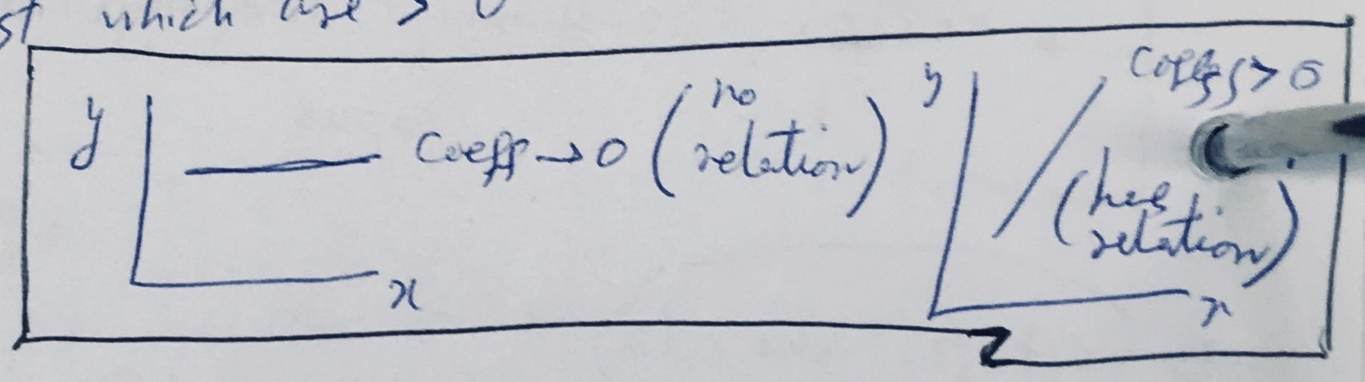
- $x_1 \ldots x_4$ → feature
- $m_1 \ldots m_4$ → regression coefficient (Slope)

+ LASSO uses Coordinate Descent

⇒ Apply LASO, exactly as Linear Regression

⇒ get coeff_

⇒ elements in coeff_ list which are > 0 are of interest.



Regressi

⇒ train. train target

+ $m = \frac{y}{x}$ → train feature

⇒ predict.

$y = \boxed{m}(x)$

test feature

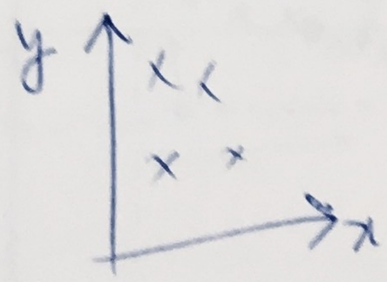
predicted target

⇒ Score

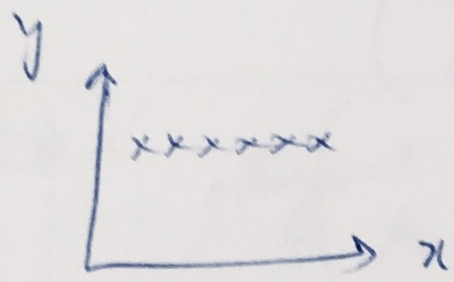
Compare

y_test & y_predicted

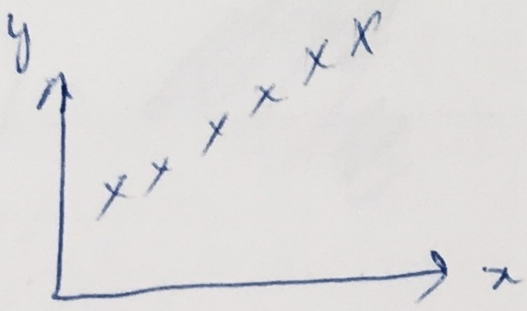
→ Principal Component Analysis — PCA

① 

2 dimensional

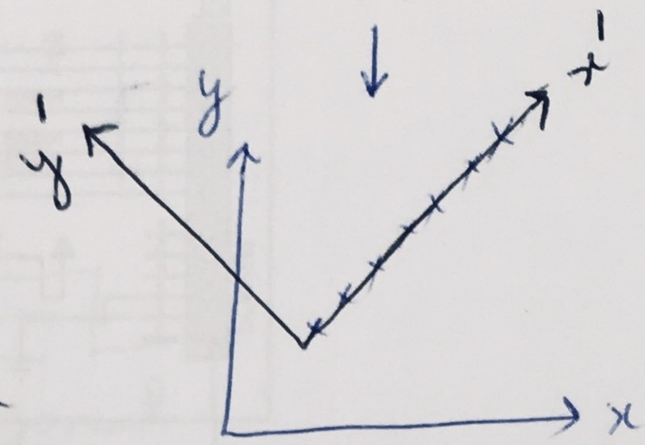
② 

1 dimensional (here x value are sufficient)

③ 

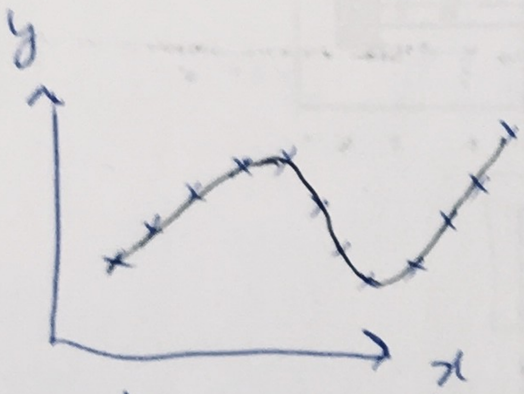
1 dimensional

as

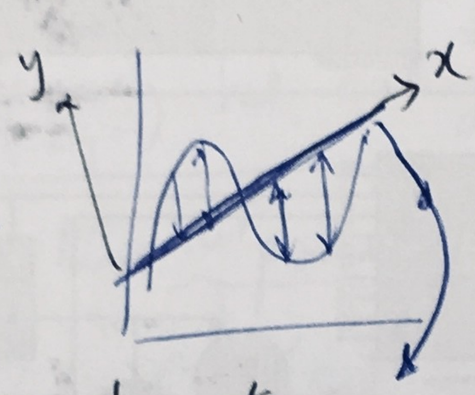


So, 1 dimensional

Or

④ 

2 dimensional

→ 

doesn't fit quite well. So 2D.

$y = mx$

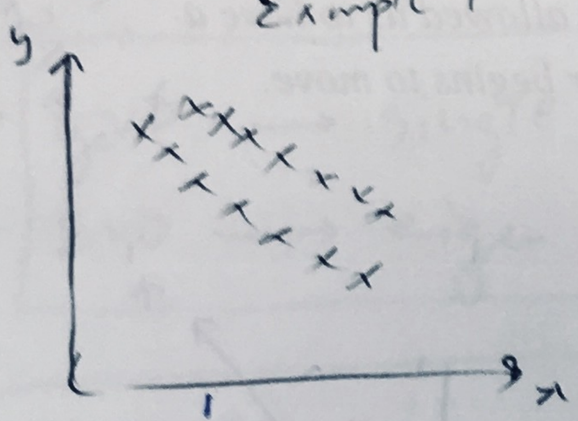
$y = mx$

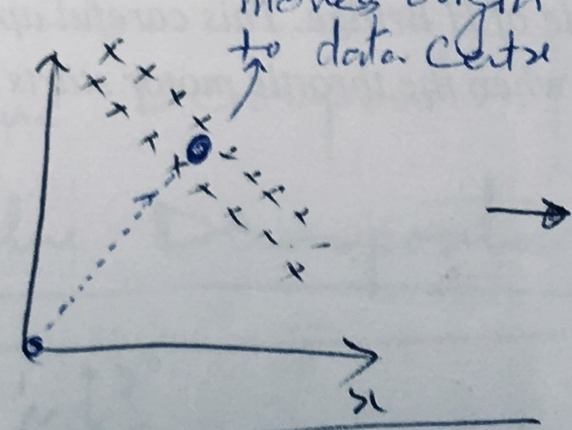
$1 = 1.5x$

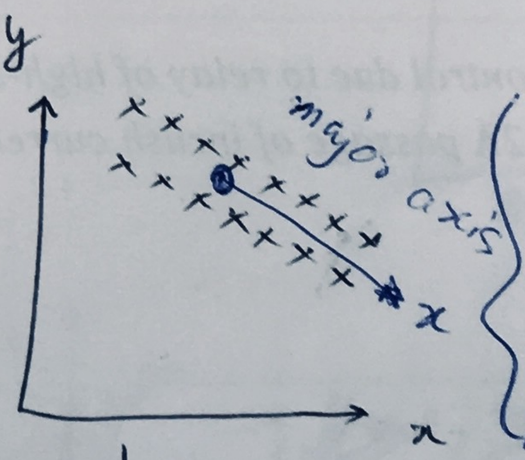
$x = 1/1.5$

→ PCA:

Example-1:

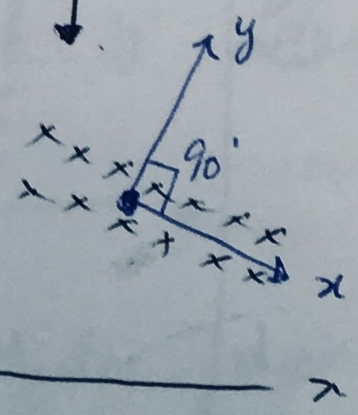


→ 

→ 

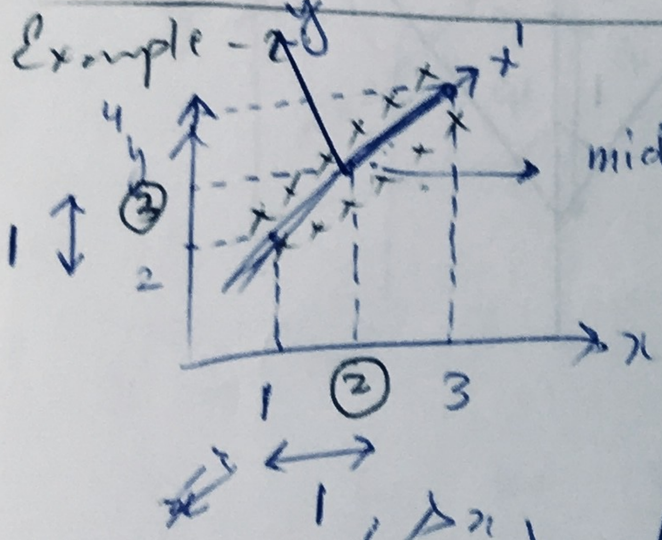
{ Makes as x where most variations are }

↓



{ ⊥ to x is y }

Example-2



midpoint $= \left(\frac{1+3}{2}, \frac{2+4}{2}\right)$

→ normalized.

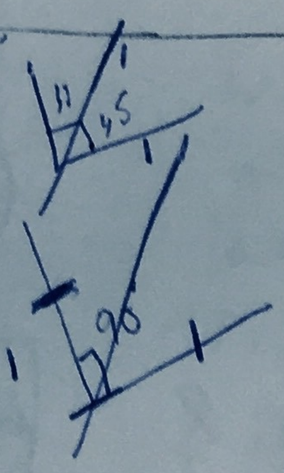
$$\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix} \quad \left\|\begin{matrix}\Delta x\\ \Delta y\end{matrix}\right\|_2 = \begin{matrix}1/\sqrt{2}\\ 1/\sqrt{2}\end{matrix}$$

→

And:

$$\begin{pmatrix}\Delta x'\\ \Delta y'\end{pmatrix} = \begin{pmatrix}-1\\1\end{pmatrix} \quad \left\|\begin{matrix}\Delta x'\\ \Delta y'\end{matrix}\right\|_2 = \begin{matrix}-1/\sqrt{2}\\ 1/\sqrt{2}\end{matrix}$$

$(\Delta x, \Delta y)$ orthogonal to $(\Delta x', \Delta y')$



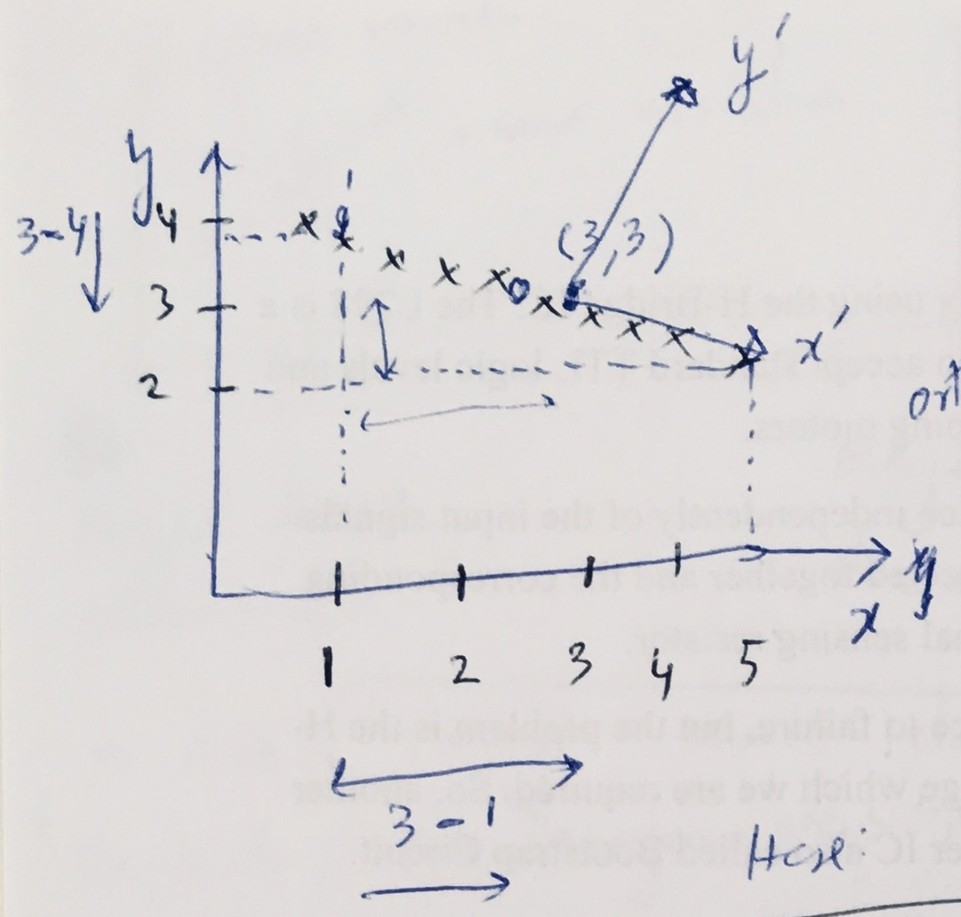
$m = \frac{\Delta y}{\Delta x}$ $\cos 45$

$m = \frac{4-3}{3-2} = 1$

$H^2 = B^2 + P^2$

$4 = B^2 + 1$

$B = \sqrt{4-1} =$



$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

orthogonal

And

$$\begin{pmatrix} \Delta x' \\ \Delta y' \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

Here:

$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$ & $\begin{pmatrix} \Delta x' \\ \Delta y' \end{pmatrix}$ are orthogonal

And these dot product is "0"

As: $\cos(90) \rightarrow 0.$

$A \cdot B = AB\cos\theta$

If orthogonal:

$A \cdot B = AB\cos 90 = 0$

$\vec{y} = 3$

$3 \cdot 3\cos\theta$

And

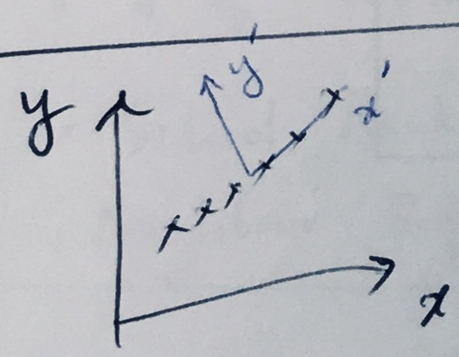
~~$A \cdot B = AB\cos\theta$~~

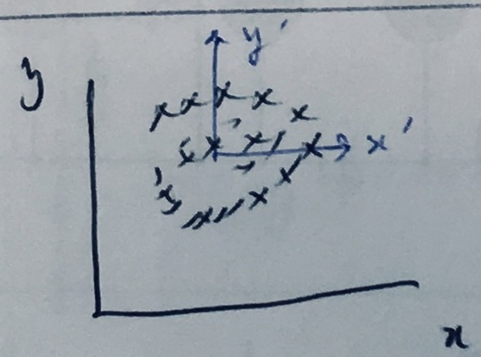
$\boxed{A = -\frac{1}{B}} \rightarrow$ orthogonal

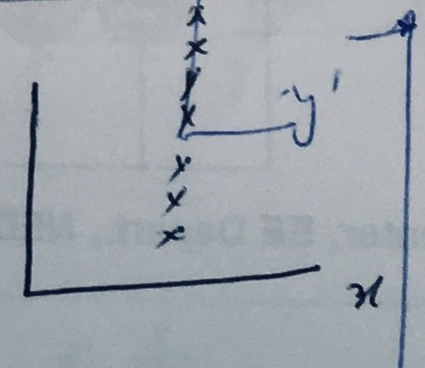
Minor Axis

Major Axis

- SVD → single value Decomposition
- EVD → eigen Value Decomposition







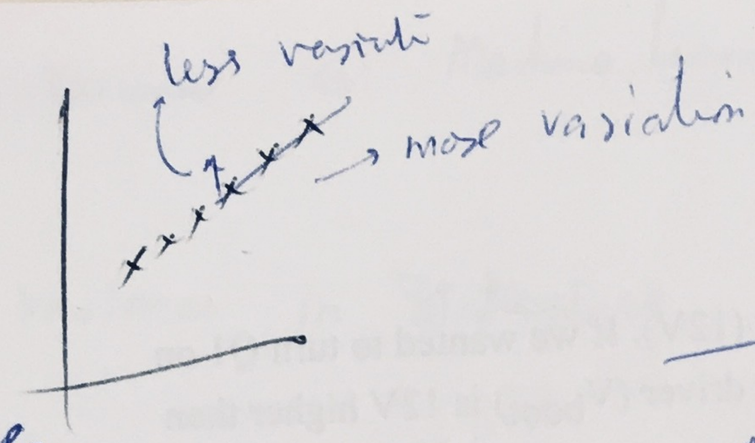
~~Here, Eigen value & regression~~

- In all 3 cases, PCA can be applied. data doesn't have to be strictly 1D.

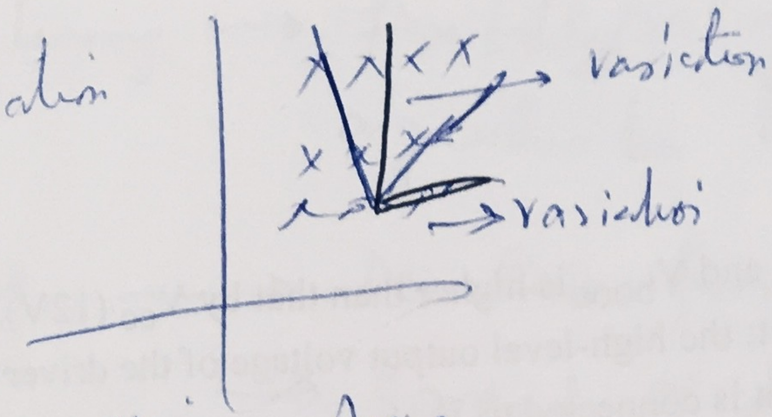
here regression mayn't work as $m \rightarrow \infty$ but PCA is vector based. & it'll work.

- Does major Axes dominate (Eigenvalue value or Importance Value)

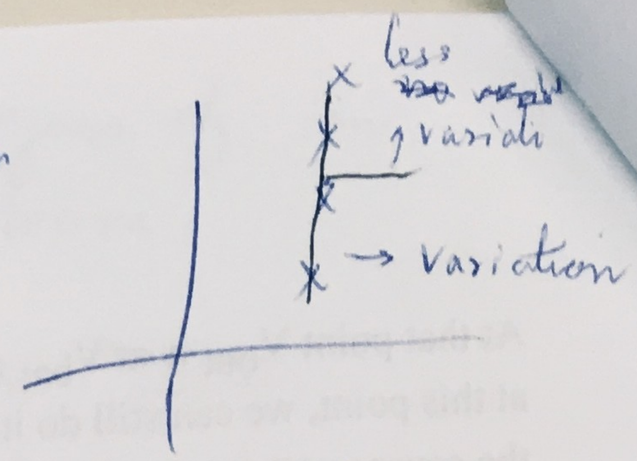
Major Axes dominate → There is more variation in one dimension than other



Major Axis dominates



Major Axis doesn't dominate



Major Axis dominates

* Either major axis dominates or not, minor axis doesn't dominate. B/c major axis is the one which has max variation, its also possible that both axes have same variation.

⇒ Measurable vs. Latent features:

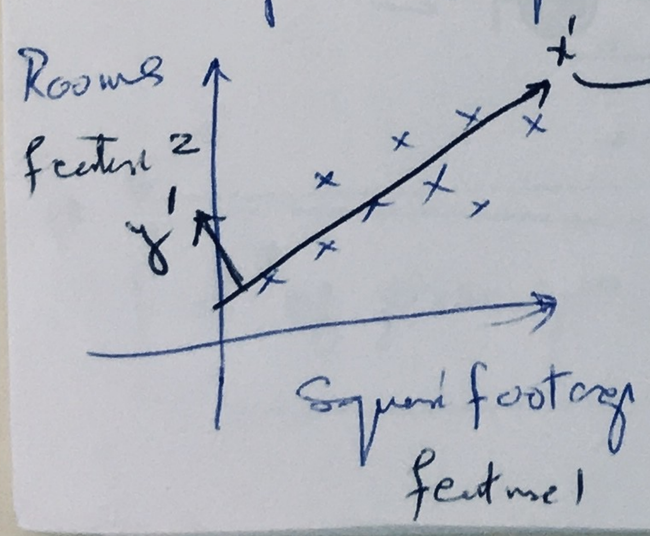
* Given features of a house, what is its price
  * DT
  * SVM
  * Regression → continuous output.
    → not interested in classification

* Measurable	Latent
* Square foot	size
* # of room	
* School Ranking	neighbourhood
* Neighbour Safety	

Latent → Can't be measured directly, found via Measurable features

* PCA is used to reduce # of features

Example: (Square Footage + # of rooms → size)



principal Component → composite feature

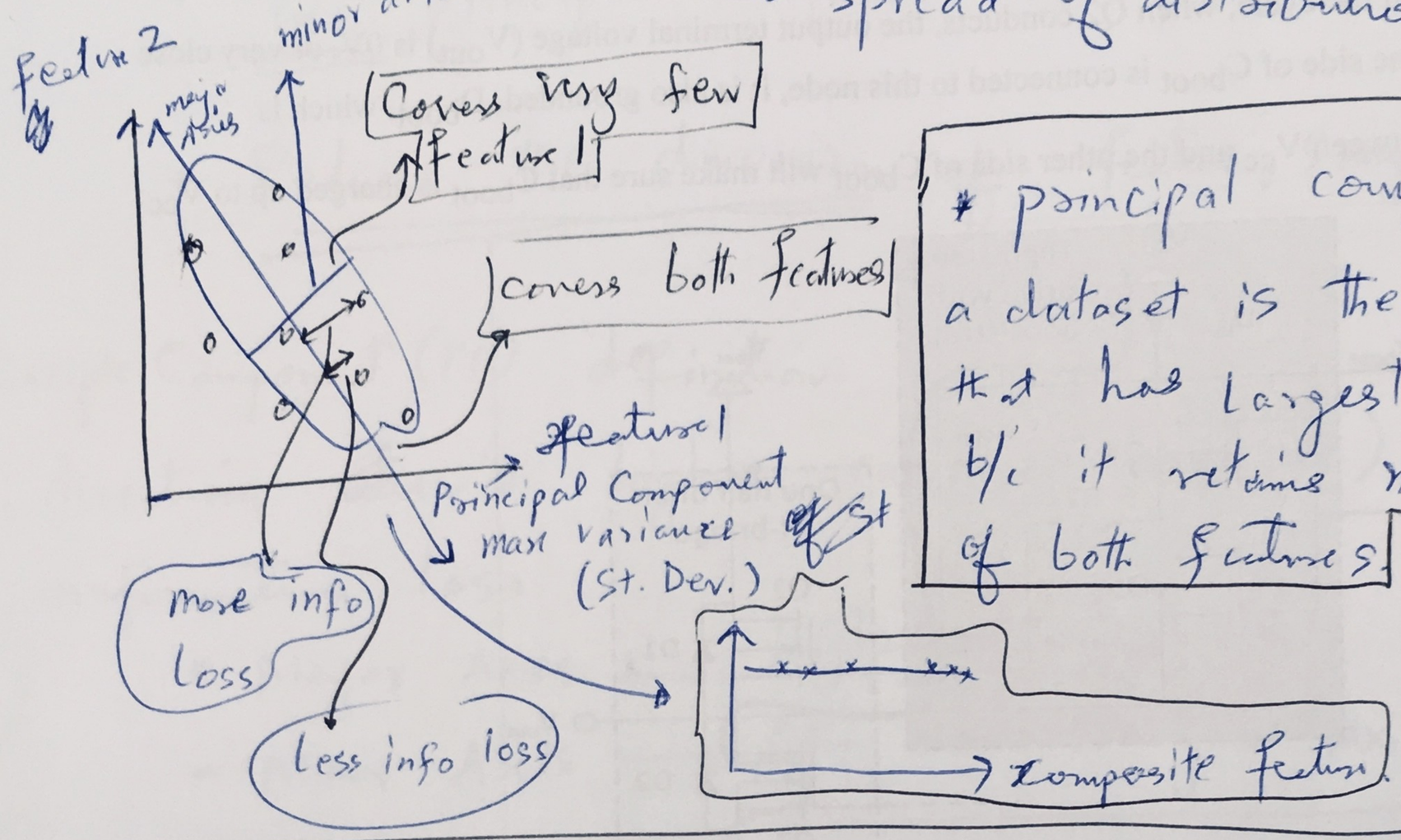
Not regression

* in regression, we predict new value &/or find relation.

* in pca, we try to reduce # of features to a composite feature

* Variance in Machine Learning → Flexibility/willingness of an algorithm to learn.

* Variance in Statistics → Deviation from mean.
  → Spread of distribution



* principal component of a dataset is the direction that has largest variance b/c it retains max. info of both features.

* Information loss → Sum of distances from data points to the principal component.

* If we take minor axis as principal component, then we've more information loss.

Maximal Variance → Minimal Info loss

A, B] PCA → x
C, D] PCA → y
Regression →
PCA
* 1st principle component → x
* 2nd principle Component → y

** # of principle Components = min(# of features, # of Samples)

PCA definition:

- Transform input features (multivariate dataset) into principle Components (which are orthogonal)
- use principal Components as features.
- Reduce the dimension of feature, sample matrix

Principle Component (PC) definition:

- directions with high variance (Statistical) i.e min information loss.
  - Major Axis → Highest variance Axis
  - Minor Axis → Other than major Axis

- Importance of PC ∝ Variance of data along this PC

- All PCs are perpendicular to each other i.e they are orthogonal to avoid overlapping — Same as OFDM (orthogonal frequency division Multiplexing) in signal transmission — product of consecutive PCs is 0 (as orthogonal: $\cos 90 \rightarrow 0$)

- If PCs ≃ # of features Then, we get no advantage or feature reduction from PCA

→ when to Use PCA:

* find Latent features which are driving the patterns in the data.
* Reduce dimension
  - visualize high dimension data
  - reduce noise.
* Major Axis — 1st PC → actual data pattern
* Minor Axis — 2nd PC → noise variations.
  - preprocessing for clustering, regression & classification.

→ PCA for facial Recognition:

* Pictures of faces has high dimensionality
* general patterns — that could be captured in smaller dimension (2 eyes, 1 mouth....)

→ 1MP camera will have 1 million pixels if used in raw pixel space.

High Dimension Problem
- → Too many features for too few data
- → Overfit
- → High variance
- → Less prediction accuracy
- → Slow especially SVM

* PCA is called Eigen Faces when applied for facial recognition

Principal Components of face data

* F1 score:

150 features → f1 ≃ 83.2 %

70 feature → f1 ≃ 84.96 %

10 features → f1 ≃ 49 %

* if SVM is run over images, then it has to train on 1000s of features. It'll be very slow & also less accurate b/c of overfit (Hyges Phenomenon)

But if SVM is run with principal components then it's very fast as ~~1000 (few~~ thousands of features are reduced to ~~few~~ hundreds of most imp. principle component. Also, accuracy is high.

1850 feature → 100 principle component

→ Deciding on # of principal component.

* Start with 10% of # of features.
* Then increase/decrease # of eigen faces (principal components) one see the f1-score.
* But there is not any fixed formula to find best # of principle components.
* Never perform feature-scaling (minmax scale) before PCA as a lot of info will be lost in this way. — feature-scaling CAN be done by PCA.

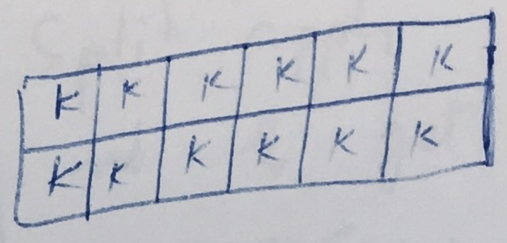
→ Validation

- why use Train/Test split:
  - gives estimates of performance on independent dataset.
  - Catches any overfitting - ~~it~~ if there is overfit, the ~~classifier/clustering will~~ classification/Clustering/regression will perform poor on ~~train~~ test data.

train_test_split

- input → features, labels, test_size
- output → train feat., test feat., train labels, test labels

⇒ K-Fold Cross Validation:



Leave one out strategy

- Using All the data for training.
- Using All the data for testing
- We do it by dividing data in k Bins
- Each time we use 'K' as test & (nk − 1) as train.
- iterate over all k Bins & then average the score of prediction on nK Bins.

So:

- Min. training time → Static Split Train/Test
- Min. runtime → Anyone of the two.
- Max Accuracy → K folds cross validation.

- Kfold just splits the data into # of folds you define. Better way is to shuffle data using shuffle = True.

- If shuffle is not possible b/c of class imbalance then StratifiedKFold( ) should be used instead of KFold( ).

KFold( ) just splits data without class consideration.

StratifiedKFold( ) → splits data so that each data sample has approx. same # of classes data.

Split apply not & get average score

13.

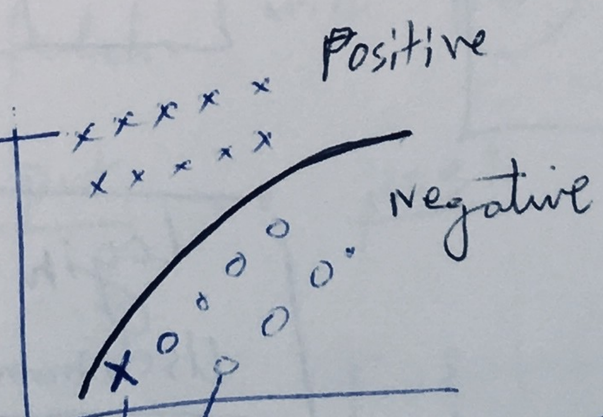
Now apply PCA.

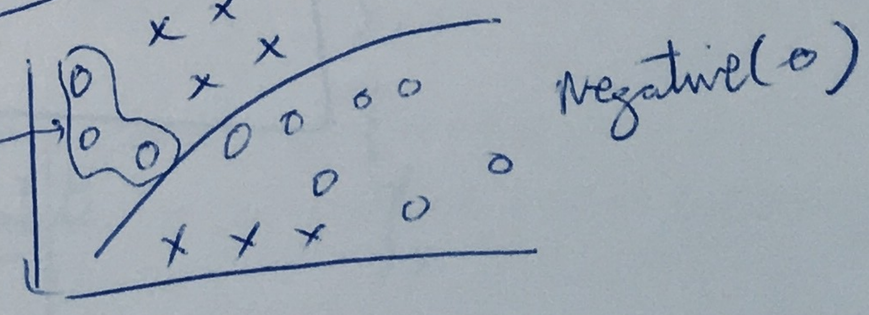
→ Evaluation Metrics:

* accuracy = $\frac{\text{\# of item labelled correctly}}{\text{All items}}$

* Confusion Matrix:

Actual Class	Pos.	Neg
Predicted Class Pos	x	x
Predicted Class Neg	x	x





Actual Class	x	o
Predicted Class x	5	3
Predicted Class o	3	7