Machine learning

(1) Data Preprocessing

→ load data pd.load_csv()
→ define dependent and independent variable I = Imputer() → simple Impute
→ Take care of catagorical data
→ For independt variable take care of catagorical lable into one hot vector
→ splitting into test-train data
→ Feature scalling -
- Independent variable should be on same scale
- because Euclidean distance $= \sqrt{(x_2-x_1)^2+(y_2-y_1)}$ is calculated in M.L between observations
  Equation
- standardization

$$\Rightarrow X_{stand} = \frac{X - mean(X)}{standard\ deviation(X)}$$

$$\Rightarrow X_{norm} = \frac{X - min(X)}{max(X) - min(X)}$$

- Code :-
  Use standard scalar from sklearn preprocessing

- make sure to fit standard scalar to train and use it for both train and test.

- Scalling of dummy variable is not required. ~~increase of~~ because information of one hot encode can be lost.

→ Data preprocessing template

→ load data
→ define independent and dependent variable
→ split test and train
→ Use/UnUse feature scalling.

Scanned with CamScanner

(2) Simple Linear Regression

Simple Linear Regression

~~$Y = b_0 + b_1 * (x_1)$~~

$Y = b_0 + b_1 * x_1$

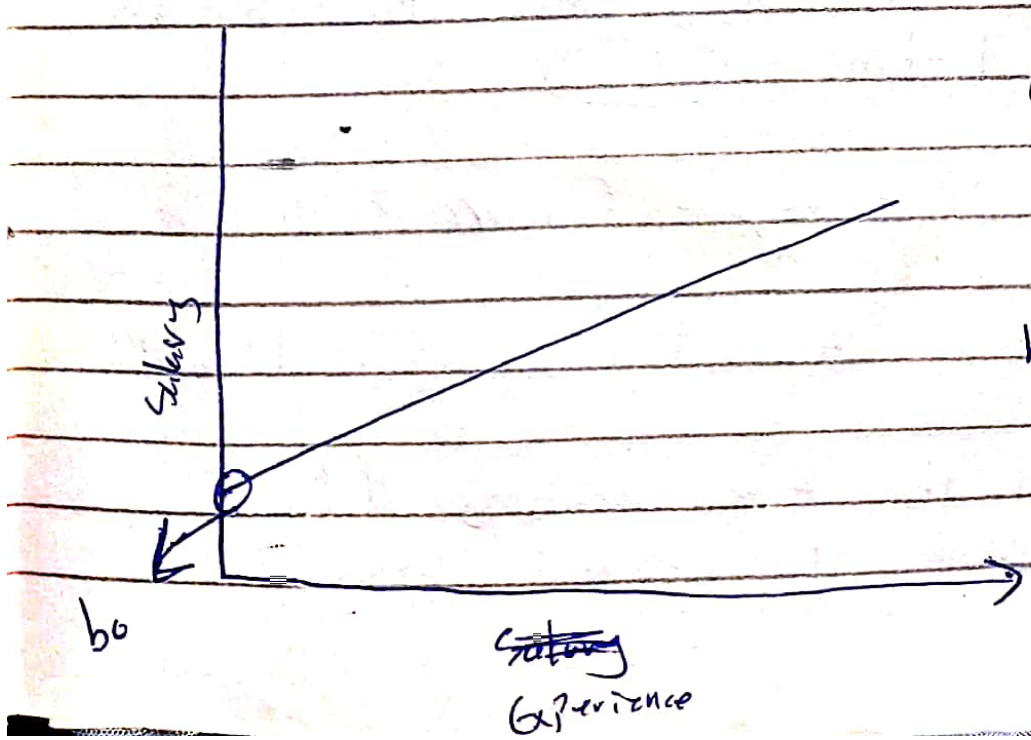
(i.e. $y = mx + c$)

Y → Dependent variable (DV)

x_1 → Independent variable (IV)

b_1 → Coefficient (unit change in x_1)

b_0 → Constant

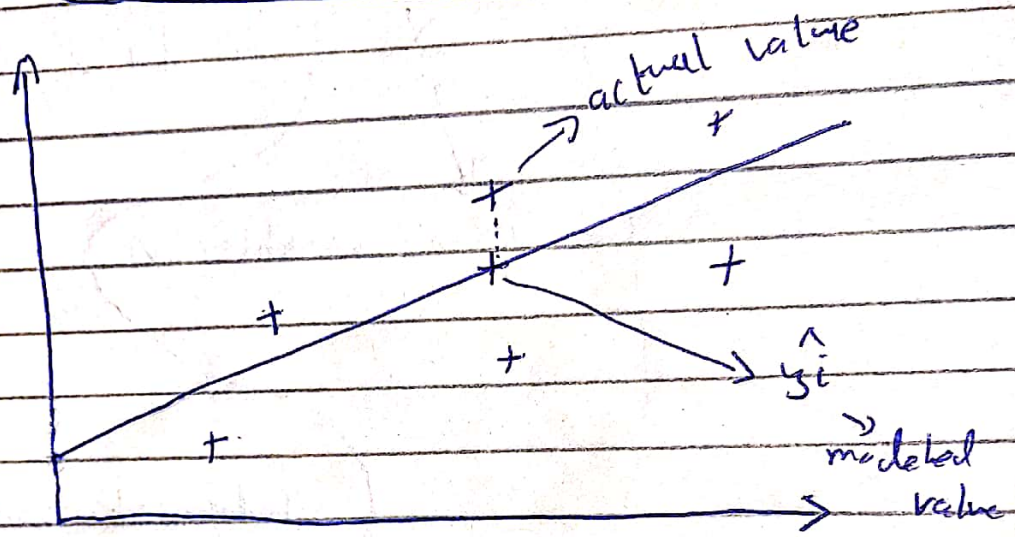


b_1 → slope of line

Continue

Simple Linear Regression

→ Ordinary least square



$$\text{Sum}(y - \hat{y})^2 \rightarrow \min$$

* * * * *

Pyton Code

Note → Don't require Feature scalling

→ Import sklearn . linear_model → LinearRegression

→ Fit it for training data.

→ Predict it for test data.

Scanned with CamScanner

[redacted]

dependant

→ Compare prediction with real dependant data.

- plot graphs or visualize data

~~⇒ Use plt. scatter (IV_train, DV_train~~

Use plt. Scatter(IV_train, DV_train, color '__')

for training data

Then,

Use plt. plot (IV_train, Predicted DV of training data, color = '__')

⇓

This plot simple linear regression line to our training set.

→ Give title
plt. title ('_____')

→ Give lable to graph
plt. X lable ( ) ; plt. Y lable( )

→ Show the graph
plt. show()

Scanned with CamScanner

→ Now visulize for Test data
(Repeat the process)

Multiple linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots b_n x_2$$

→ Intiuation

Assumptions of Linear Regression

1. Linearity
2. Homoscedasticity
3. Multivariate normality
4. Independence of errors
5. Lack of multi collinearity

Research this

Scanned with CamScanner

Dummy variable

For more than one dummy variable always, eliminate one variable i.e.

for 2 eliminate 1 → 1
for 20 elimrate 1 → 19
" 200 " 1 → 199

→ Always omit one dummy variable ←

HOW TO BUILD MODELS (STEP BY STEP)

5 methods

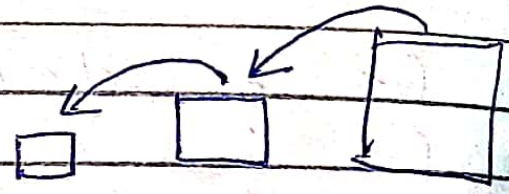
1. All in
2. Backward Elimination
3. Forward selection
4. Bidirectional Elimination.
5. Score comparison

Stepwise Regression. (also)

1) "All-in" cases:

- Prior knowledge ; OR
- You have to ; OR
- Preparing for Backward Elimination

2) Backward Elimination



Step1: select a significance Level to stay in the model (e.g. SL = 0.05)

Step2: Fit the full model with all possible predictors

Step3: Consider the predictor with the highest P-value if P > SL, go to [Step4], otherwise go to FIN

Step4: Remove the predictor

Step5: Fit the model without this variable

Scanned with CamScanner

3) Forward Selection

step 1: Select a significance level to enter the model (e.g. SL = 0.05)

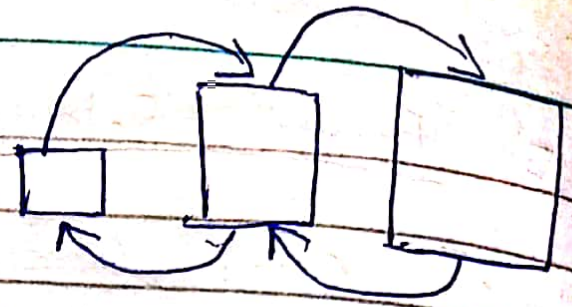
step 2: Fit all simple regression models $y \sim x_n$. select the one with the lowest P-value.

step 3: keep this variable and fit all possible models with one extra predictor added to the one(s)

step 4: Cosinder the predictor with the lowest P-value. If $P < SL$, go to step 3, otherwise go to FIN

FIN: keep the previous model

Bidirection Elimination:-



Step 1: Select a significance level to enter and to stay in the model e.g. SLEnter = 0.05 ; SLSTAY = 0.05

Step 2: Perform the next step of forward selection (new variables must have P < SLEnter to enter)

Step 3: Perform all steps of Backward Elimination (old variable must have P < SLSTAY to stay)

Step 4: No new variable can enter and no old varible can exit.

$\boxed{\text{FIN : Your model is ready}}$

Scanned with CamScanner

5. All possible models

Step 1: Select a criterion of goodness to fit (e.g. Akaike criterion)

↓

Step 2: Construct all possible models: $2^n - 1$ total combinations.

↓

Step 3: Select the one with the best criterion

↓

Fin: Your model is Ready

Example

10 columns means 1023 models

INSANE!!

Implementing of Multiple Linear Regression in python

1) load dataset

2) define x and y

3) ~~Split in train and test~~ Encode categorical Variable

4) Create one hot encoder for dummy variable

5) avoid dummy variable trap.

6) split train and test

7) Import → sklearn. linear model → linear Regression

8) create machine

9) train machine.

10) predict on test data

Scanned with CamScanner

Backward Elimination

(Python Code)

1) Library → import statsmodels.formula.api as sm

$y = b_0 x_0 + b_1 x_1 + b_2 x_2 + \cdots b_n x_n$

$x_0 = 1$

→ This does not consider b_0 unless you add $x_0 = 1$

2) add $x_0 = 1$ in x (a matrix) np.append( )

3) Create optimal team of IV (that are only statistically significant for a dependent variable profit)

4) first ~~ad~~ add all IV into optimal X.

Scanned with CamScanner

5) create new object for class
OLS → ~~optimal~~ ordinary least square

y obs-x

— = sm.OLS(—, —).fit()

6) Fit the model after giving OLS parameters

7) Call .summary()

8) look for highest P value
If P > SL (0.05) Remove that IV.

9) Fit again the model without that IV

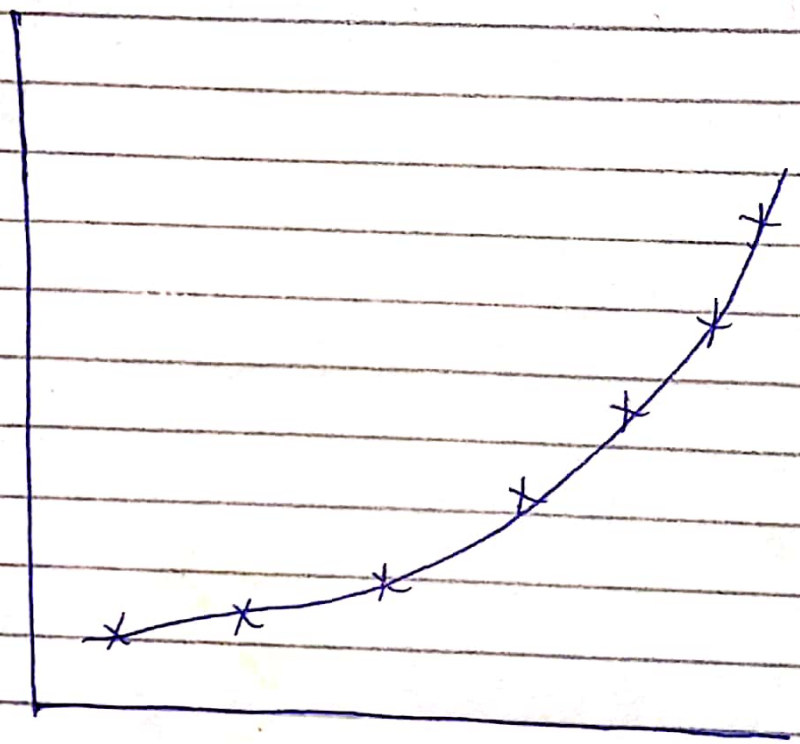
10) Repeat until P < SL

Scanned with CamScanner

Poly Numial linear Regression.

Polynomial Linear Regression

$$Y = b_0 + b_1 x_i^2 + b_2 x_i^2 + \dots b_n x_i^n$$



: PYTHON CODE :

1) Import a new class for polynomials

```
From sklearn.preprocessing import polynomial-
                                  -features
```

2) make a new IV. which containts polinomials (i.e. $\{x + x^2 + x^3 + x^7\}$)

— Polynomial feature (degree = —)

3) create poly_IV = _____ . fit_transform
(IV)

4) Now create a linear regressor
and fit that with new poly_IV

Scanned with CamScanner

Support vector Regression

→ support Linear and non-linear Regression.

→ Performs Linear Regression in higher dimensional space.

Building an SVR

1) Collect training set $\tau = (\vec{x}, \vec{y})$

2) choose a kernel and it's parameters as well as any regularisation needed.

3) Form the correlation matrix, $\vec{K}$

4) Train your machine, exactly or approximately to get contraction coefficients $\vec{\alpha} = \{\alpha_i\}$

5) Use those coefficients, create your estimator
$f(\vec{X}, \vec{\alpha}, x^*) = y^*$

→ Next step choose kernel
- Gaussian
  Regulirization
- Noise

Correlation matrix

$$K_{ij} = \exp\left(\sum_k \theta_k |x_k^i - x_k^j|^2 \right) + \epsilon \delta_{ij}$$

$$\vec{K} \cdot \vec{\alpha} = y$$

$$\Rightarrow \vec{\alpha} = \vec{K}^{-1} \vec{y}$$

Code in PYTHON

1. Import SVR from sklearn.svm.
2. Create objects of SVR class and add parameters
   slect 'rbf' or 'poly' of non linear
   → selecto kernal.
3. Fit your SVR object.

NOTE: MUST APPLY Feature scaling

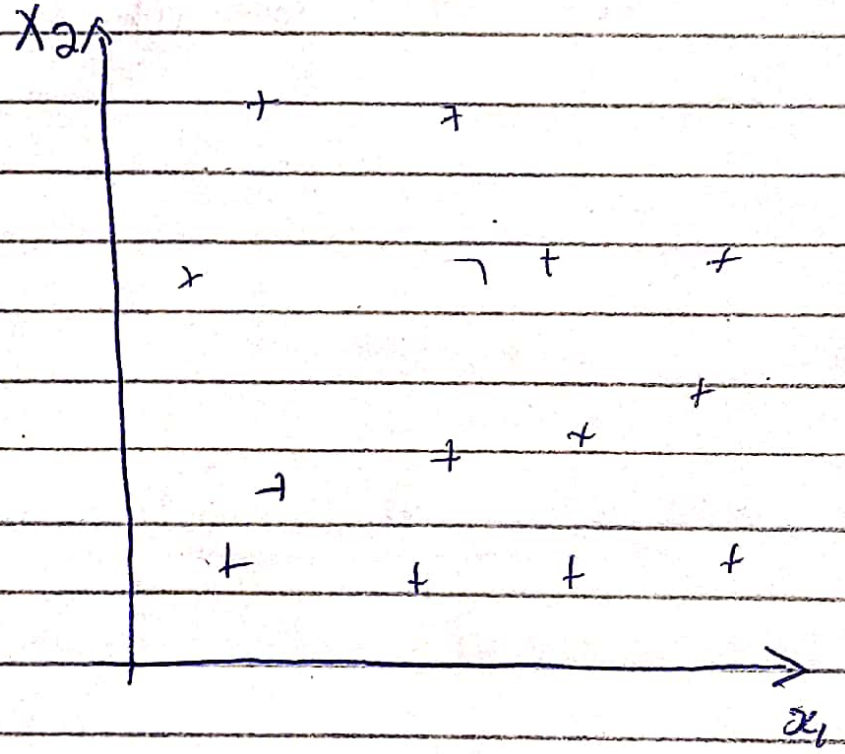
(4) after applying prediction make sure to inverse value of prediction.

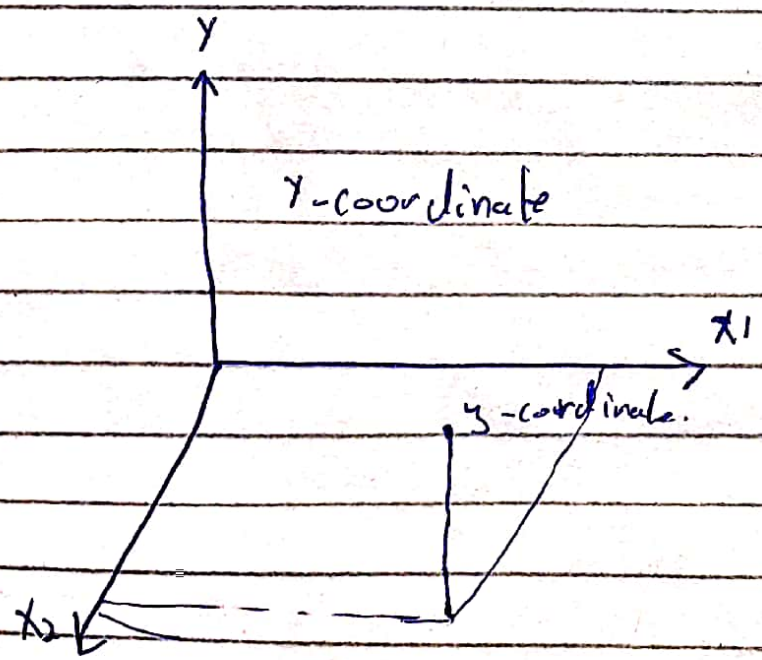
(5) use .inverse transform() from standard scalar libraly.

Scanned with CamScanner

Desicion Trees

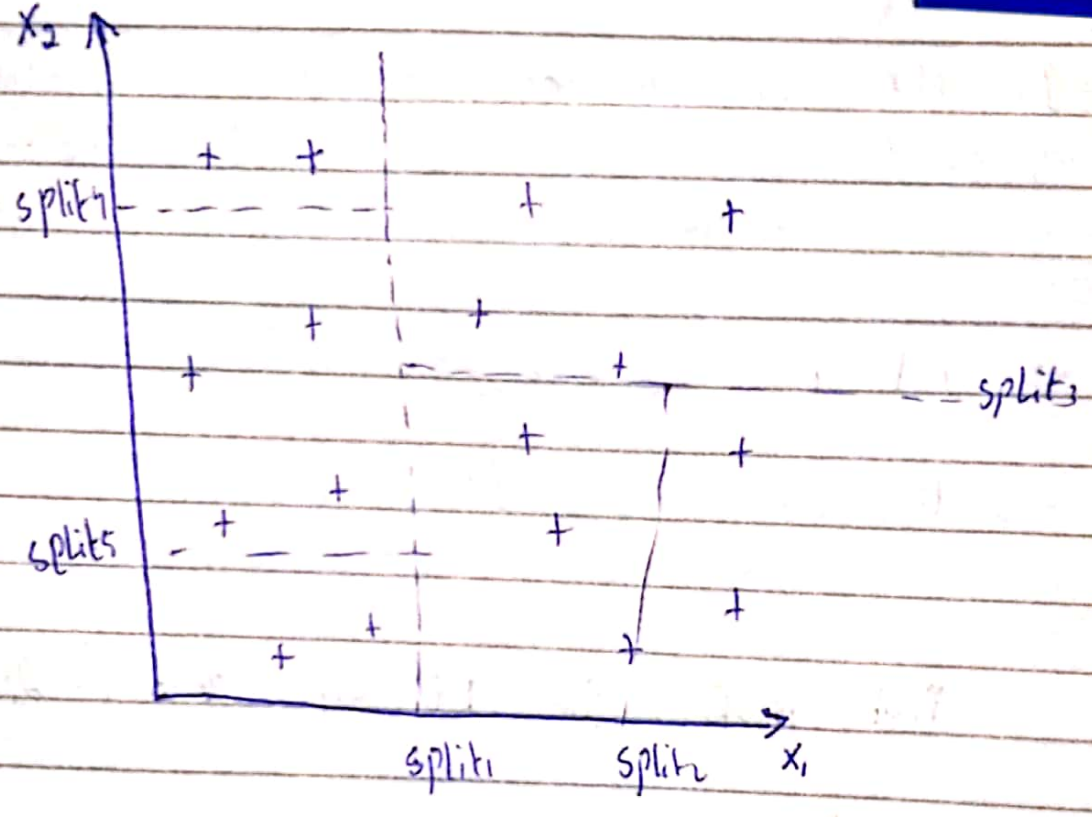
Regression Trees





Scanned with CamScanner

→ After running algorithm we have
→ Study information entropy.



~~Then~~ Then for prediction take average of every split block.

(Non-linear and Non-Continous model)

Python Code

(Desicion tree Regression)

→ Import class → from sklearn.tree import DesicionTreeRegressor

→ Create object of class and fill parameters.

→ specify random state

→ fit regression object to dataset.

→ Visualize with higher resolution

Not useful for 1D but good for 2D

Scanned with CamScanner

Random Forest Intuation

(Regression)

→ Essemblge learning.

: Same algorithum multiple time for a powerful prediction

(1) pick a random k data point From Training set.

(2) Build the Decision Tree associated to these k data points.

(3) Choose the number N tree of trees you want to build and repeate step (1) and (2)

(4) For a new data point, make each one of your N tree trees predict the value of Y to For the data point in question, and assign the new data point the average across all of the predicted Y values

Scanned with CamScanner

(Python Code):

Step 1 Import:

From sklearn.assemble import RandomForestRegressor

Step 2 Create object and input argument:

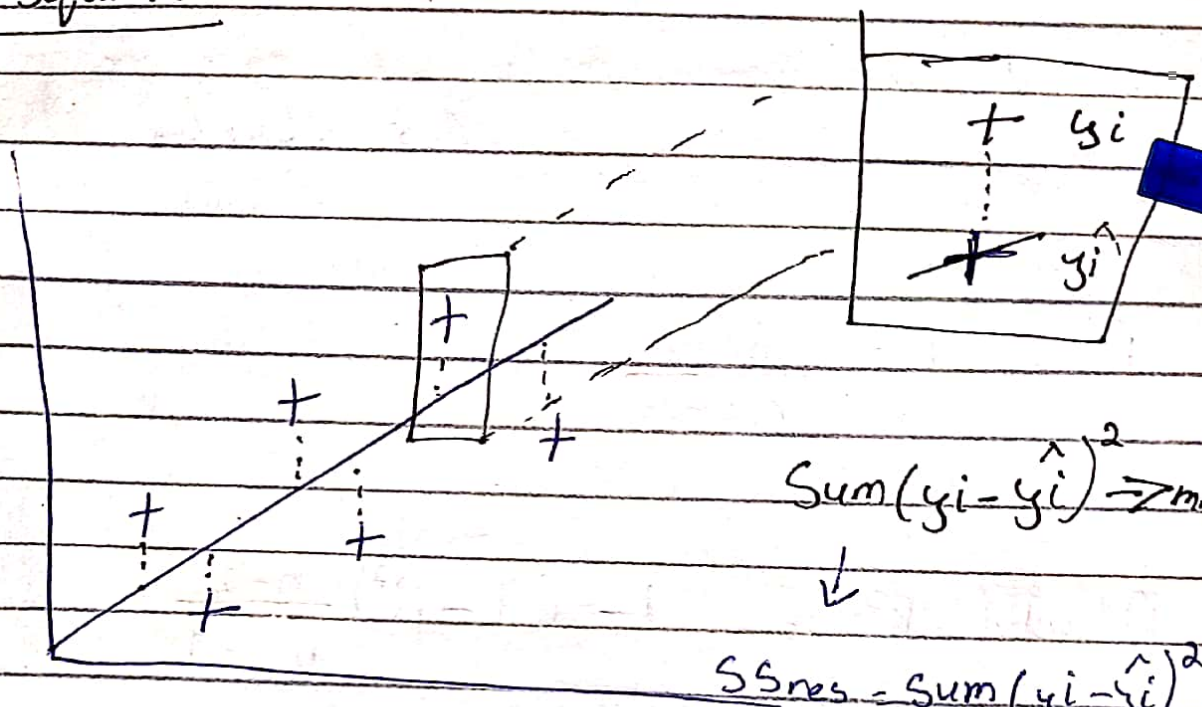
→ n_estimator
→ criterion default "mse"
→ random state.

Step 3 Fit regressor object to dataset.

Scanned with CamScanner

Evaluating Regression Models

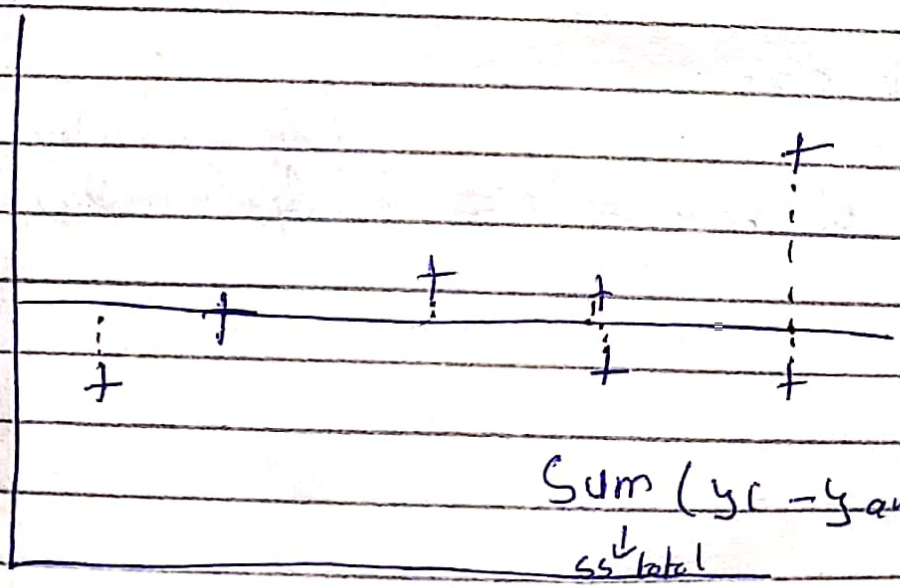
→ R squared



$$\text{Sum}(y_i - \hat{y}_i)^2 \Rightarrow \min$$

$$\downarrow$$

$$SS_{res} = \text{sum}(y_i - \hat{y}_i)^2$$



$$\text{Sum}(y_i - y_{avg})^2 \Rightarrow \min$$

$$\downarrow$$

$$SS_{total}$$

Goodness of fit

$$SS_{res} = \text{sum}(y_i - \hat{y}_i)^2$$

$$SS_{tot} = \text{sum}(y_i - y_{avg})^2$$

$$\boxed{R^2 = 1 - \frac{SS_{res}}{SS_{tot}}}$$

→ Closer to 1 greater the line of prediction

ADJUSTED R^2

~~⇒ Usefull in case of more ind~~

1) → Use for increase in no of regressors (i.e. independent variable)

2)

3) $$R^2 = 1 - \frac{SS_{res}}{SS_{TOT}}$$

4) $$\text{Adjusted } R^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

n = sample size

p = no. of regressors (# of independent variable)

Analyzing

→ In Backward elimination also see adj R^2.

→ If remaining I.V have +ve 'b' then they are related to D.V.

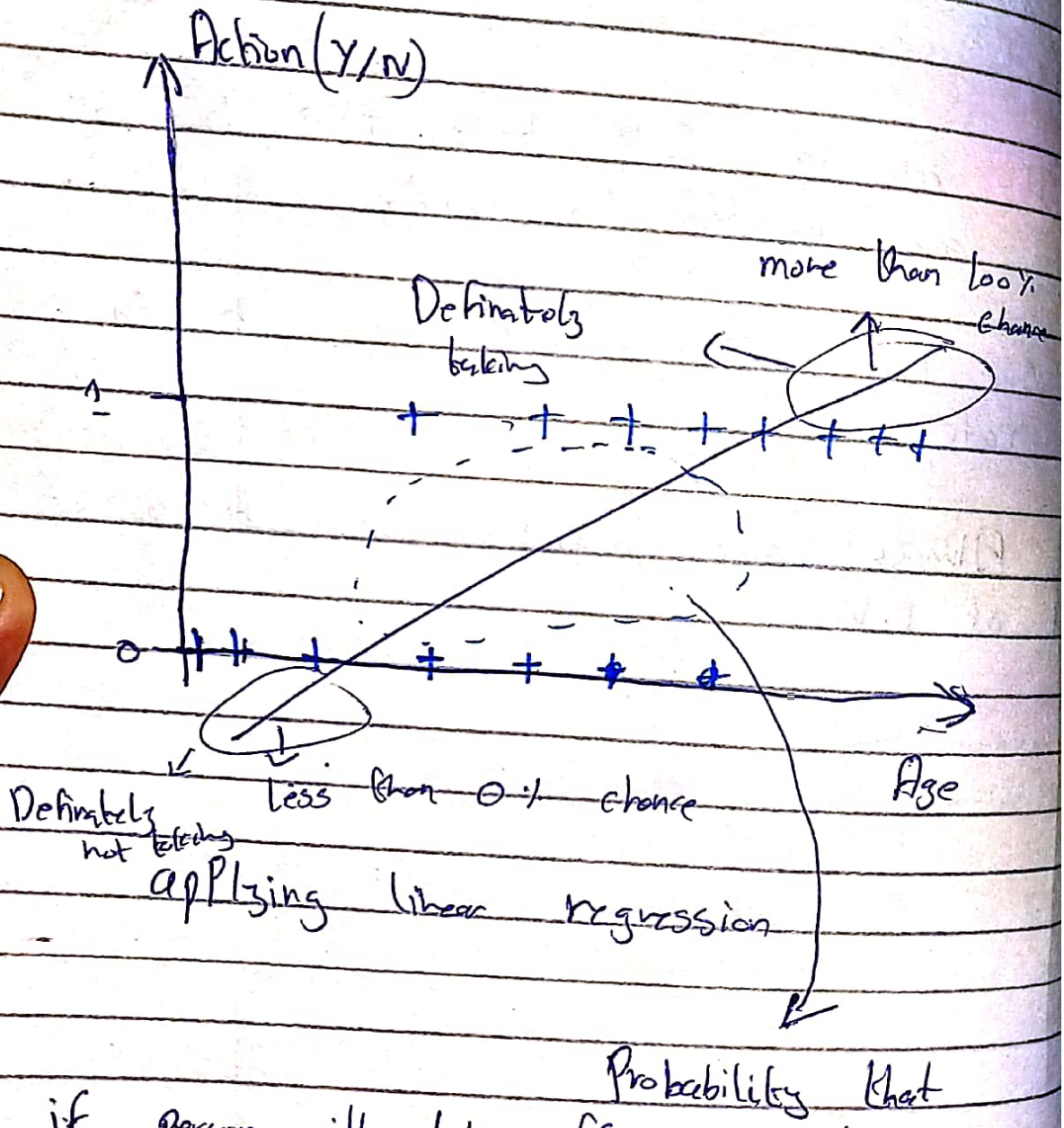
→ See the magnitude of 'b' in each I.V

Always see magnitude in terms of unit of I.V.

Scanned with CamScanner

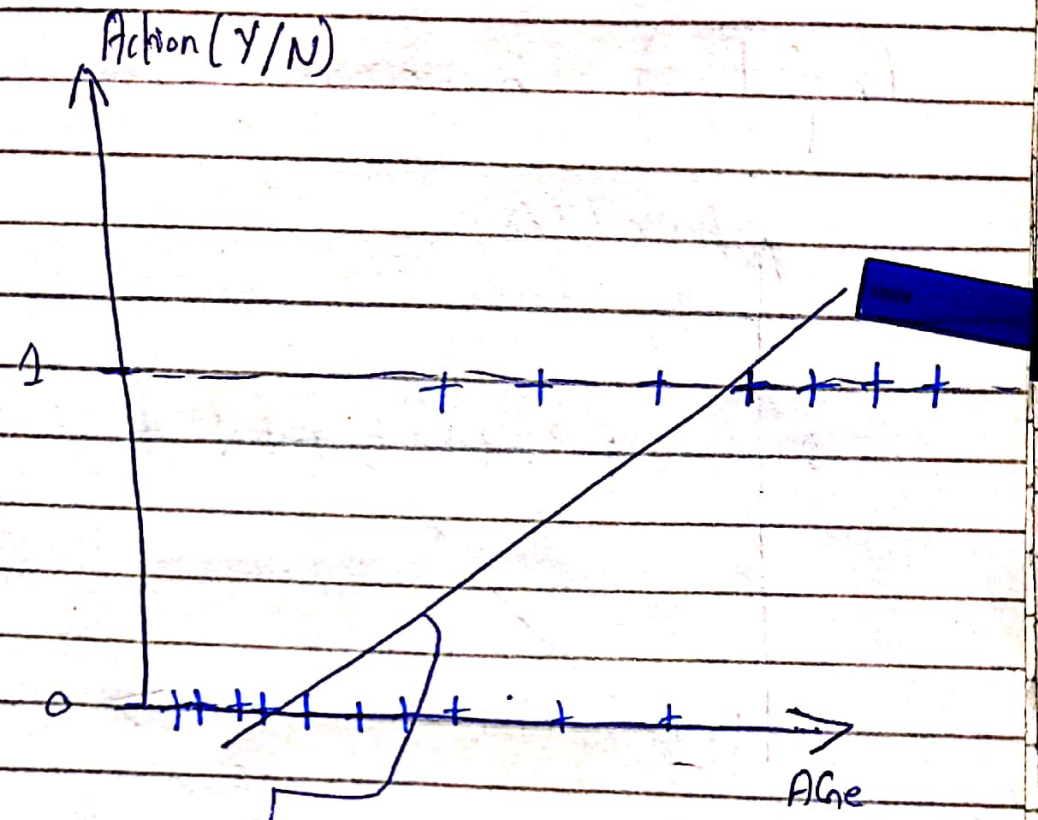
Classification

→ Logistic Regression



if person will take offer or not.

we can change Regression line to following



$$y = b_0 + b_1 x_1$$

Sigmoid: $P = \frac{1}{1-e^{-y}}$

after solving for y we have new equation for Regression line (now called logistic Regression).

$$l\left(\frac{P}{1-P}\right) = b_0 + b_1 x_1$$

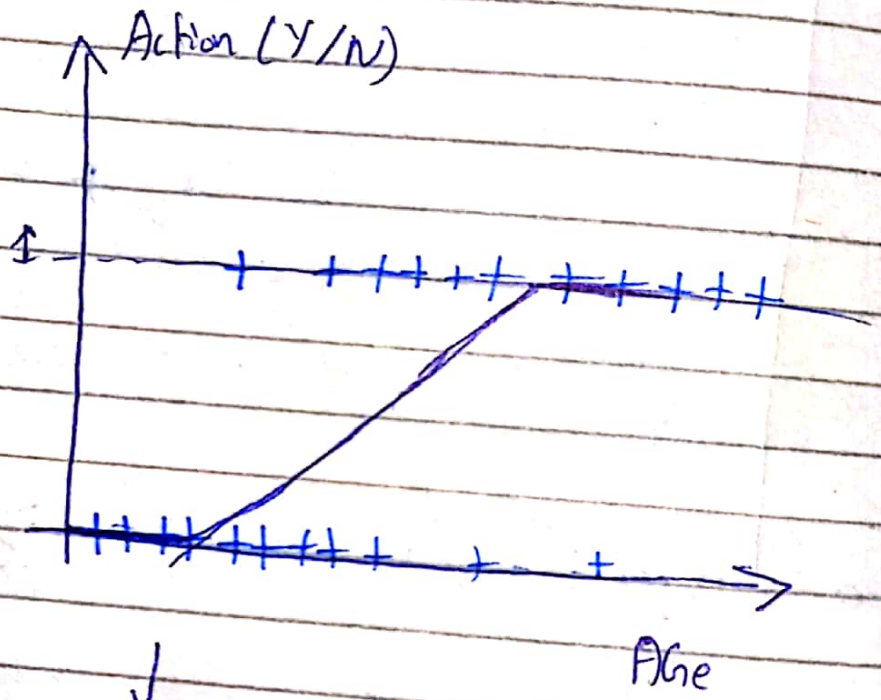
Scanned with CamScanner

So logistic Regression is...

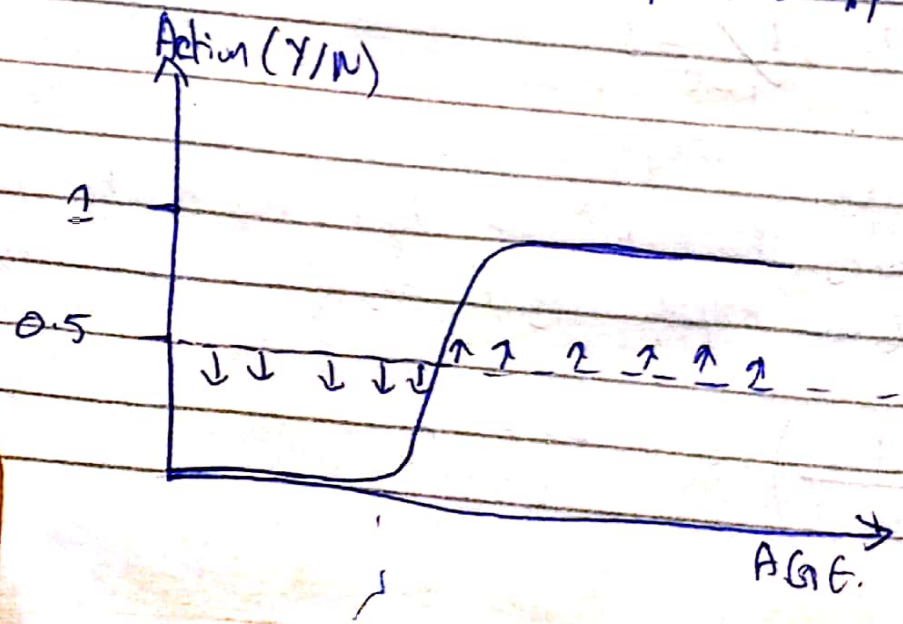
1) $$\ln\left(\frac{P}{1-P}\right) = b_0 + b_1 x_1$$

2.

3.



$\hat{P}$ (Probability) → P-hat → something we are predicting



Logistic Regresson on PYTHon

→ Apply feature scalling to both X_train and X_test.

→ From sklearn_linear_model import logistic Regression

→ create object classifier of class logisticRegression.

~~→ fit classifier and give set para~~ set

→ Set classifier parameters.

→ fit classifier.

Evaluate model

The confusion matrix

→ from sklearn.metrics import confusion_matrix function.

— = confusion_matrix ( ) enter parameters

Correct Prediction	Incorrect Prediction
Incorrect Prediction	correct Prediction

Scanned with CamScanner

Visualizing Results

→ Import Listed Colormap from matplotlib.colors

→ define local variable x-Set and y-Set.

→ create grid from meshgrid()

→ use plt.contourf to make contourf between predictions.

Scanned with CamScanner

K - Nearest Neighbour (algorithm)

Step 1 : Choose the number k of neighbors

Step 2: Take the k nearest neighbor of new datapoint, according to the ~~ec~~ Euclidean distance

Step 3: Among these k neighbours, count the number of data point in each catagory

Step 4: Assign the new data point to the catagory where you counted the most neighbors.

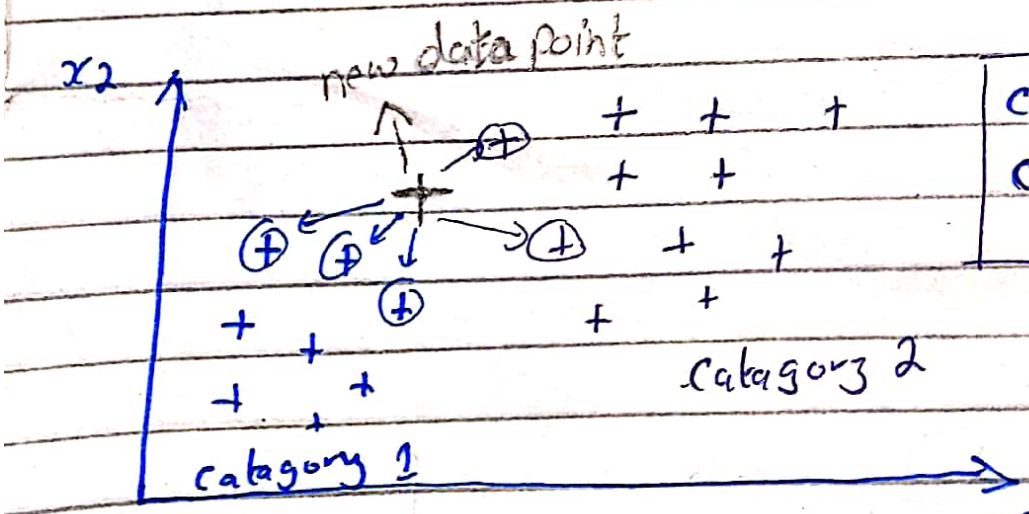
↓

$\boxed{\text{Your model is ready}}$

Euclidean distance $= \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

- - - - - -

Step1 : Choose the number of k neighbors = 5



Catagory 1 : 3 neighbors
Catagory 2 : 2 neighbors

→ So new data point lies in catagory 1

Scanned with CamScanner

K-NN (Python Code)

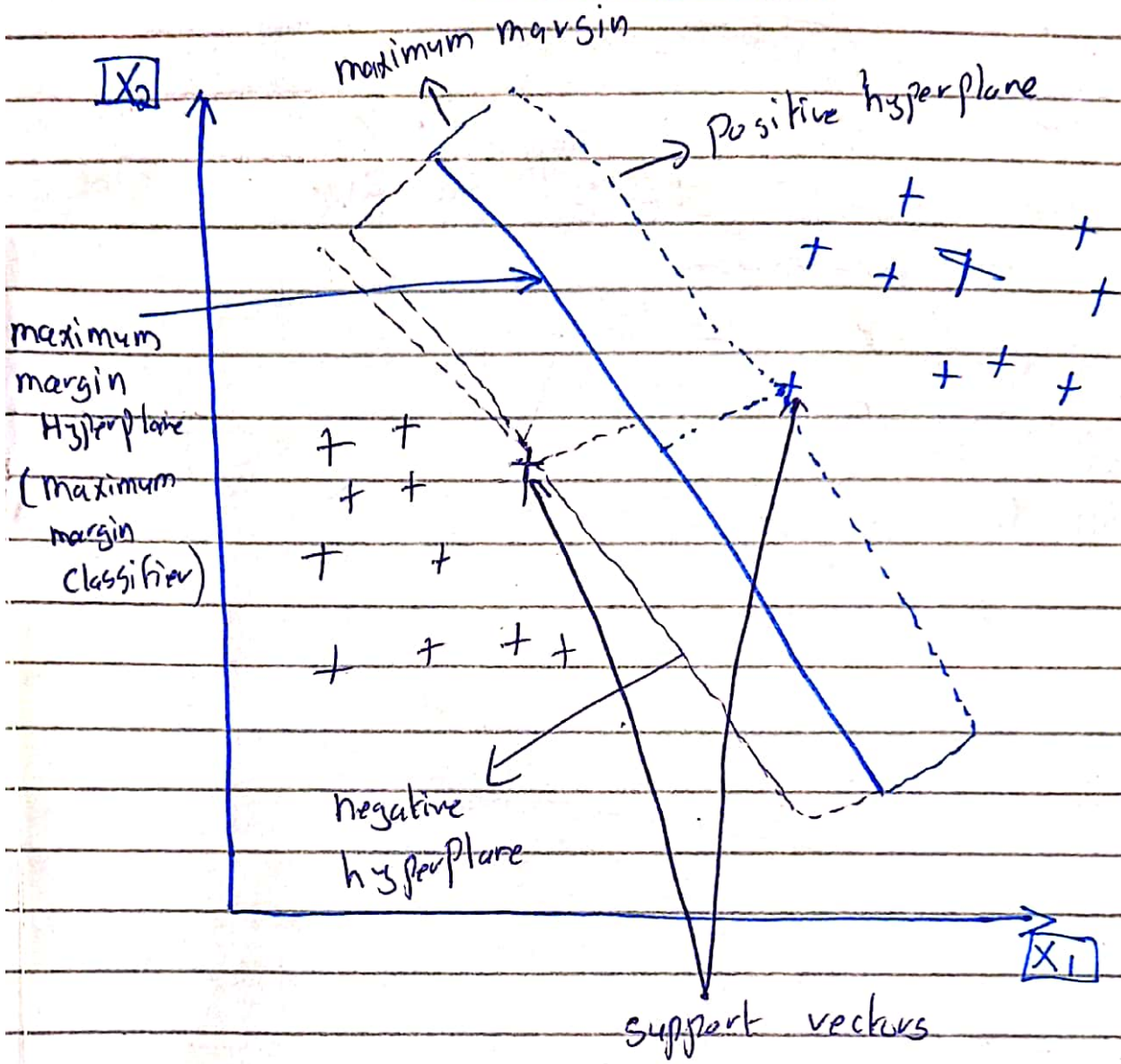
→ From Sklearn.neighbors import KNeighborsClassifier

→ create classifier object of KNeighborsClassifier
select parameter (n-neighbor)
metric (Euclidean)
p = 2 (for Euclidean distance

→ fit classifier.

Scanned with CamScanner

SVM INTUITION



SVM (code)

Python

→ From sklearn.svm import SVC.

→ create classifier object of SVC

→ input parameters.

kernel → linear,

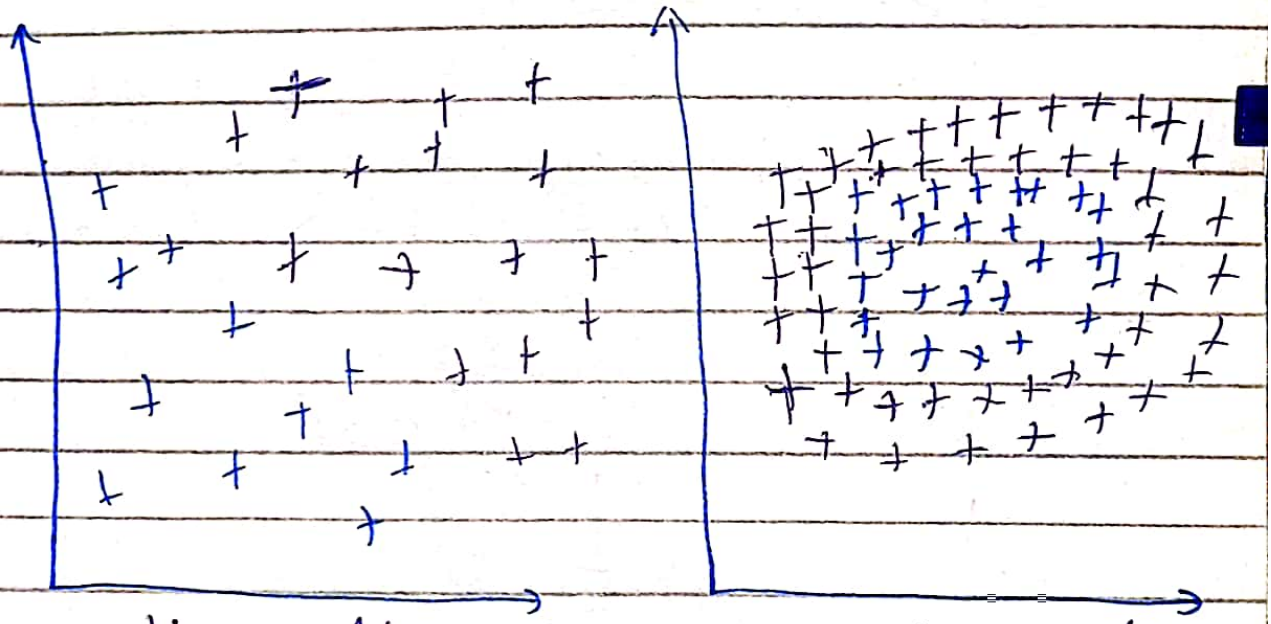
random state.

→ fit the classifier.

Scanned with CamScanner

Kernel SVM Intiuation

(For non linear data)



Linear data Non linear data

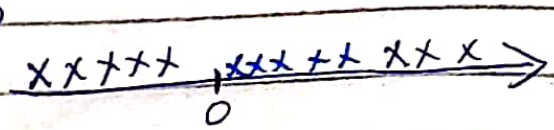
Method 1:

→ mapping to a higher dimension.

Suppose a 1-D data

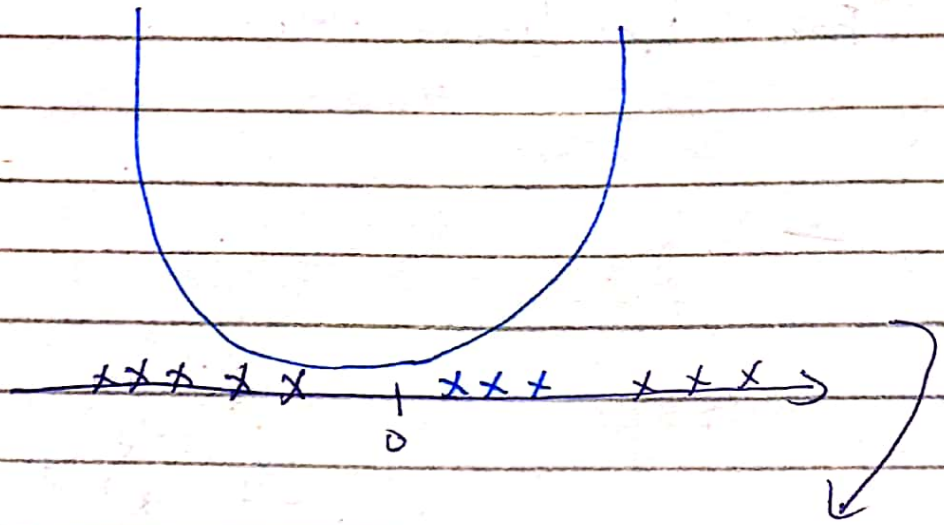
suppose we do $f = x - 5$

then data becomes

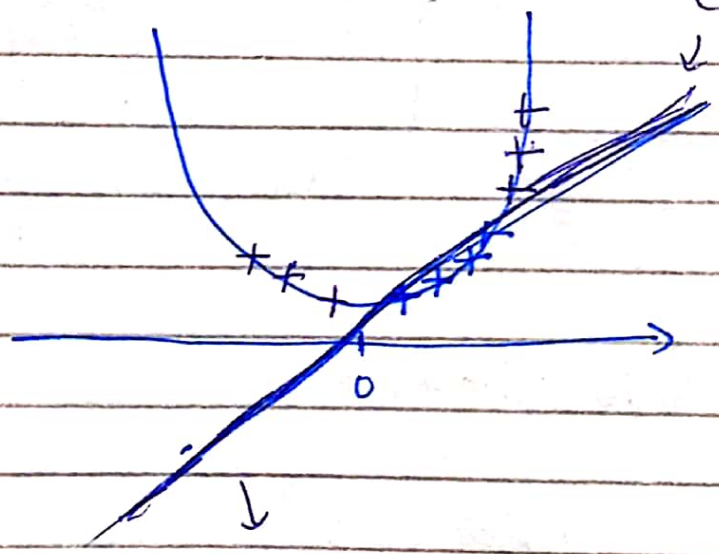


Scanned with CamScanner

then we apply square function

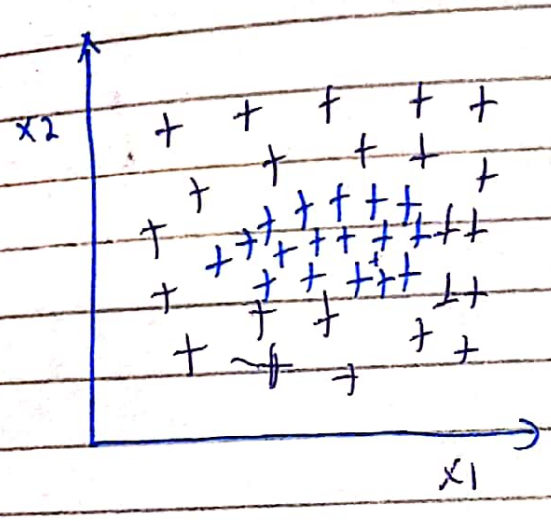


$F = (x-5)^2$



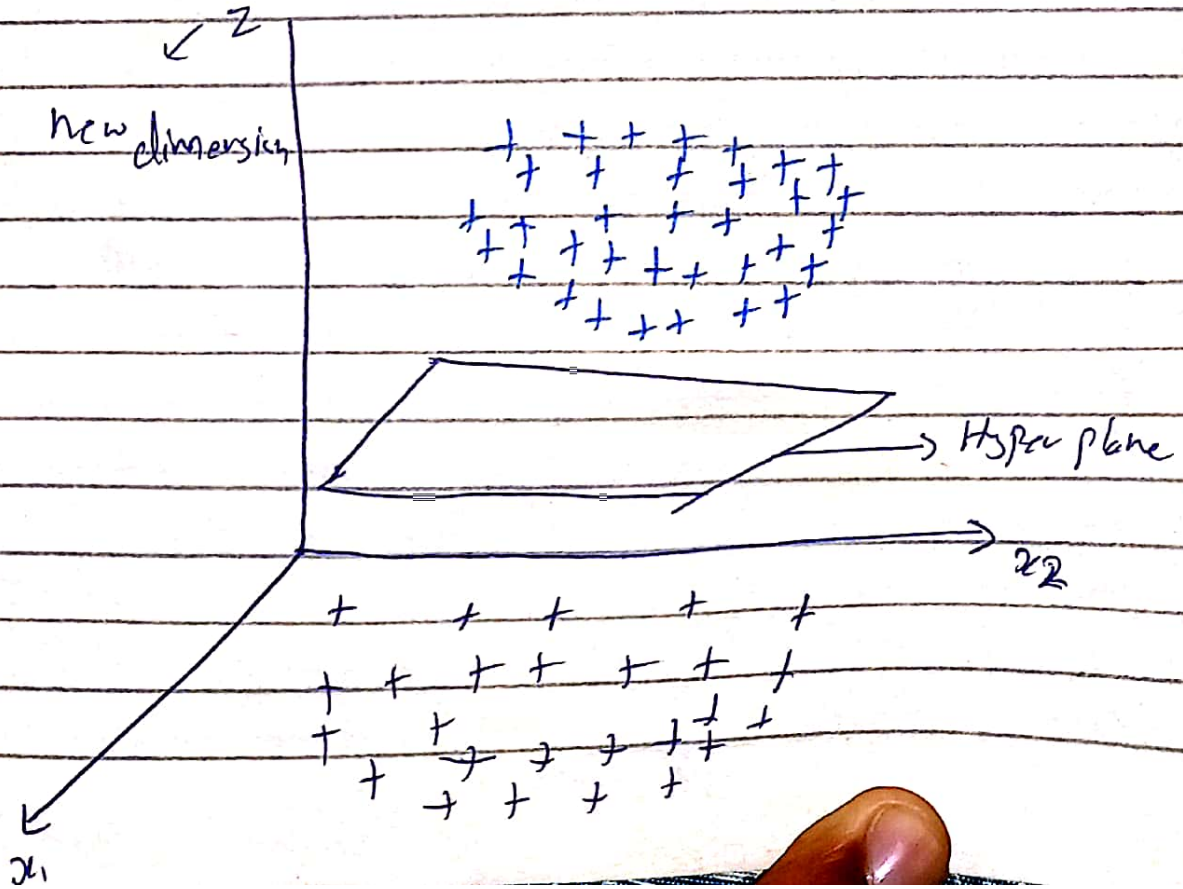
then draw linear line

similarly for 2D data.

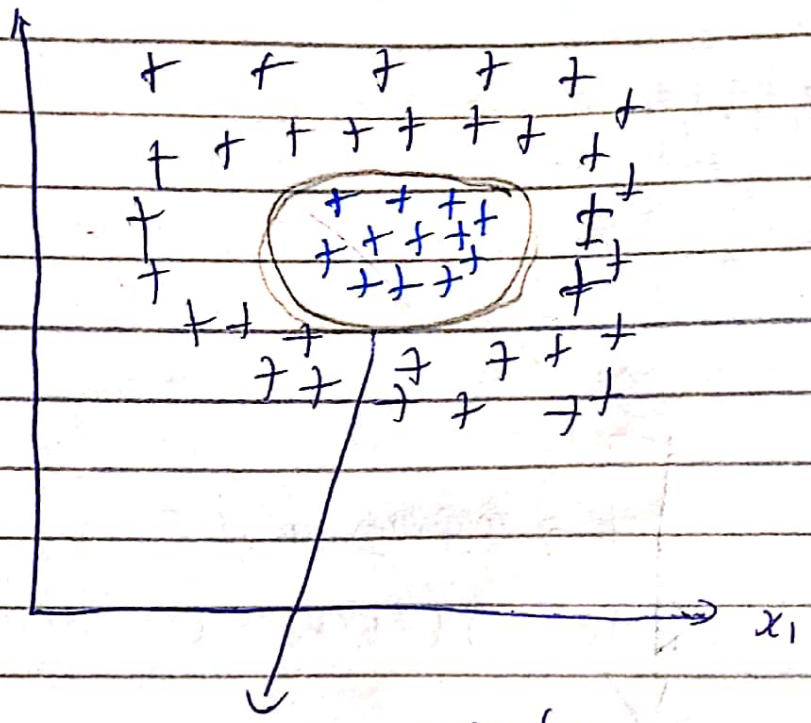


mapping function

$f(x_1, x_2) = (x_1, x_2, z)$



Then project



non linear separator

Scanned with CamScanner

method 2:

kernel trick

(Gaussian / RBF kernel)

$$K(\vec{x}, \vec{l}^{i}) \quad e^{-\frac{\|\vec{x} - \vec{l}^{i}\|^2}{2\alpha^2}}$$

(see mob pics)

TYPES OF kernel function

(See mob pics)

Go to following for 3D pics

m1kernels.readthedocs.io/en/latest/kernelfunctions.html

Scanned with CamScanner

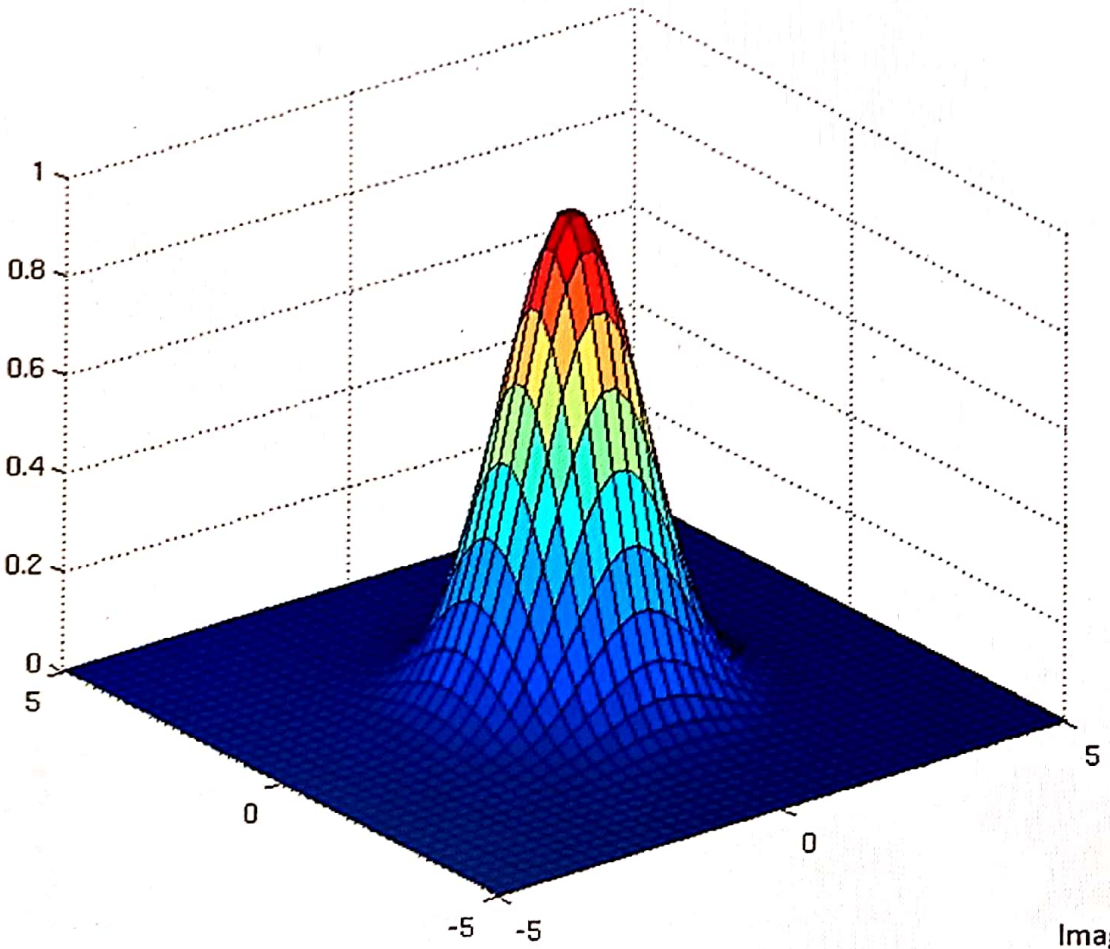
The Gaussian RBF Kernel

$$K\left(\vec{x}, \vec{l}^{i}\right) = e^{-\frac{\left\|\vec{x}-\vec{l}^{i}\right\|^{2}}{2\sigma^{2}}}$$

© SuperDataScience

Scanned with CamScanner

The Gaussian RBF Kernel



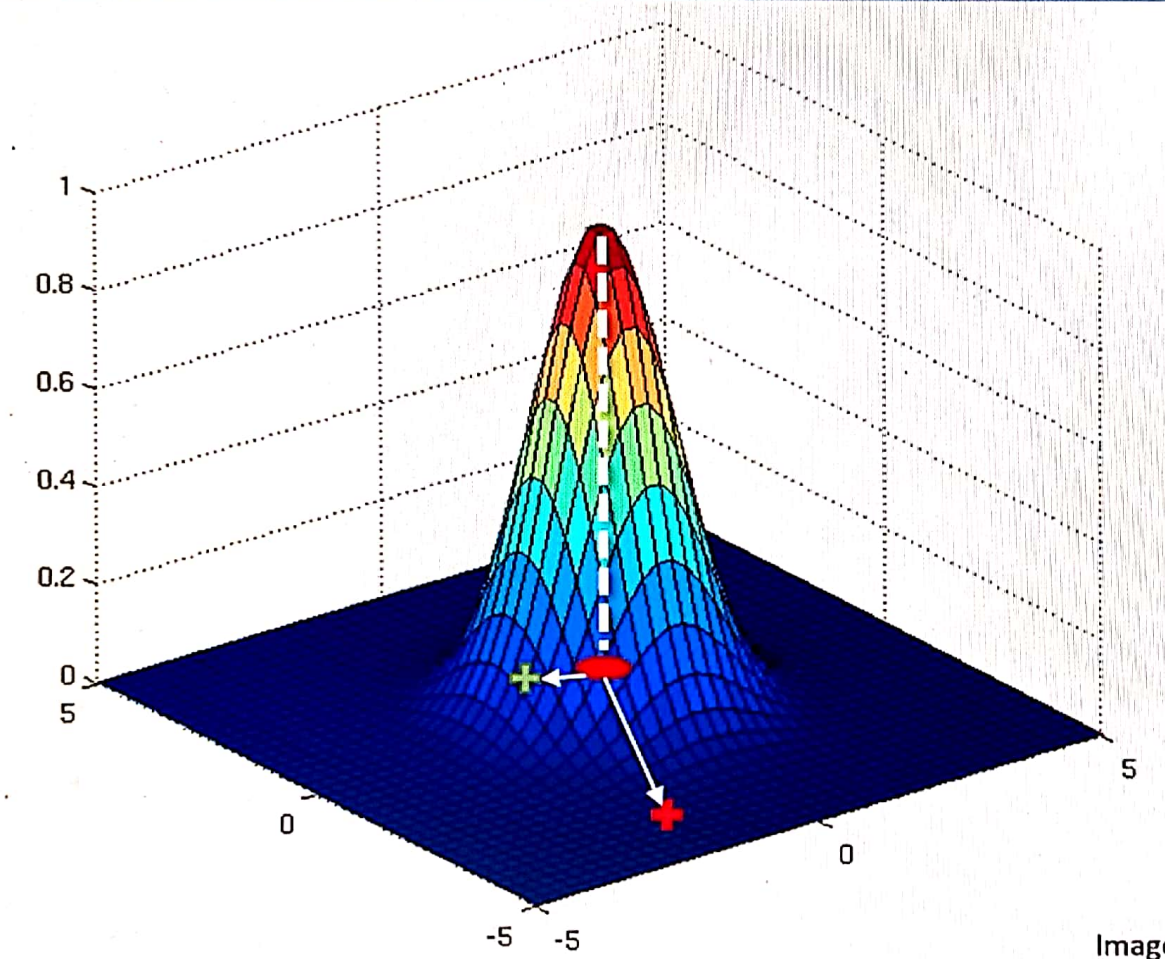
$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$

Image source: http://www.cs.toronto.edu/~duvenaud/cookbook/index.html

© SuperDataScience

Scanned with CamScanner

The Gaussian RBF Kernel



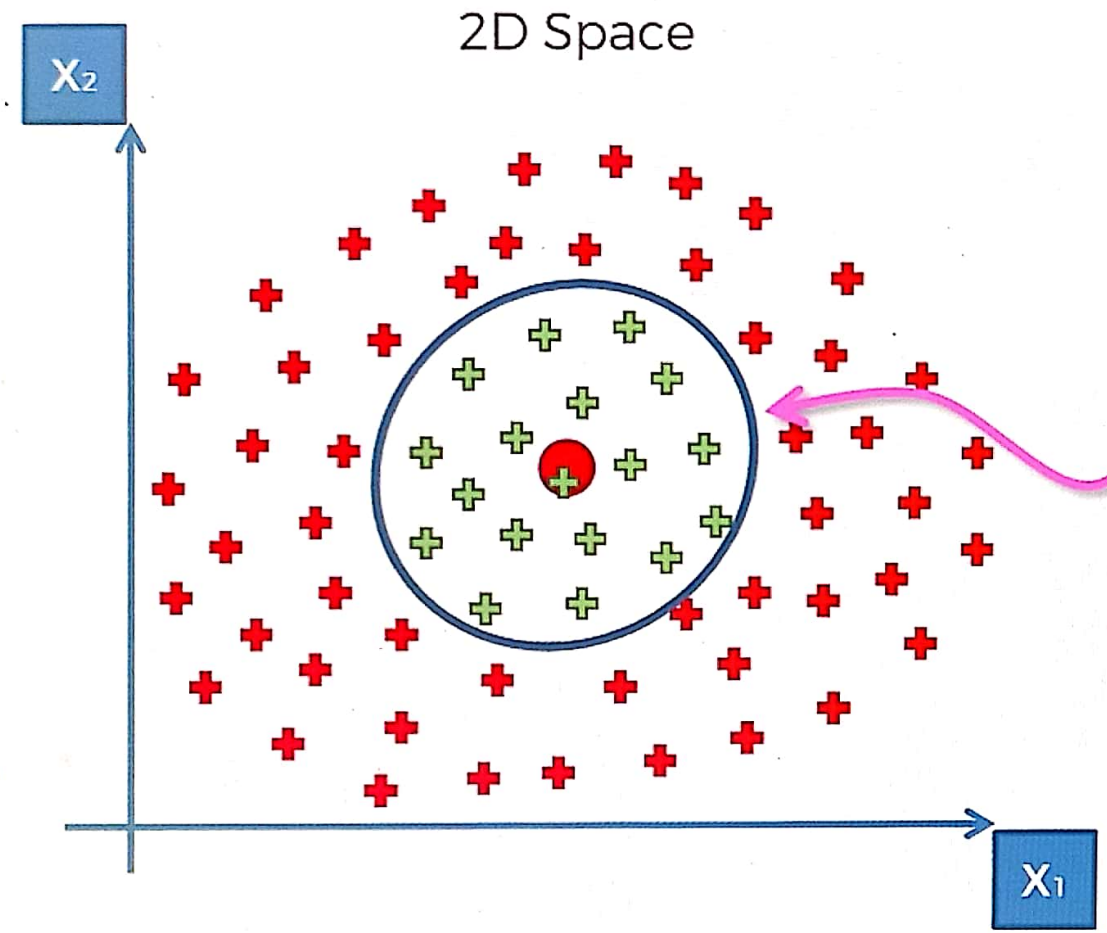
$$K(\vec{x}, \vec{l}^{i}) = e^{-\frac{\|\vec{x}-\vec{l}^{i}\|^2}{2\sigma^2}}$$

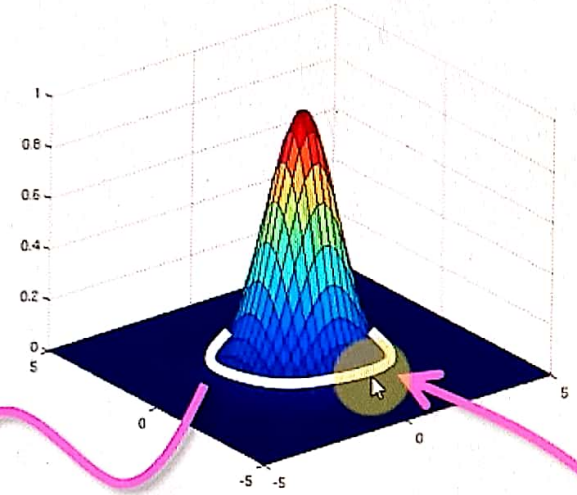
Image source: http://www.cs.toronto.edu/~duvenaud/cookbook/index.html

© SuperDataScience

Scanned with CamScanner

The Gaussian RBF Kernel



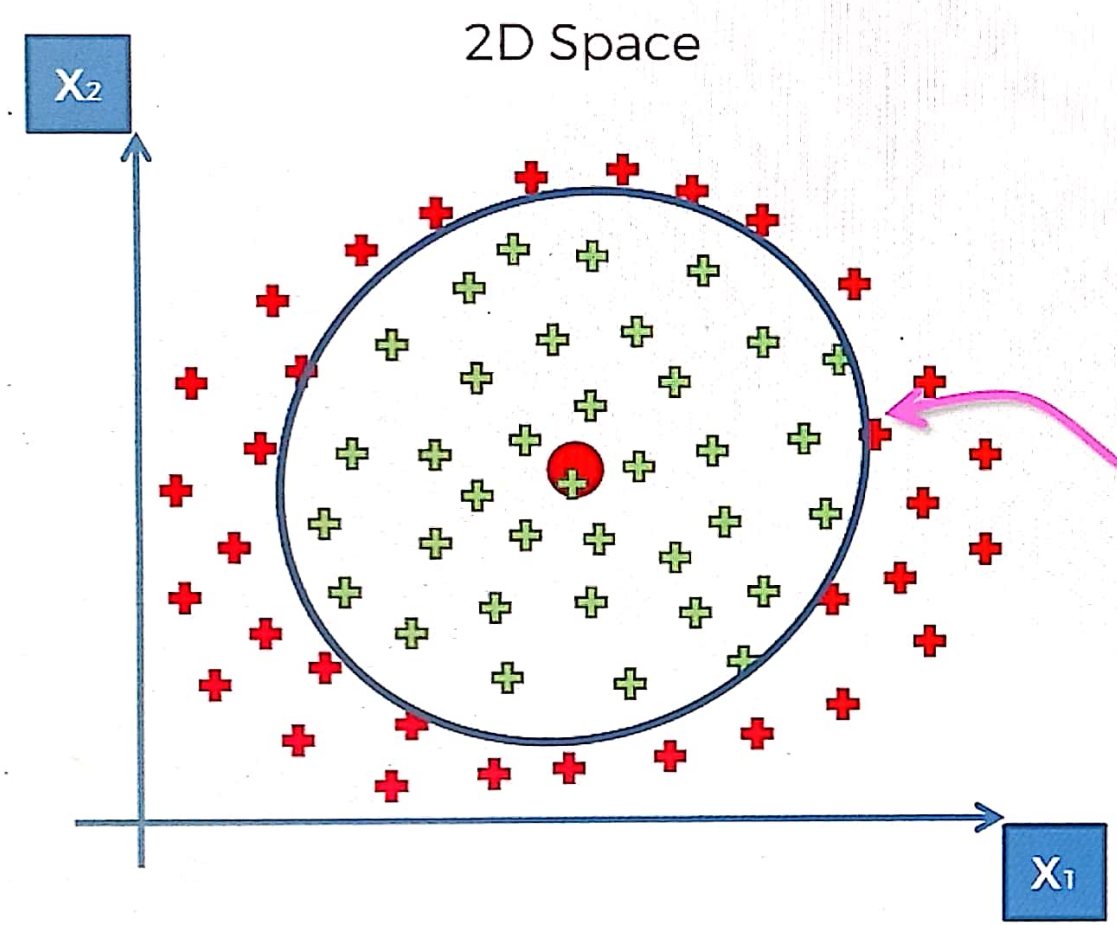


$$K(\vec{x}, \vec{l}^{i}) = e^{-\frac{\|\vec{x}-\vec{l}^{i}\|^2}{2\sigma^2}}$$

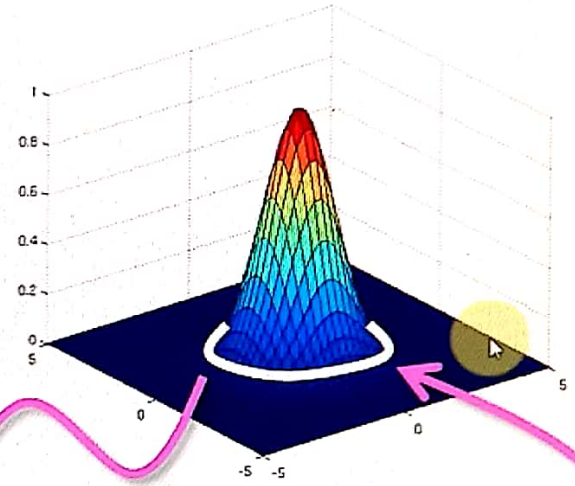
© SuperDataScience

Scanned with CamScanner

The Gaussian RBF Kernel



$\sigma \uparrow$

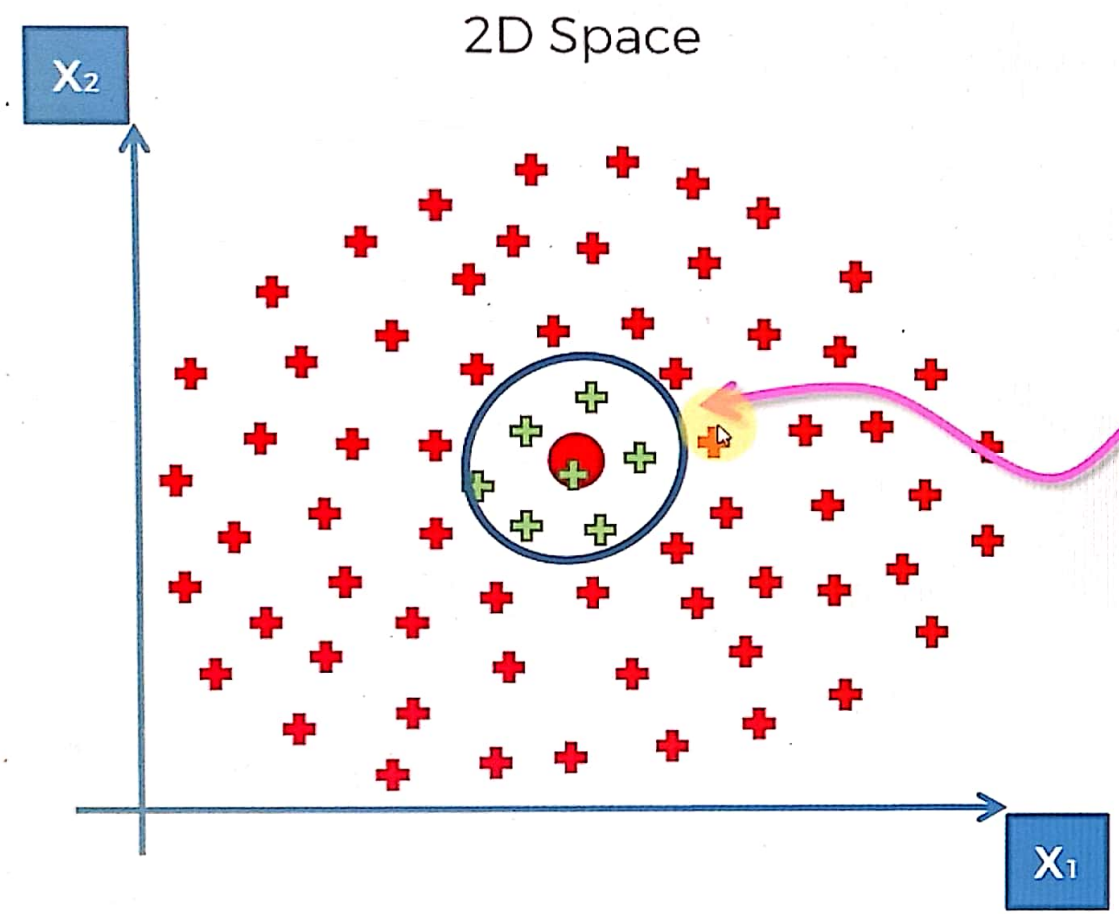


$$K(\vec{x}, \vec{l}^{i}) = e^{-\frac{\|\vec{x}-\vec{l}^{i}\|^2}{2\sigma^2}}$$

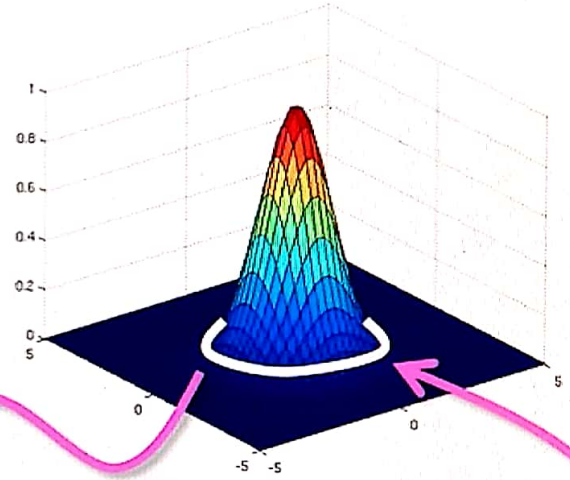
© SuperDataScience

Scanned with CamScanner

The Gaussian RBF Kernel



σ

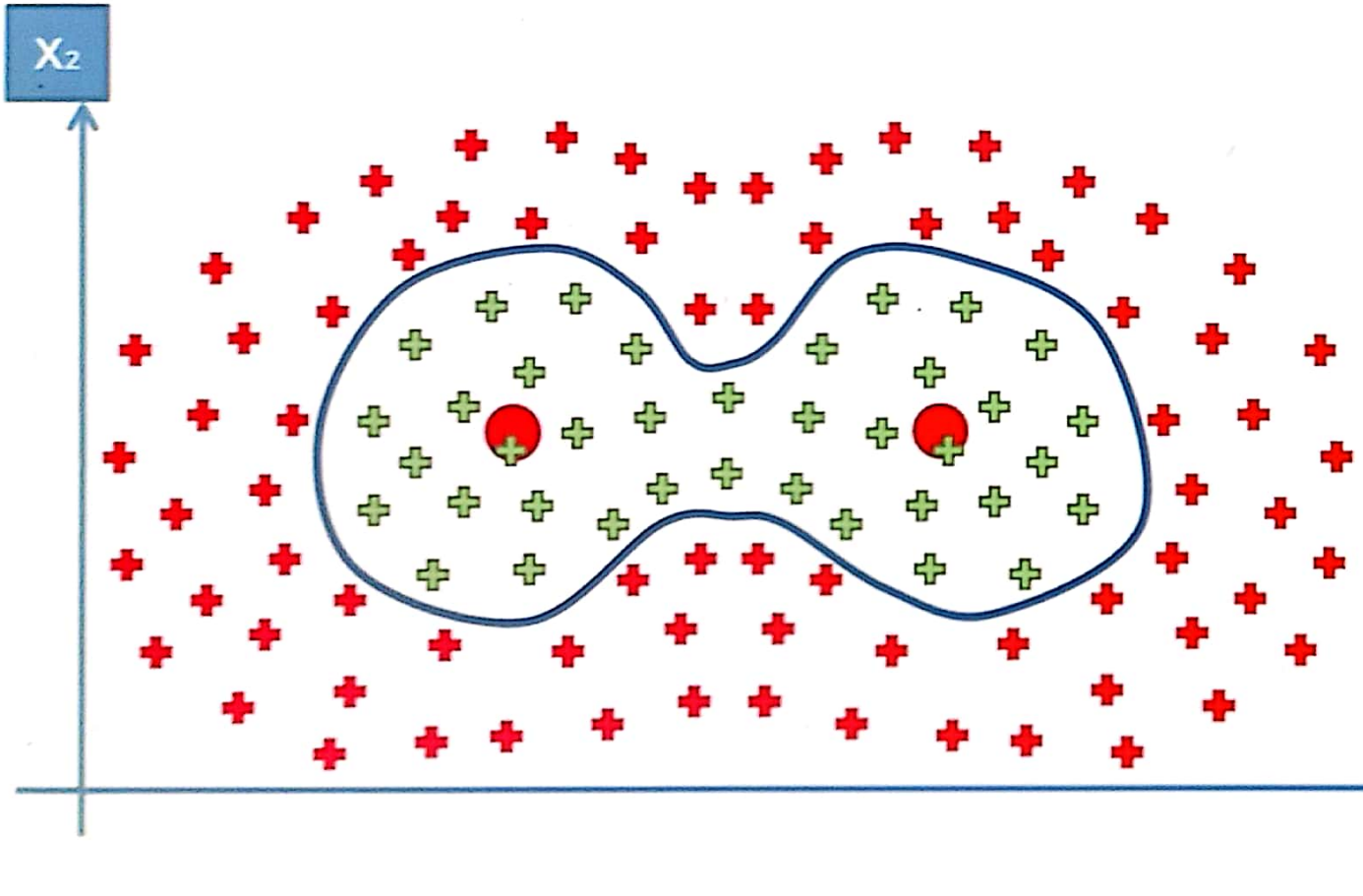


$$K(\vec{x}, \vec{l}^{i}) = e^{-\frac{\|\vec{x}-\vec{l}^{i}\|^2}{2\sigma^2}}$$

© SuperDataScience

Scanned with CamScanner

The Gaussian RBF Kernel



$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2)$$

(Simplified Formula)

Green when:

$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) > 0$$

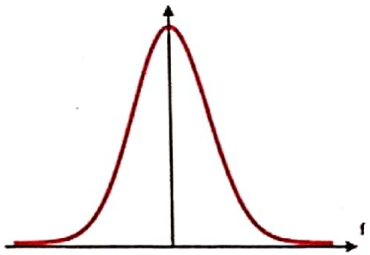
Red when:

$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) = 0$$

© SuperDataScience

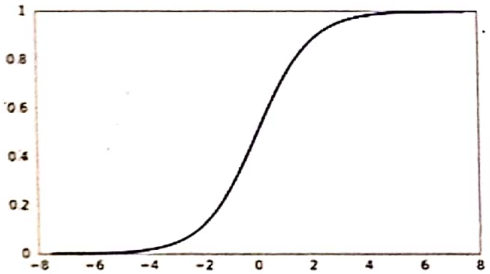
Scanned with CamScanner

Types of Kernel Functions



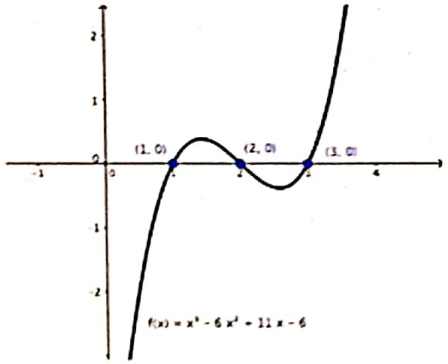
Gaussian RBF Kernel

$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$



Sigmoid Kernel

$$K(X, Y) = \tanh(\gamma \cdot X^T Y + r)$$



Polynomial Kernel

$$K(X, Y) = (\gamma \cdot X^T Y + r)^d, \gamma >$$

© SuperDataScience

Scanned with CamScanner

Kernel Functions — Ma
mlkernels.readthedocs.io/en/latest/kernelfunctions.html

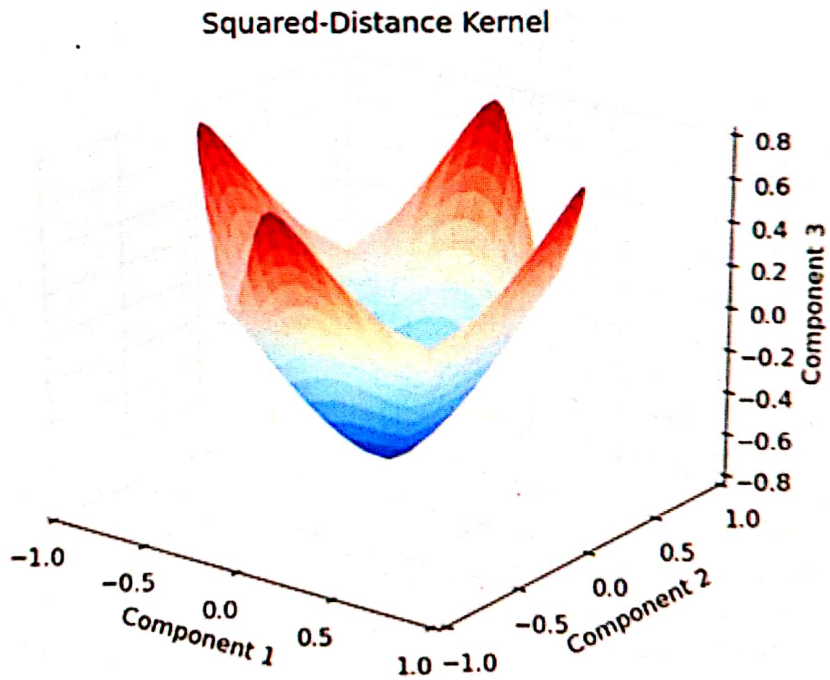
Squared Distance Kernel

```
SquaredDistanceKernel(t=1)
```

The squared distance kernel is a modification of the squared Euclidean distance with an additional shape parameter:

$$\kappa(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} (x_i - y_i)^{2t} \qquad 0 < t \leq 1$$

The squared distance is a **negative definite** stationary kernel [berg]. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a squared distance kernel ($t = 0.5$) are visualized below:



Sine Squared Kernel

```
SineSquaredKernel(p=π,t=1)
```

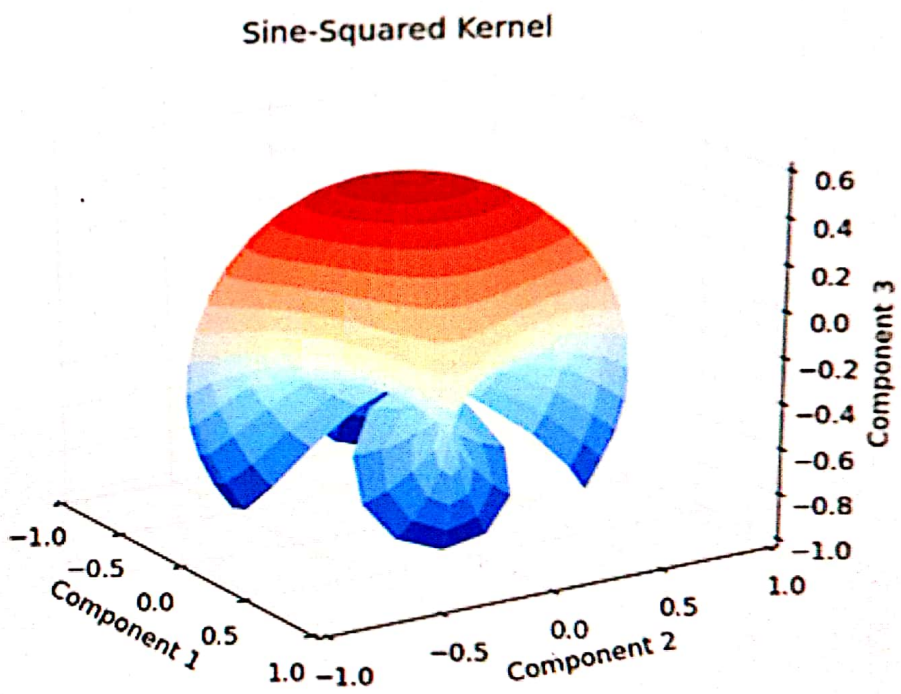
Scanned with CamScanner

```
SineSquaredKernel(p=1,t=1)
```

The sine squared kernel is another **negative definite** stationary kernel [berg]. It can be used to construct the periodic kernel which is useful in situations where data may be periodic:

$$\kappa(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \sin^{2t}(p(x_i - y_i)) \qquad p > 0,\ 0 < t \leq 1$$

The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a sine squared kernel ($p = 1$, $t = 1$) are visualized below:



Over a larger range, the projected surface can be seen to fold in on itself and repeat the shape.

Chi-Squared Kernel

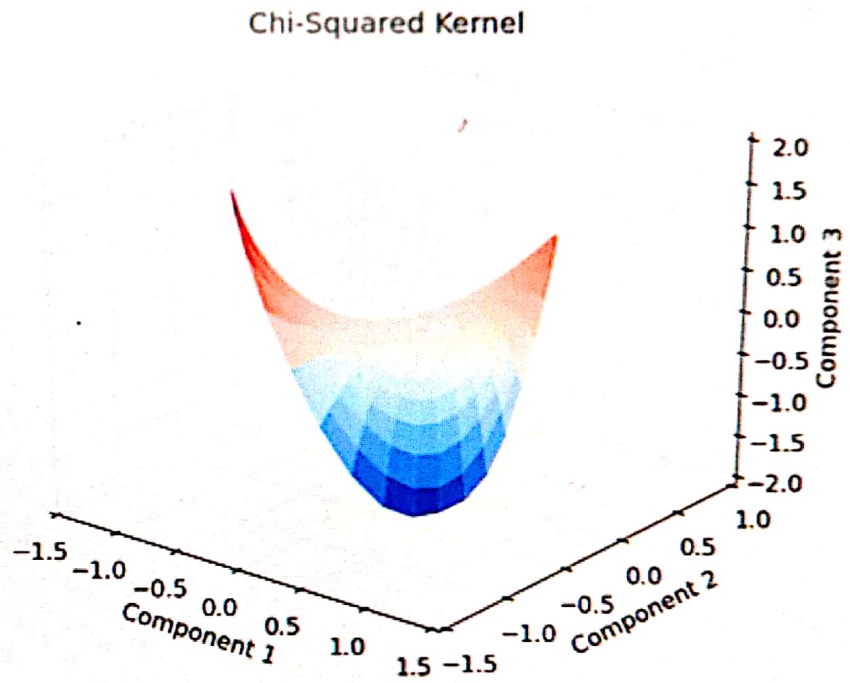
```
ChiSquaredKernel(t=1)
```

The chi-squared kernel is a **negative definite** most often used with bag-of-words models.

Scanned with CamScanner

$$\kappa(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \left(\frac{(x_i - y_i)^2}{x_i + y_i} \right)^t \qquad 0 < t \leq 1,\ x_i > 0\ \forall i,\ y_i > 0\ \forall i$$

The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a chi-squared kernel ($t = 1$) are visualized below:



Gaussian Kernel

```
GaussianKernel(α=1)
RadialBasisKernel(α=1)
SquaredExponentialKernel(α=1)
```

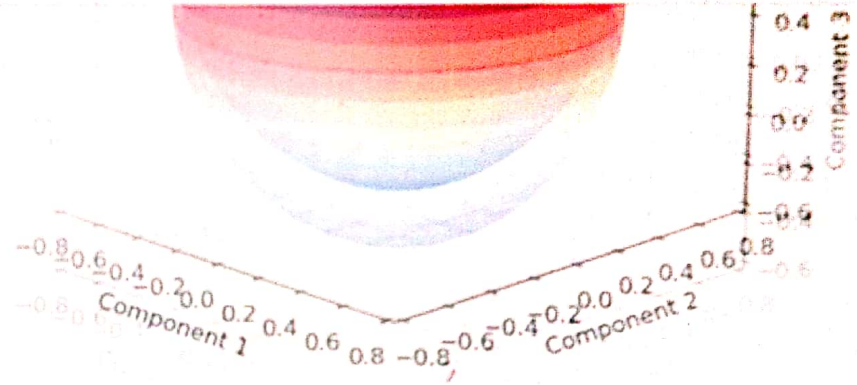
The Gaussian kernel is an isotropic Mercer kernel given by:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\alpha \|\mathbf{x} - \mathbf{y}\|^2\right) \qquad \alpha > 0$$

where α is a scaling parameter of the squared distance. The Gaussian kernel often goes by t
other names: the radial basis kernel and the squared exponential covariance function (Gau

Scanned with CamScanner

Kernel Functions — Ma
mlkernels.readthedocs.io/en/latest/kernelfunctions.html



Laplacian Kernel

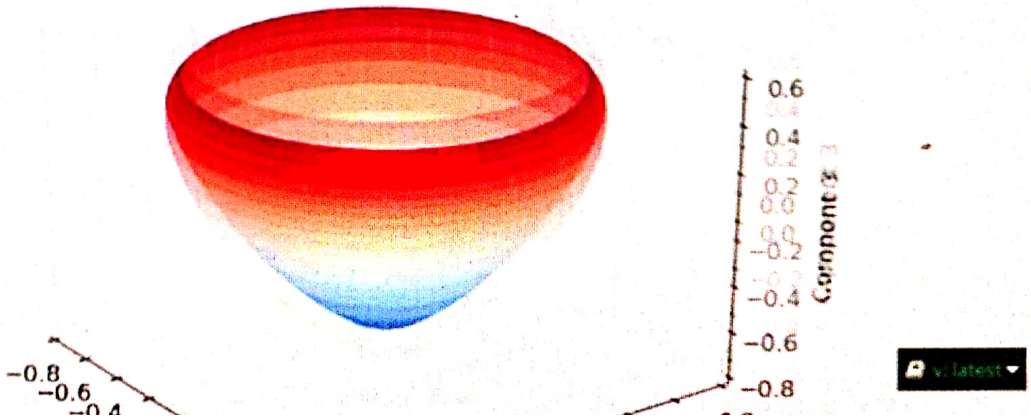
```
LaplacianKernel(α=1)
LaplacianKernel(α=1)
```

The Laplacian kernel is given by:

$$k(\mathbf{x}, \mathbf{y}) = \exp(-\alpha \|\mathbf{x} - \mathbf{y}\|) \qquad \alpha > 0$$

where α is a scaling parameter of the Euclidean distance. The Laplacian kernel is closely related to the Gaussian kernel; the difference is that the Laplacian kernel makes use of the Euclidean distance and the Gaussian kernel uses the squared Euclidean distance. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a Laplacian kernel ($t = 1$) for a similar surface to a Gaussian kernel:

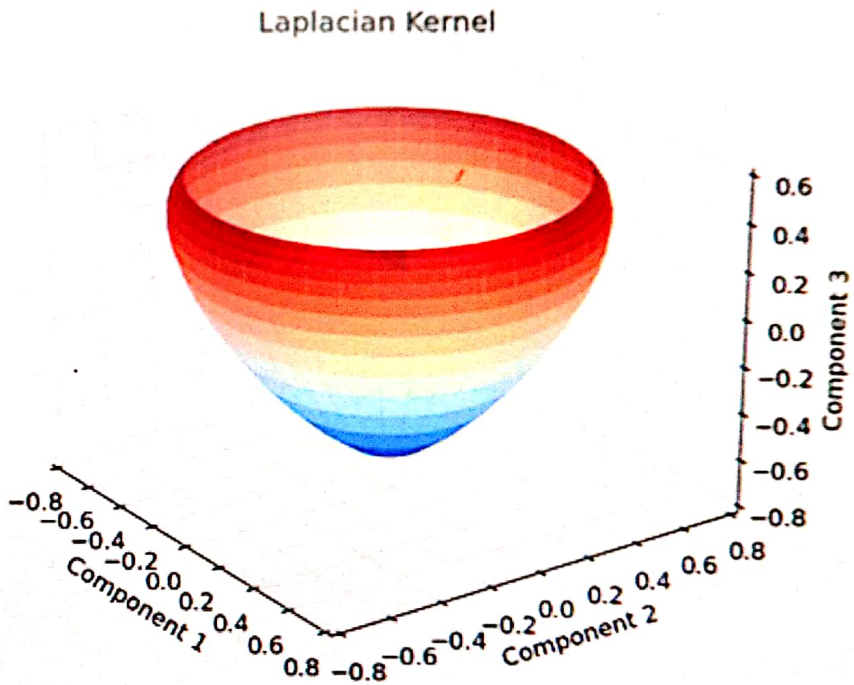
Laplacian Kernel



v: latest
Scanned with CamScanner

Kernel Functions — Ma
mlkernels.readthedocs.io/en/latest/kernelfunctions.html

the Gaussian kernel, the difference is that the Laplacian kernel makes use of the Euclidean distance and the Gaussian kernel uses the squared Euclidean distance. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a Laplacian kernel ($t = 1$) for a similar surface to a Gaussian kernel:



Periodic Kernel

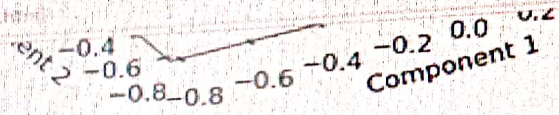
```
PeriodicKernel(α=1,p=π)
```

The periodic kernel is given by:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\alpha \sum_{i=1}^{n} \sin(p(x_i - y_i))^2\right) \qquad p > 0,\ \alpha > 0$$

where $\mathbf{x}$ and $\mathbf{y}$ are n dimensional vectors. The parameters p and α are scaling parameters for the periodicity and the magnitude, respectively. This kernel is useful when data has periodicity to it. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a periodic kernel ($\alpha =$ $p = \pi$) are visualized below:

v: latest
Scanned with CamScanner



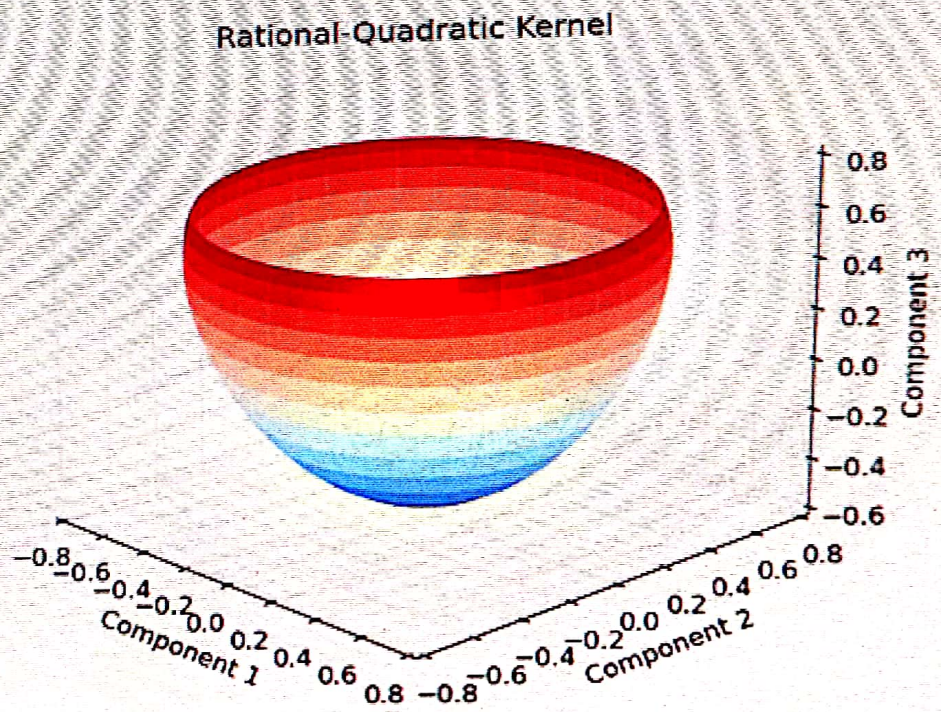
Rational-Quadratic Kernel

```
RationalQuadraticKernel(α=1,β=1,γ=1)
```

The rational-quadratic kernel is given by:

$$k(\mathbf{x},\mathbf{y}) = \left(1 + \alpha\|\mathbf{x},\mathbf{y}\|^2\right)^{-\beta} \qquad \alpha > 0,\ \beta > 0$$

where α is a scaling parameter and β is a shape parameter. This kernel can be seen as an infinite sum of Gaussian kernels. If one sets $\alpha = \alpha_0/\beta$, then taking the limit $\beta \to \infty$ results in the Gaussian kernel with scaling parameter α_0. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a rational-quadratic kernel ($\alpha = 1, \beta = 1, \gamma = 1$) are visualized below:



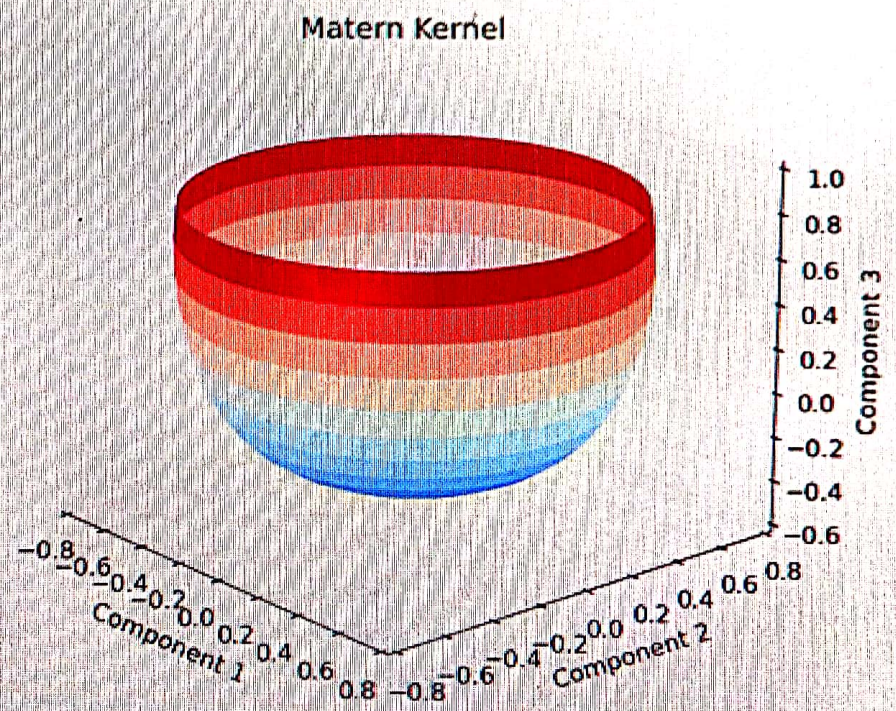
Matern Kernel

v: latest
Scanned with CamScanner

The Matern kernel is a Mercer Kernel [ras] given by:

$$k(\mathbf{x}, \mathbf{y}) = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{2\sqrt{\nu}\|\mathbf{x}-\mathbf{y}\|}{\theta} \right)^{\nu} K_{\nu} \left(\frac{2\sqrt{\nu}\|\mathbf{x}-\mathbf{y}\|}{\theta} \right)$$

where math:*Gamma* is the gamma function, K_ν is the modified Bessel function of the second kind, $\nu > 0$ and $\theta > 0$. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a Matern kernel ($\nu = 1, \theta = 1$) are visualized below:



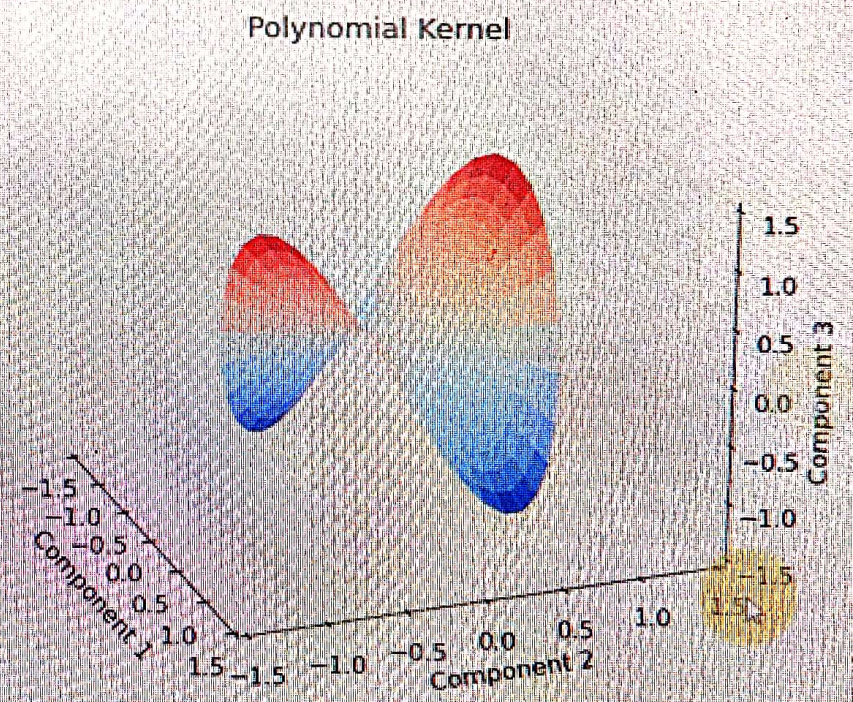
Linear & Polynomial Kernel

```
LinearKernel(a=1,c=1)
PolynomialKernel(a=1,c=1,d=3)
```

The polynomial kernel is a Mercer kernel given by:

Scanned with CamScanner

The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a polynomial kernel ($a = 1$, $c = 1$, $d = 3$) are visualized below:



Sigmoid Kernel

```
SigmoidKernel(a=1,c=1)
```

The sigmoid kernel is given by:

$$k(\mathbf{x}, \mathbf{y}) = \tanh(a\mathbf{x}^T\mathbf{y} + c) \qquad a > 0,\ c \geq 0$$

The sigmoid kernel is a not a true kernel, although it has been used in application. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a sigmoid kernel ($a = 1$, $c = 1$) are visualized below:

Sigmoid Kernel

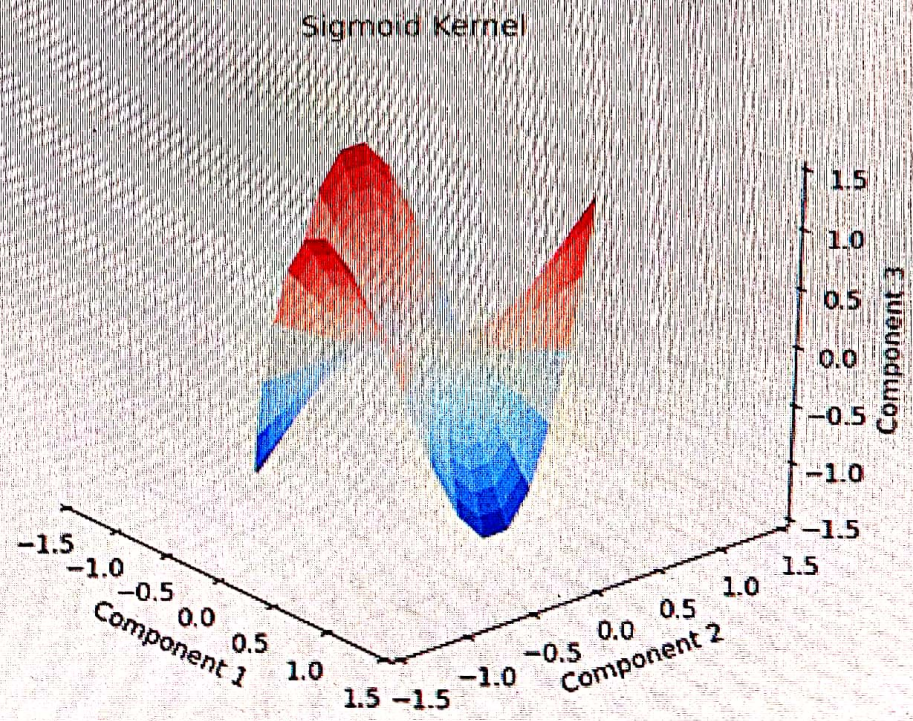
Scanned with CamScanner

Sigmoid Kernel

The sigmoid kernel is given by:

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x}^T \mathbf{y} + c) \qquad \alpha > 0,\ c \geq 0$$

The sigmoid kernel is a non-true kernel, although it has been used in application. The first three components of kernel PCA over an ellipse in $\mathbb{R}^2$ with a sigmoid kernel ($\alpha = 1, c = 1$) are visualized below:



Previous

Next

© Copyright 2015, Tim Thatcher.

Scanned with CamScanner

SVM kernel in PYTHON

→ From sklearn.svm import SVC

→ Create classifier object

→ Set following parameters.

kernel → 'rbf'.
random state

Scanned with CamScanner

Baye's Theorem

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

Naive Bayes classifier Intuition

Step 1 find

$$\cancel{P(Y/X)} \quad P(Y/X) = \frac{P(X/Y) \times P(Y)}{P(X)}$$

Step 2 find

$$P(\bar{Y}/X) = \frac{P(X/\bar{Y}) \times P(\bar{Y})}{P(X)}$$

Step3 compare $P(Y)$ vs $P(\bar{Y})$

if $P(Y) > P(\bar{Y})$ then new data point is Y and vice versa

(see photos for detail.)

Scanned with CamScanner

Bayes Theorem

Mach1: 30 wrenches / hr
Mach2: 20 wrenches / hr

-> P(Mach1) = 30/50 = 0.6
-> P(Mach2) = 20/50 = 0.4

Out of all produced parts:
We can SEE that 1% are defective

-> P(Defect) = 1%

Out of all defective parts:
We can SEE that 50% came from mach1
And 50% came from mach2

-> P(Mach1 | Defect) = 50%
-> P(Mach2 | Defect) = 50%

Question:
What is the probability that a part produced by mach2 is defective = ?

-> P(Defect | Mach2) = ?

© SuperDataScience

Scanned with CamScanner

Bayes Theorem

Mach1: 30 wrenches / hr
Mach2: 20 wrenches / hr
Out of all produced parts:
We can SEE that 1% are defective
Out of all defective parts:
We can SEE that 50% came from mach1
And 50% came from mach2
Question:
What is the probability that a part produced by mach2 is defective = ?

-> P(Mach2) = 20/50 = 0.4
-> P(Defect) = 1%
-> P(Mach2 | Defect) = 50%
-> P(Defect | Mach2) = ?

$$P(\text{Defect} \mid \text{Mach2}) = \frac{0.5 * 0.01}{0.4} = 0.0125$$

© SuperDataScience

Scanned with CamScanner

It's intuitive!

$$P(\text{Defect} \mid \text{Mach2}) = \frac{P(\text{Mach2} \mid \text{Defect}) * P(\text{Defect})}{P(\text{Mach2})} = 1.25\%$$

Let's look at an example:

- 1000 wrenches
- 400 came from Mach2
- 1% have a defect = 10
- of them 50% came from Mach2 = 5
- % defective parts from Mach2 = 5/400 = 1.25%

© SuperDataScienc

Scanned with CamScanner

Meet Now

Move to Categorise Snooze Undo ...

FYP meet
22:00

Confusion Regarding FYP Thesis and FYP Report

1

MUHAMMAD ASGHAR ALI QURESHI
Assalam o Aleikum Respected Sir, Hope this message finds you in g... Mon 24/05/2021 18:16

KS

khurram Shehzad
Mon 24/05/2021 18:34
To: MUHAMMAD ASGHAR ALI QURESHI
Cc: Muhammad Ramzan Malik; Guftaar Ahmad

Dear Student,

Thanks a lot for the email. Please note that the FYP thesis and FYP report are the same. Meanwhile, please note that the complete project report needs to be submitted prior to the internal (departmental) presentations.

Furthermore, It is important to note that the same copy of FYP report would be forwarded to the external examiners for evaluation as well. The FYP thesis/report needs to be submitted as per the schedule already shared on the PMC Facebook page.

Lastly, the book binded FYP report has to be submitted after the external examiner's evaluation.

I hope there must not be any doubt now.

Best regards,

Khurram Shehzad

Get Outlook for Android

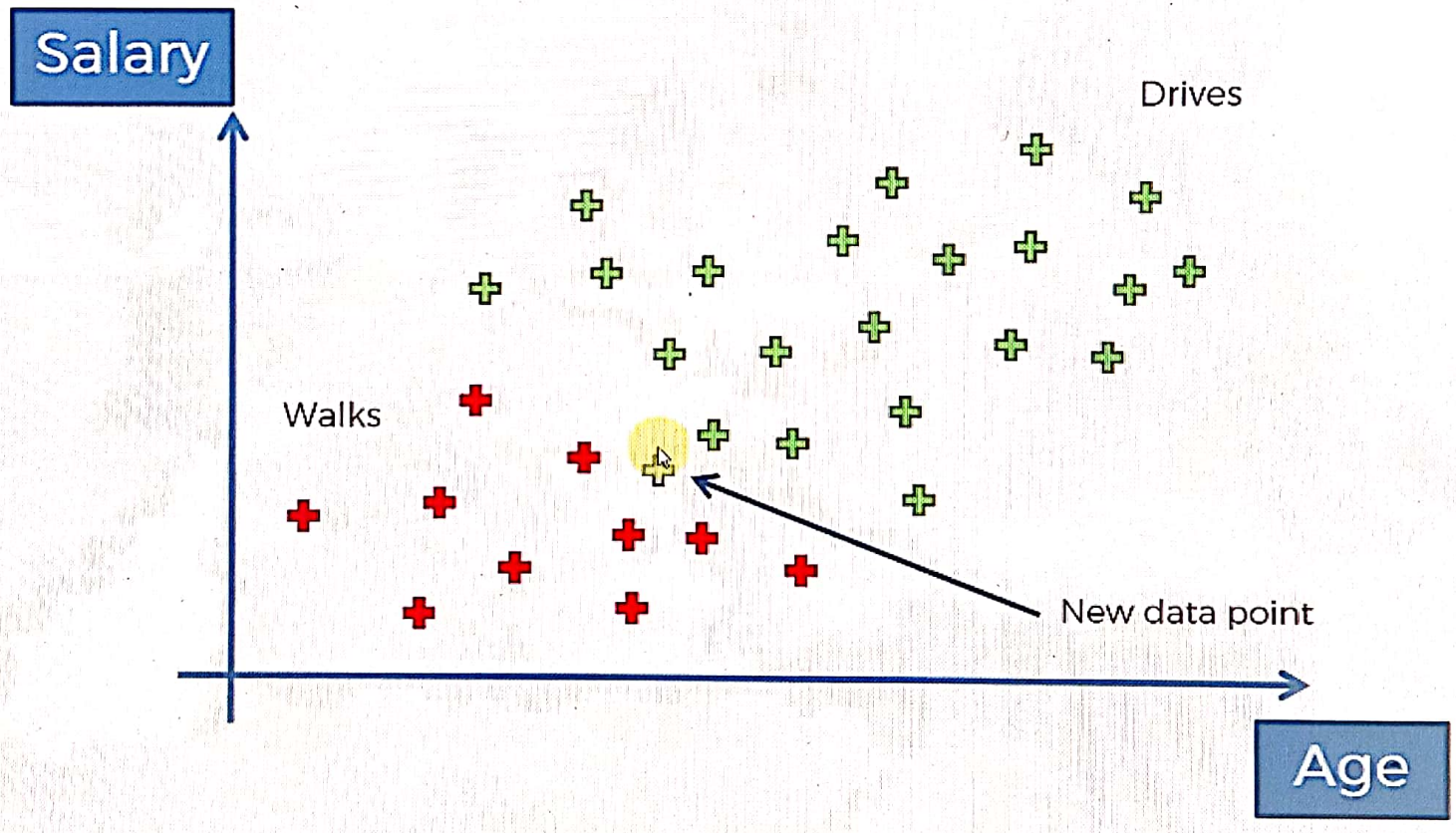
...

MUHAMMAD ASGHAR ALI QURESHI
Mon 24/05/2021 18:44

Activate Windows

Scanned with CamScanner

Naïve Bayes



© SuperDataScience

Scanned with CamScanner

Step 1

#4 Posterior Probability

#3 Likelihood

#1 Prior Probability

$$P(Walks|X) = \frac{P(X|Walks) * P(Walks)}{P(X)}$$

#2 Marginal Likelihood

© SuperDataScience

Scanned with CamScanner

Step 2

Posterior Probability

Likelihood

Prior Probability

$$P(Drives|X) = \frac{P(X|Drives) * P(Drives)}{P(X)}$$

Marginal Likelihood

© SuperDataScience

Scanned with CamScanner

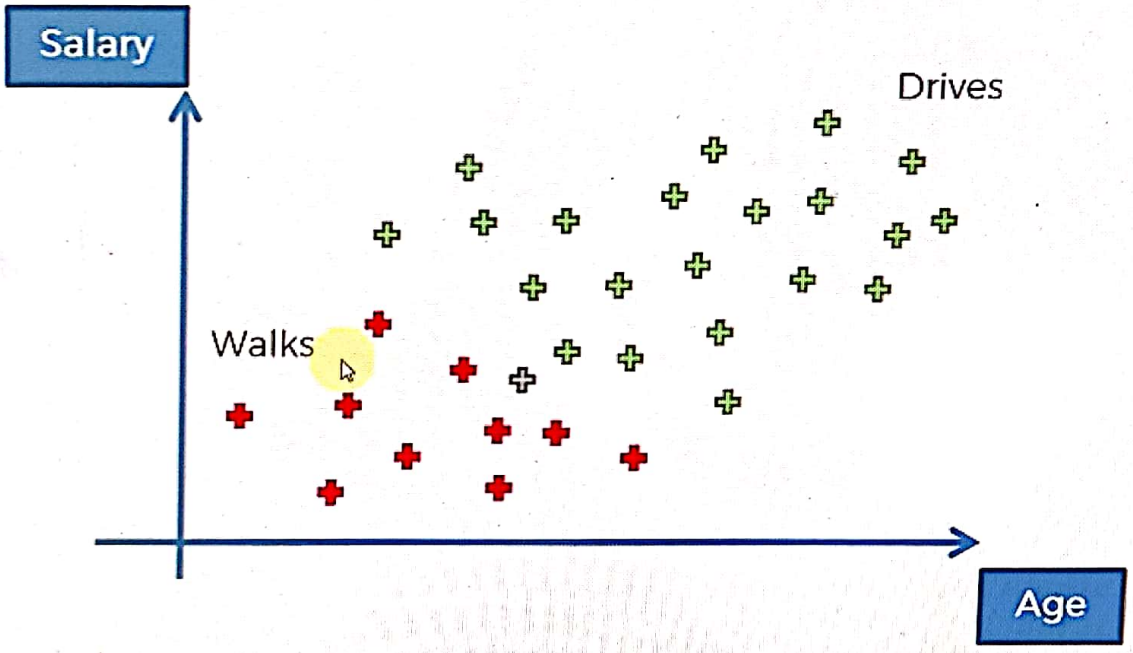
Step 3

$$P(Walks|X) \; v.s. \; P(Drives|X)$$

© SuperDataScience

Scanned with CamScanner

Naïve Bayes: Step 1



#1. P(Walks)

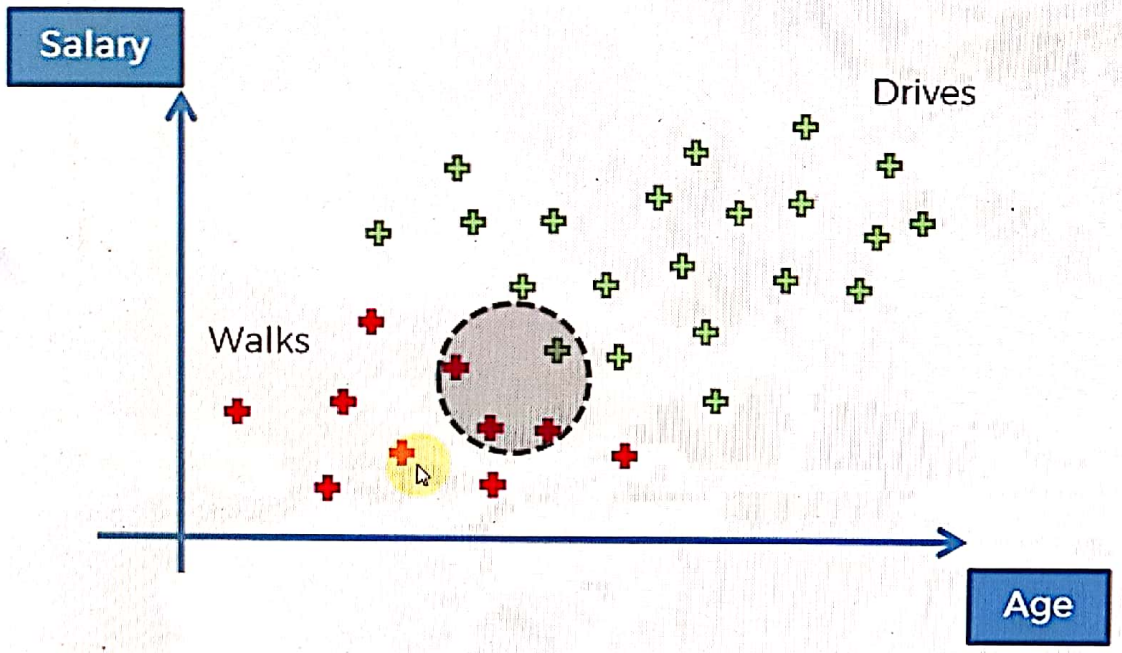
$$P(Walks) = \frac{Number\ of\ Walkers}{Total\ Observations}$$

$$P(Walks) = \frac{10}{30}$$

© SuperDataScience

Scanned with CamScanner

Naïve Bayes: Step 1



#2. P(X)

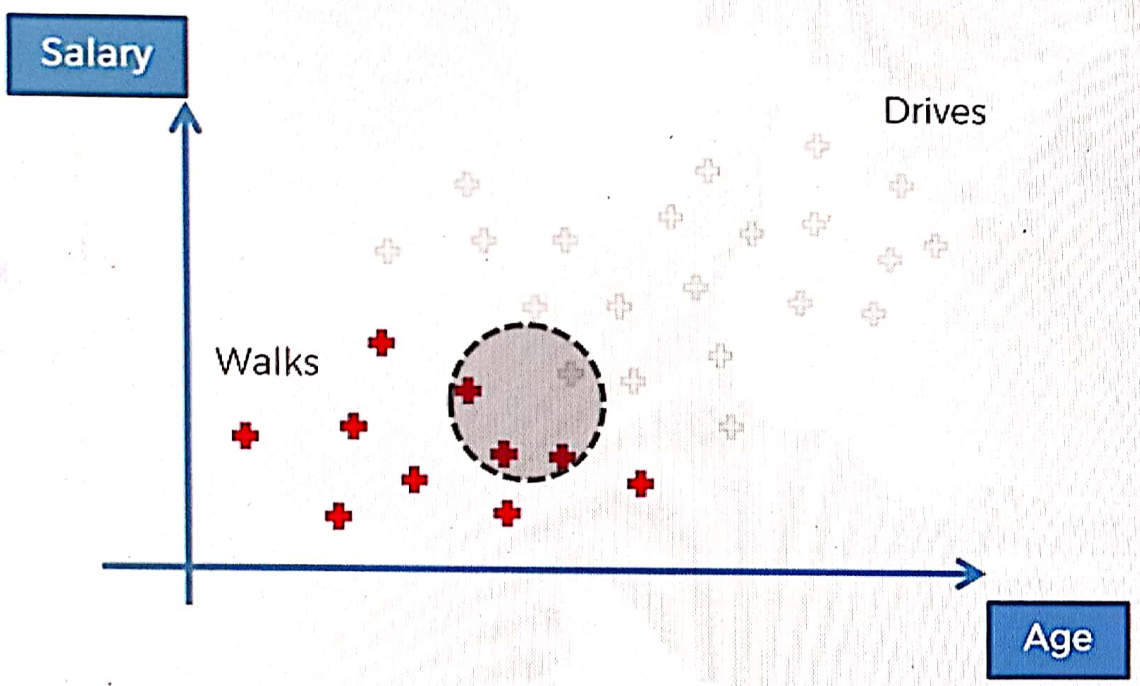
$$P(X) = \frac{Number\ of\ Similar\ Observations}{Total\ Observations}$$

$$P(X) = \frac{4}{30}$$

© SuperDataScience

Scanned with CamScanner

Naïve Bayes: Step 1



#3. P(X|Walks)

$$P(X|Walks) = \frac{\text{Number of Similar Observations Among those who Walk}}{\text{Total number of Walkers}}$$

$$P(X|Walks) = \frac{3}{10}$$

© SuperDataScience

Scanned with CamScanner

Naïve Bayes: Step 1

#4 Posterior Probability

#3 Likelihood

#1 Prior Probability

$$P(Walks|X) = \frac{\frac{3}{10} * \frac{10}{30}}{\frac{4}{30}} = 0.75$$

#2 Marginal Likelihood

© SuperDataScience

Scanned with CamScanner

#4 Posterior Probability

#3 Likelihood

#1 Prior Probability

$$P(Drives|X) = \frac{P(X|Drives) * P(Drives)}{P(X)}$$

#2 Marginal Likelihood

© SuperDataScience

Scanned with CamScanner

Naïve Bayes: Step 2

#4 Posterior Probability

#3 Likelihood

#1 Prior Probability

$$P(Drives|X) = \frac{\frac{1}{20} * \frac{20}{30}}{\frac{4}{30}} = 0.25$$

#2 Marginal Likelihood

© SuperDataScience

Scanned with CamScanner

Step 3

$$P(Walks|X) \; v.s. \; P(Drives|X)$$

© SuperDataScience

Scanned with CamScanner

Step 3

$$0.75 \; v.s. \; 0.25$$

© SuperDataScience

Scanned with CamScanner

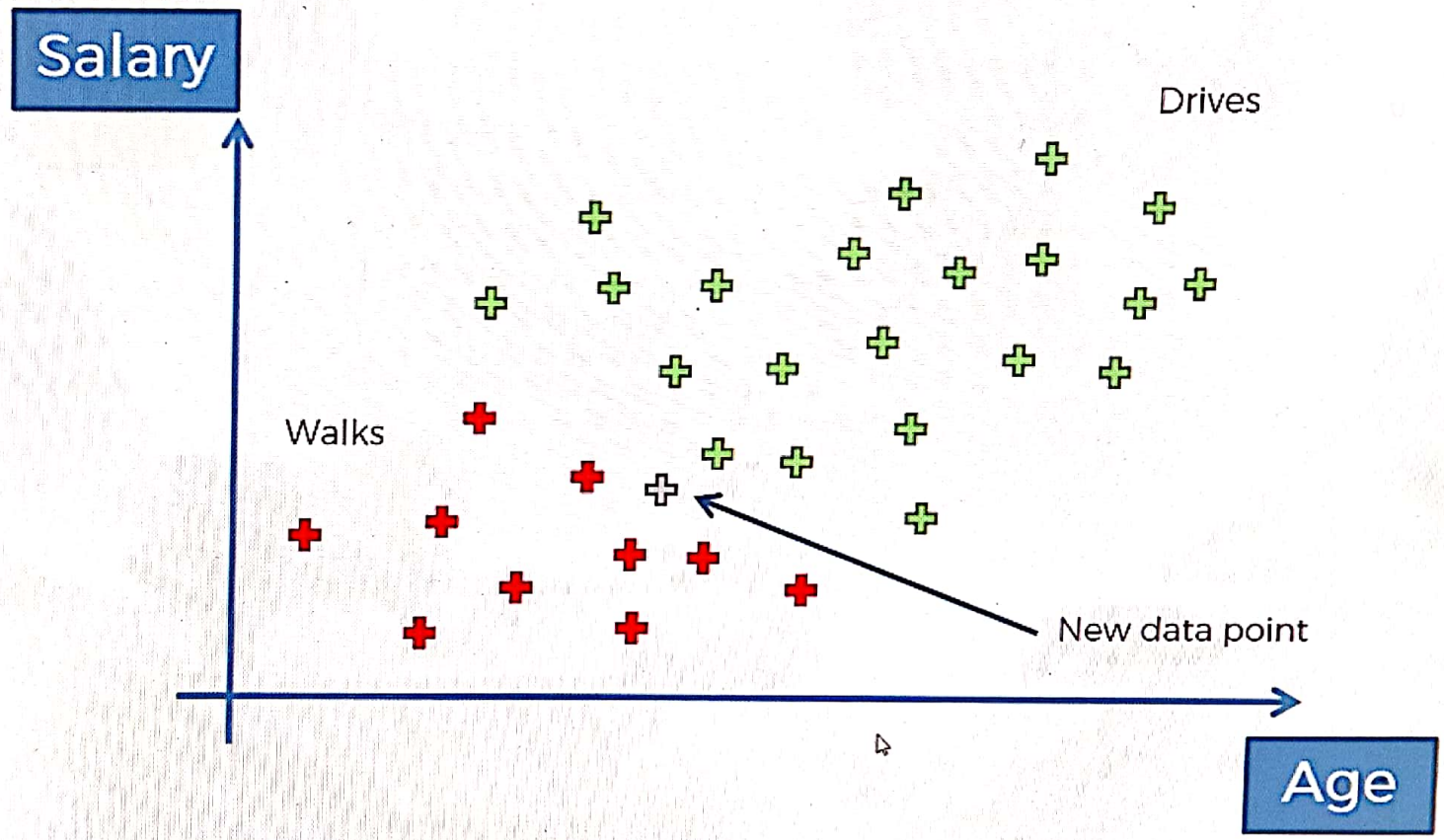
Step 3

$$P(Walks|X) > P(Drives|X)$$

© SuperDataScience

Scanned with CamScanner

Naïve Bayes



© SuperDataScience

Scanned with CamScanner

Naive Bayes's PYTHON

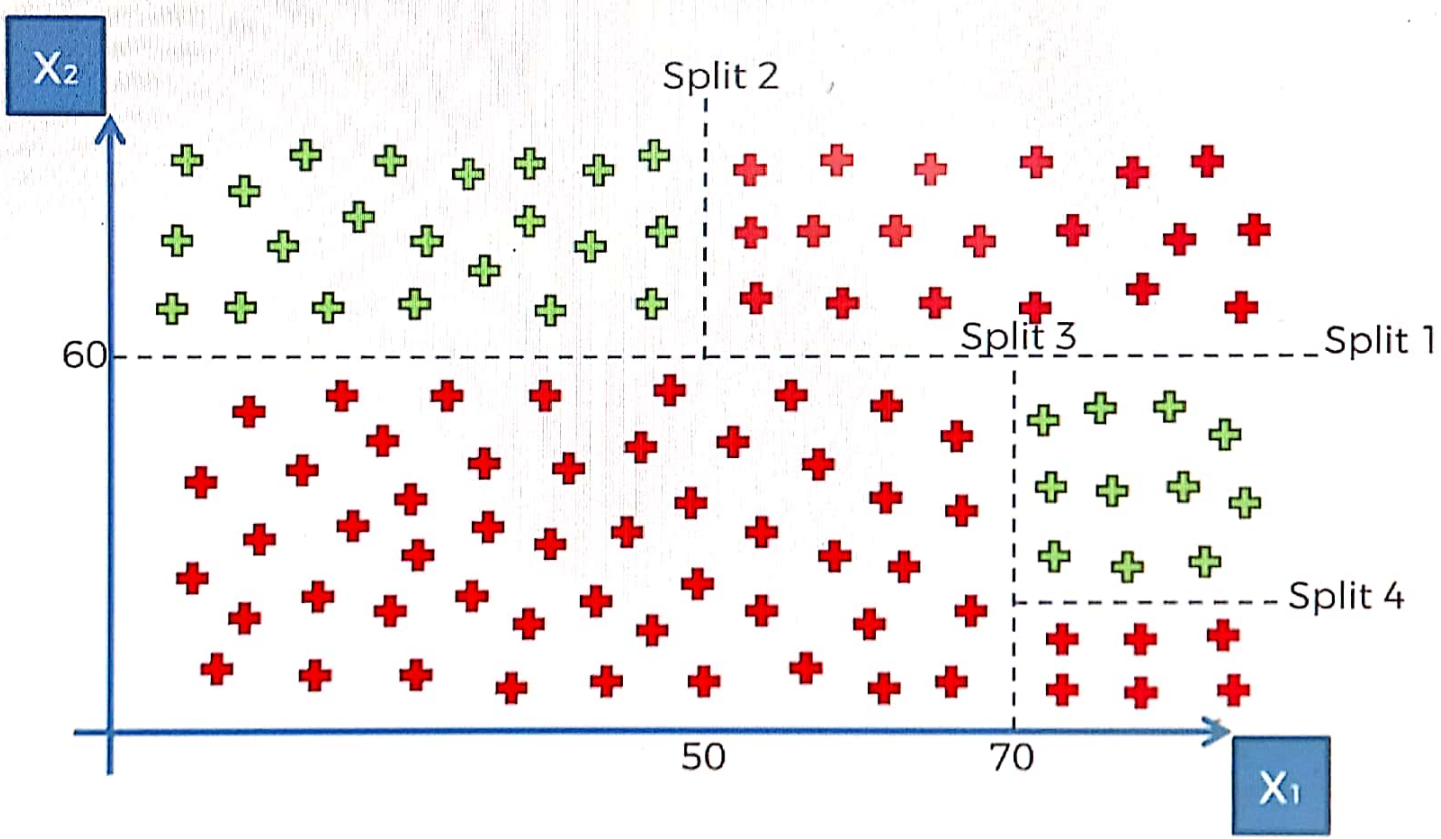
→ From sklearn. naive_bayes import GaussianNB.

→ make object classifier. (no argument required)

→ fit classifier.

Scanned with CamScanner

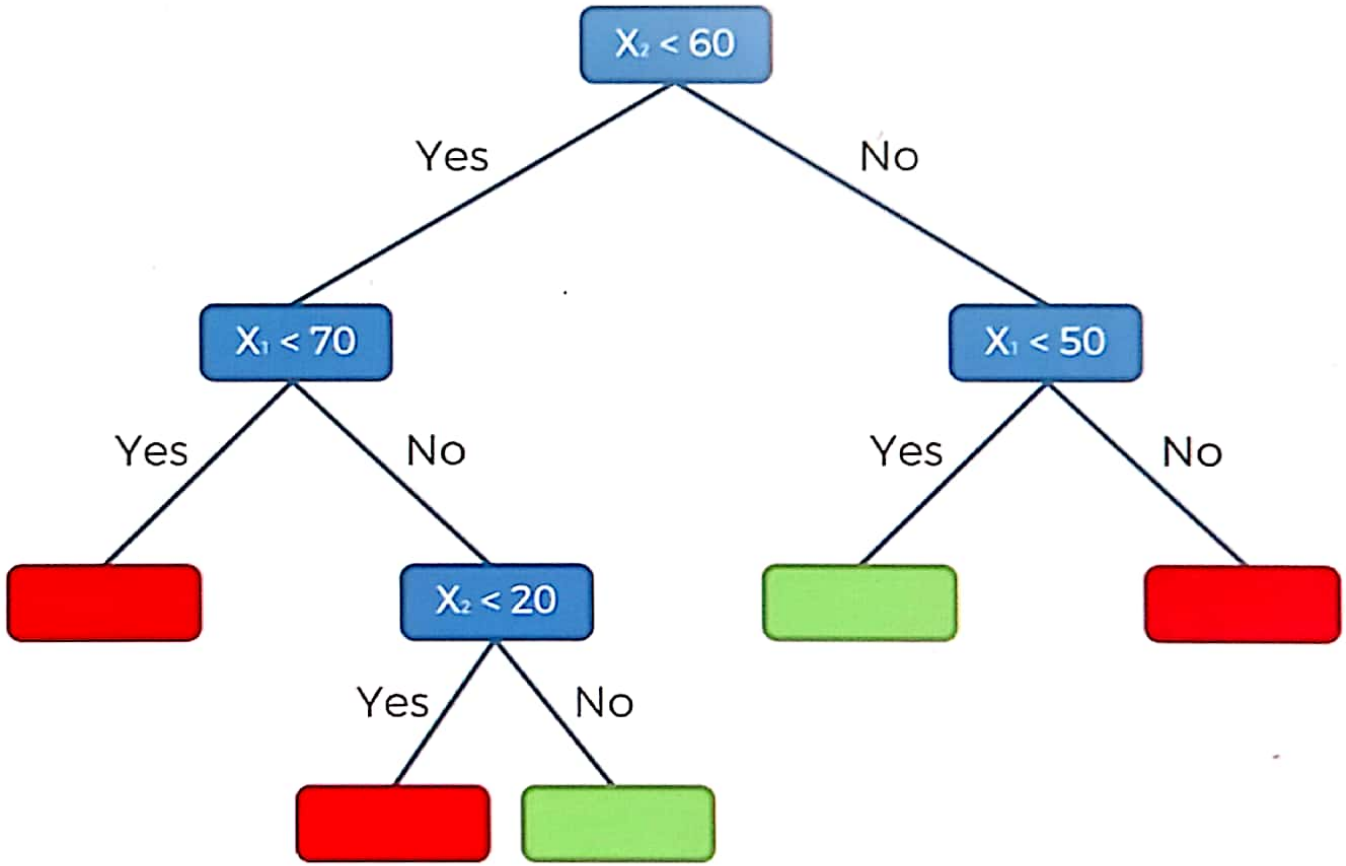
Decision Tree Intuition



© SuperDataScience

Scanned with CamScanner

Split 4



© SuperDataScience

Scanned with CamScanner

Decision Trees

- Old Method
- Reborn with upgrades
- Random Forest
- Gradient Boosting
- etc.

© SuperDataScience

Scanned with CamScanner

Decision Trees classification

PYTHON CODE

(1) Dosen't Require feature scalling

(because algo don't use Eculedion distance).

(2) From sklearn.tree import DecisionTreeClassifier.

classifier = DecisionTreeClassifier()

parameters: criterion = 'entropy'

random_state = _

Random Forest Intuition

STEP 1: Pick at random K data points from the Training set.

STEP 2: Build the Decision Tree associated to these K data points.

STEP 3: Choose the number Ntree of trees you want to build and repeat STEPS 1 & 2

STEP 4: For a new data point, make each one of your Ntree trees predict the category to which the data points belongs, and assign the new data point to the category that wins the majority vote.

© SuperDataScience

Scanned with CamScanner

Random Forest Intuition

© SuperDataScience

Scanned with CamScanner

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation

Abstract

We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. Our large and highly varied training dataset allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.

The system runs at 200 frames per second on consumer hardware. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.

1. Introduction

Robust interactive human body tracking has applications including gaming, human-computer interaction, security, telepresence, and even health-care. The task has recently been greatly simplified by the introduction of real-time depth cameras [16, 19, 44, 37, 28, 13]. However, even

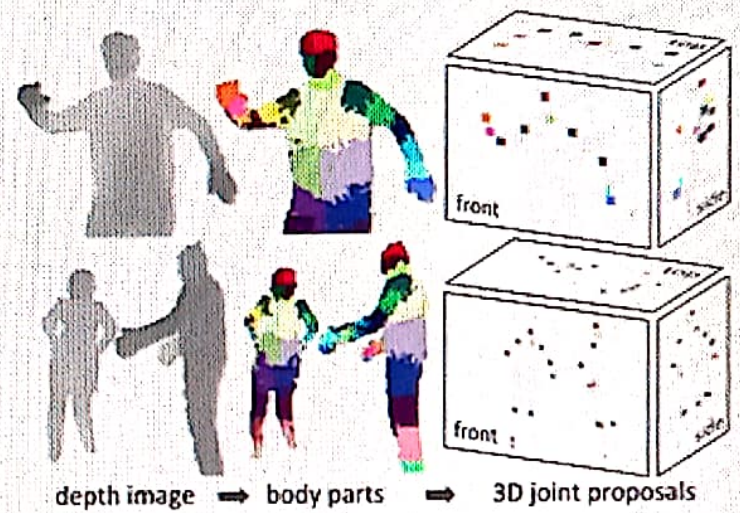


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

joints of interest. Reprojecting the inferred parts into world space, we localize spatial modes of each part distribution and thus generate (possibly several) confidence-weighted proposals for the 3D locations of each skeletal joint.

We treat the segmentation into body parts as a per-pixel classification task (no pairwise terms or CRF have proved necessary). Evaluating each pixel separately avoids a com-

Scanned with CamScanner

Random Forest :

(PYTHON CODE)

→ ~~No~~ Does N need for feature scalling

→ ~~From sklearn. ensemble import RandomForestClas.~~

→ from sklearn. ensemble import Random Forest Classifier.

Parameters →

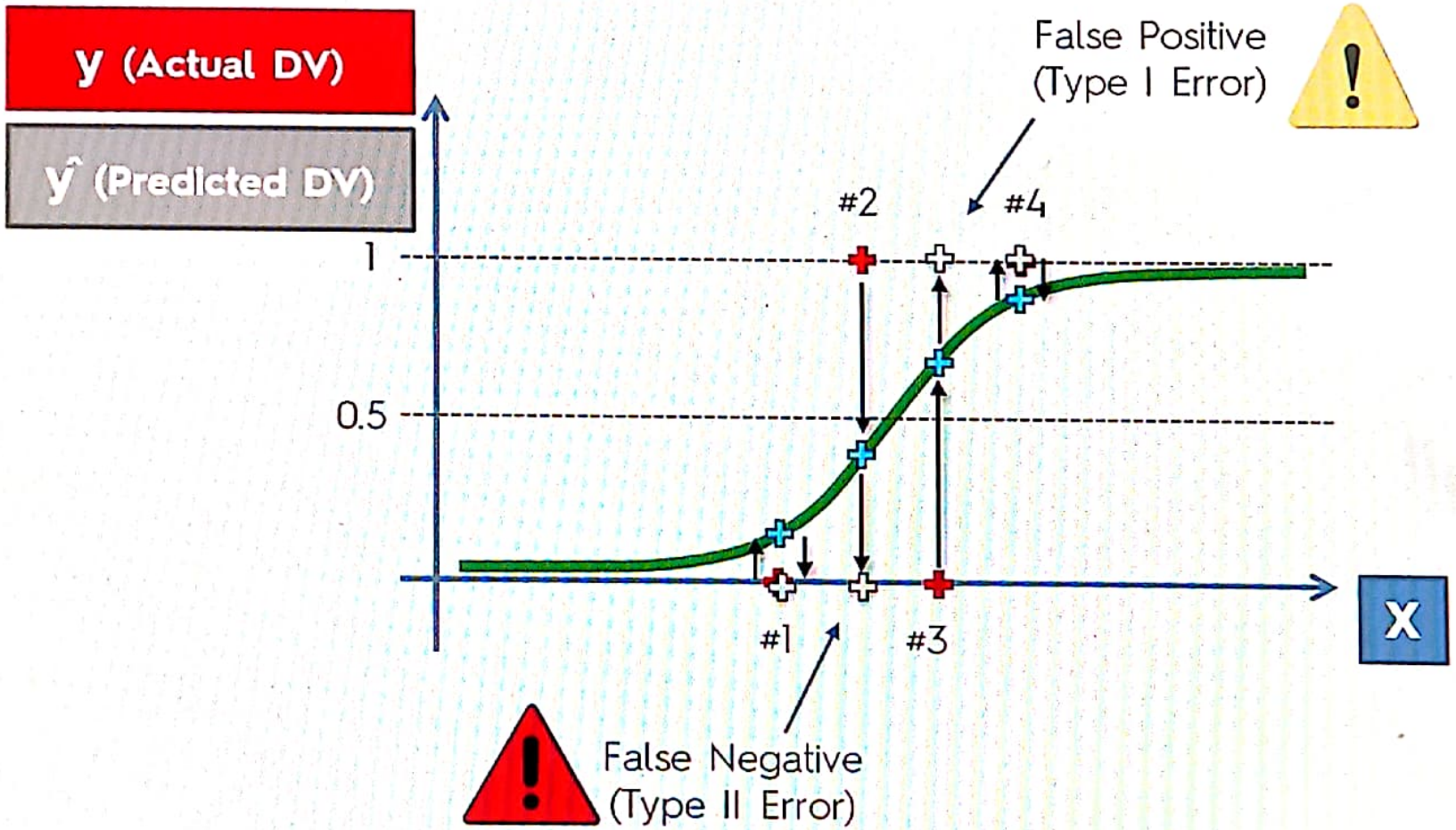
→ n_estimator = _ → no of trees.

→ ~~ct~~ criterion = entropy

→ random_state = _

Scanned with CamScanner

False Positives & Negatives



Fin.

© Kirill Eremenko

Scanned with CamScanner

Confusion Matrix

		$\hat{y}$ (Predicted DV)	
		0	1
y (Actual DV)	0	35	5
	1	10	50

False Positive
(Type I Error)

False Negative
(Type II Error)

Calculate two rates

1. Accuracy Rate = Correct / Total
AR = 85/100 = 85%

2. Error Rate = Wrong / Total
ER = 15/100 = 15%

© Kirill Eremenko

Scanned with CamScanner

Accuracy Paradox

		$\hat{y}$ (Predicted DV)	
		0	1
y (Actual DV)	0	9,700	150
	1	50	100

Scenario 1:

Accuracy Rate = Correct / Total
AR = 9,800/10,000 = 98%

© Kirill Eremenko

Scanned with CamScanner

Accuracy Paradox

		$\hat{y}$ (Predicted DV)	
		0	1
y (Actual DV)	0	9,850	0
	1	150	0

Scenario 1:

Accuracy Rate = Correct / Total
AR = 9,800/10,000 = 98%

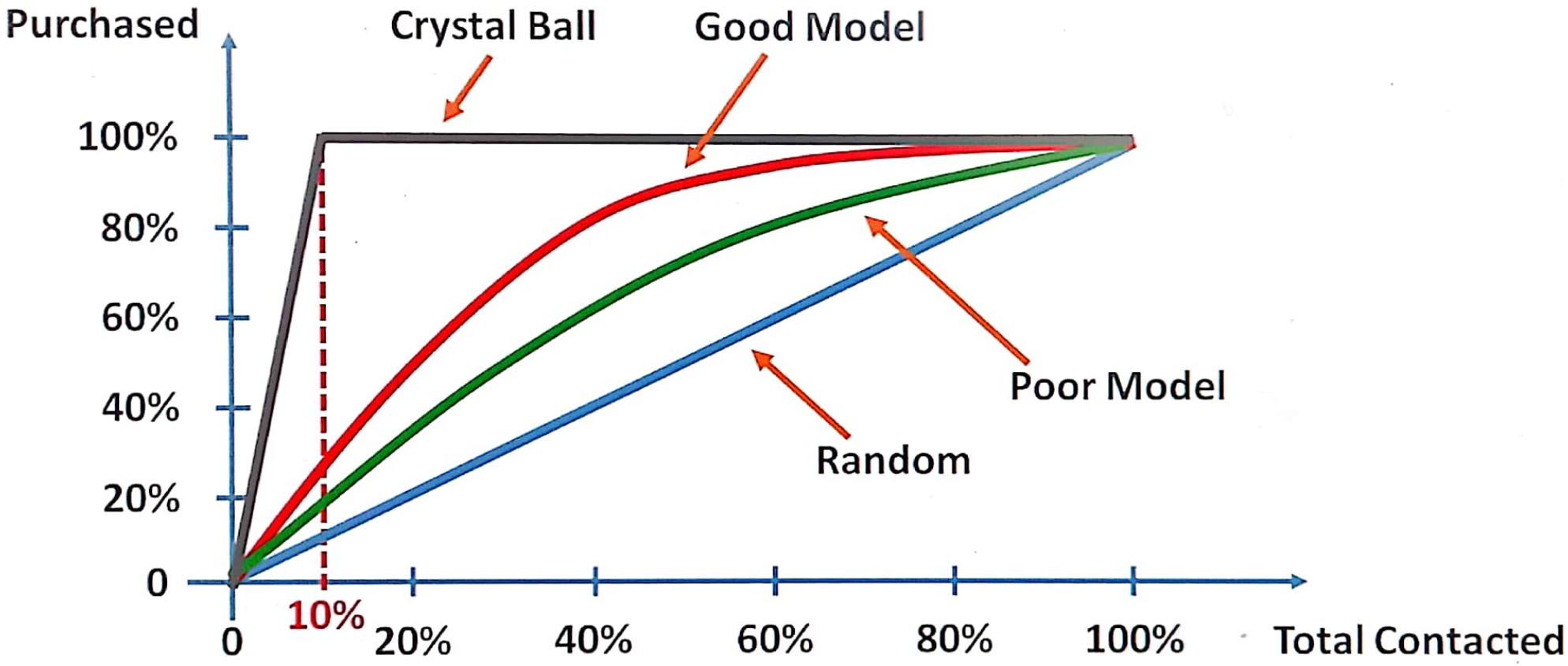
Scenario 2:

Accuracy Rate = Correct / Total
AR = 9,850/10,000 = 98.5%

© Kirill Eremenko

Scanned with CamScanner

CAP



© Kirill Eremenko

Scanned with CamScanner

CAP

Note:

CAP = Cumulative Accuracy Profile

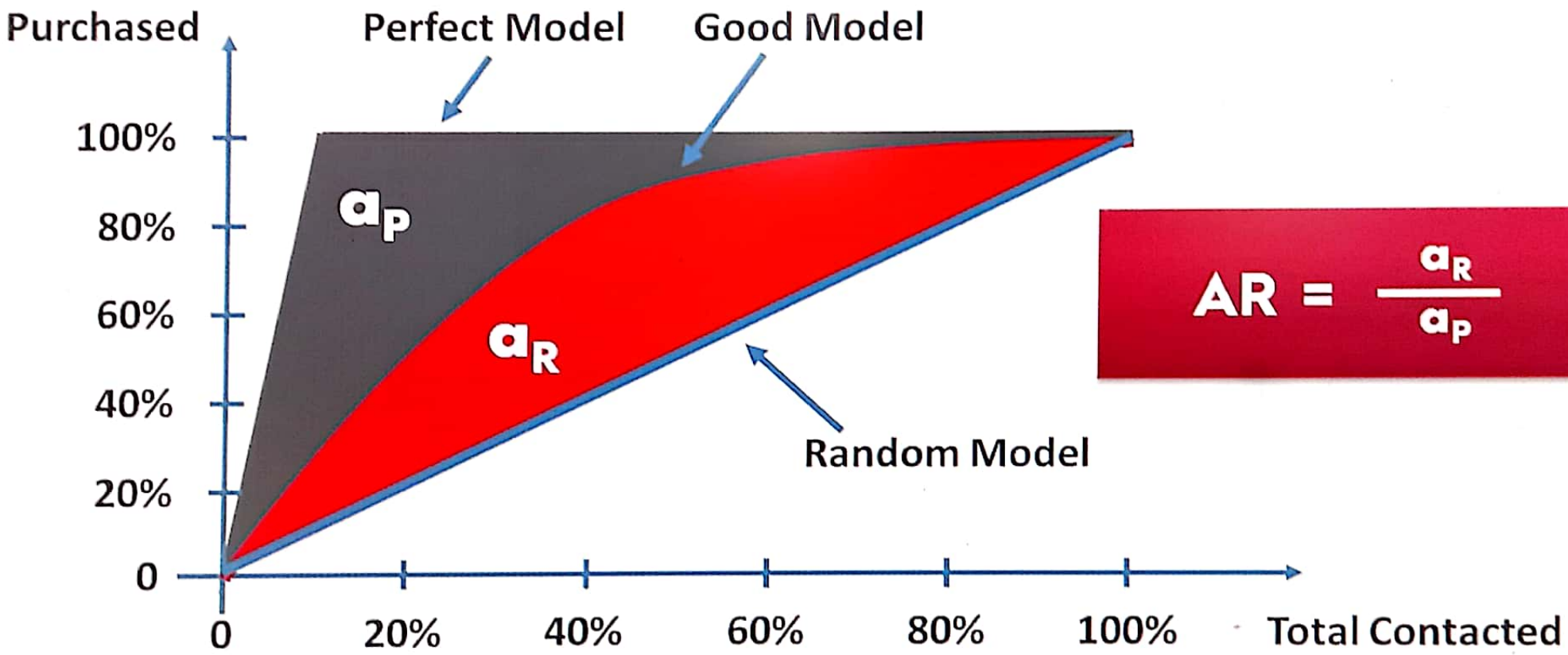
$\neq$

ROC = Receiver Operating Characteristic

© Kirill Eremenko

Scanned with CamScanner

CAP Analysis

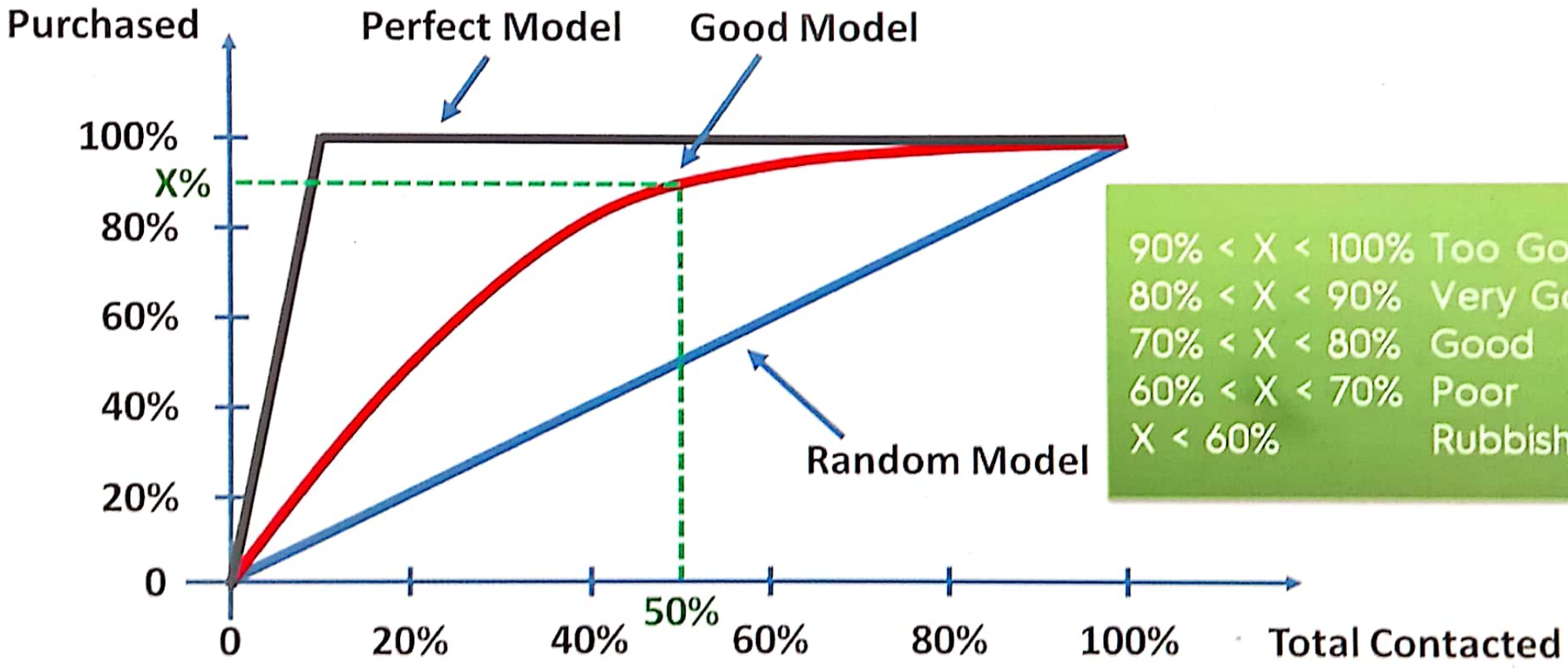


$$AR = \frac{a_R}{a_P}$$

© Kirill Eremenko

Scanned with CamScanner

CAP Analysis



90% < X < 100%	Too Good
80% < X < 90%	Very Good
70% < X < 80%	Good
60% < X < 70%	Poor
X < 60%	Rubbish

© Kirill Eremenko

Scanned with CamScanner