Topics:

- SVM - dual (recap) ✓
- Kernel-Trick → polynomial kernel
- RBF-kernel vs KNN
- SV-Regression
- SVMs in practice
- Naive-Bayes (this & next class)

SVM - Dual

$$\max_{\alpha_i} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \underbrace{x_i^T x_j}_{\text{Similarity}} \rightarrow K(x_i, x_j)$$

$$\text{s.t} \quad 0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

1. $x_i \rightarrow \alpha_i$
2. $x_i^T x_j$ form
3. $f(x_q) = \hat{y}_q$ ($K(x_i, x_q)$)

$$\sum_{i=1}^{n} \alpha_i y_i x_i^T x_q + b$$

4. $\alpha_i > 0$ for SVs only

$\alpha_i = 0$ for non SVs

Primal - Dual (proof)

Recap

$K(x_1, x_2) = (1 + x_1^T x_2)^2$: Quadratic kernel, $x_1, x_2 \in \mathbb{R}^2$

$= x_1'^T \cdot x_2'$

$=$

$x_1' = [1,\ x_{11}^2,\ x_{12}^2,\ \sqrt{2}x_{11},\ \sqrt{2}x_{12},\ \sqrt{2}x_{11}x_{12}]^T$

$x_2' = [1,\ x_{21}^2,\ x_{22}^2,\ \sqrt{2}x_{21},\ \sqrt{2}x_{22},\ \sqrt{2}x_{21}x_{22}]^T$

6-dim

Q

Task: Vizualization { PCA → d → d'

2 or 3

d' < d

↳ Vizualize

preserving variance

Task: Classification { SVM → d —Kernel→ d'

d' > d

↓

it is easier to find a $\Pi^{d'}_{sep}$

↓

logistic regression

Lagrange - Mul → concept in optimzn

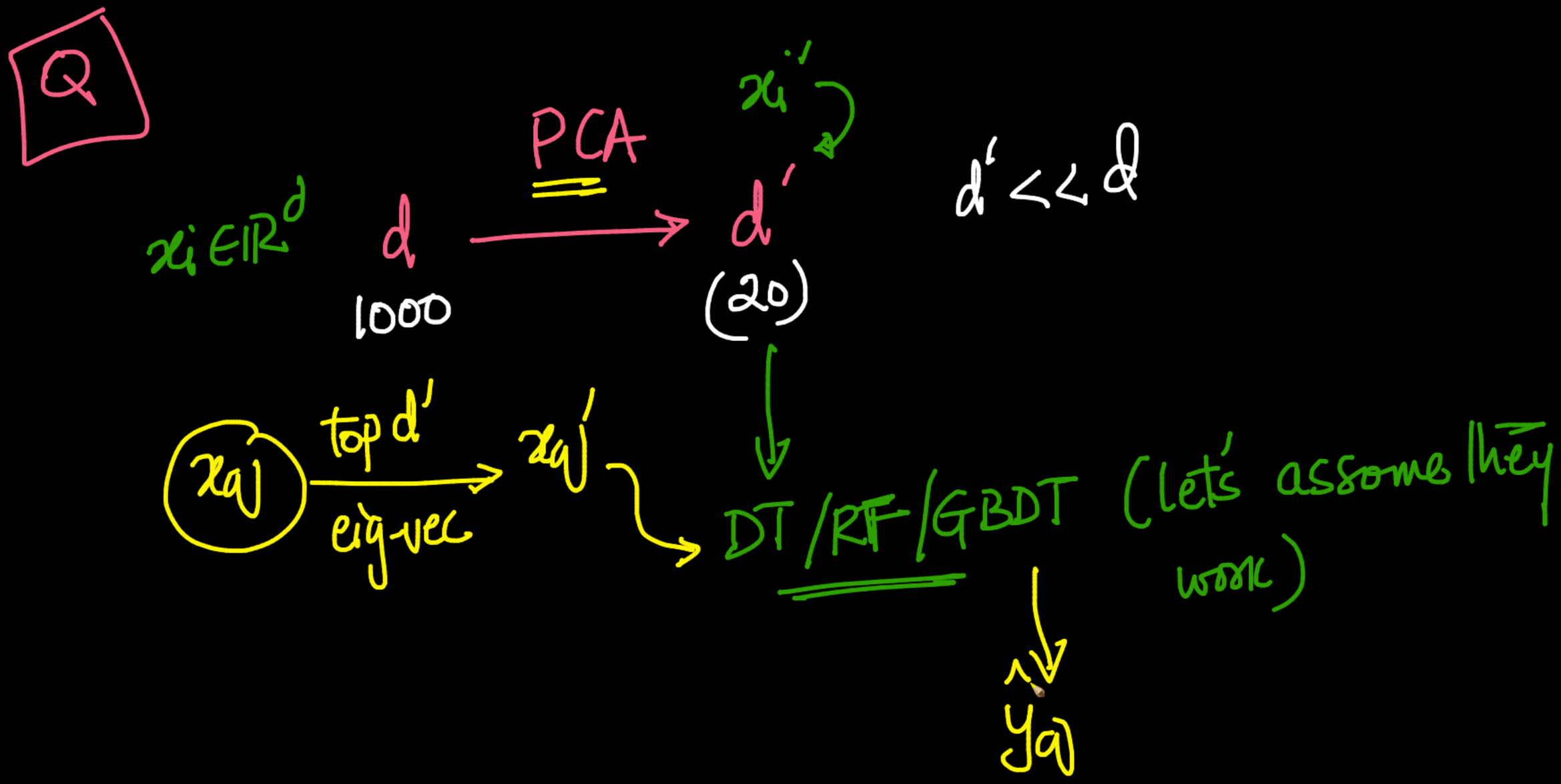
PCA

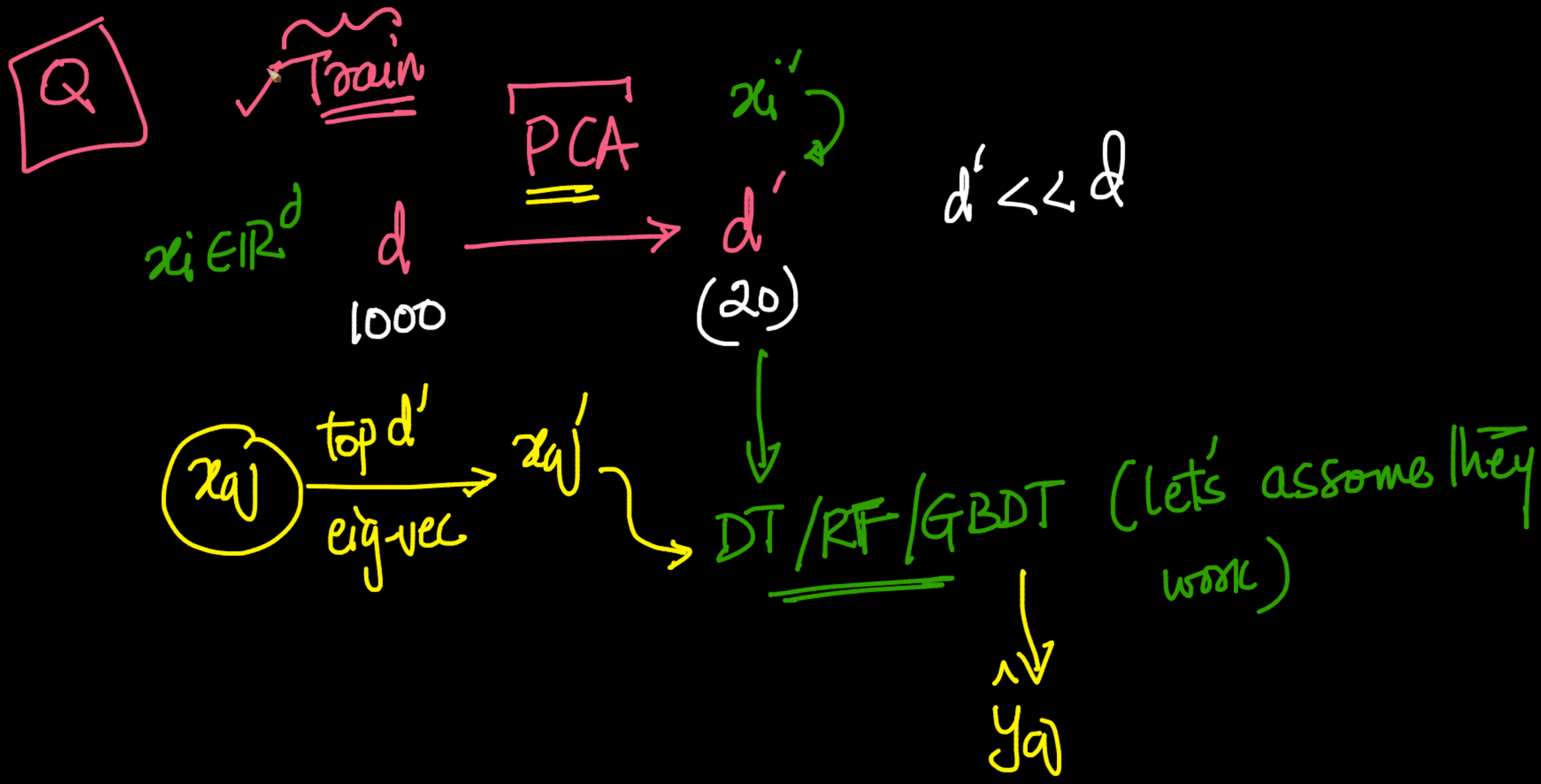
Lr. reg

$$\min_{x} \; f(x) + \lambda_1 \square + \lambda_2 \square$$

$$s.t \quad g(x) = 0$$

$$h(x) > 0$$


Q
PCA
$x_i \in \mathbb{R}^d$
d
1000
d'
(20)
x_i'
$d' << d$
x_{qj}
top d' eig-vec
x_{qj}'
DT / RF / GBDT (let's assume they work)
$\hat{y}_q$


Q
Train
PCA
$x_i \in \mathbb{R}^d$
d
1000
d'
(20)
x_i'
$d' << d$
x_{qj}
top d' eig-vec
x_{qj}'
DT / RF / GBDT (Let's assume they work)
y_q

$D_{Train} \longrightarrow PCA\ (d \to d')$

top d' eig-vec

$D_{Test}\ (x_i) \longrightarrow$ top d' - eigvec $\longrightarrow \underline{\underline{x_i'}}$

✓ Radial Basis fn (RBF / Gaussian)

- most popular
- general default if you dont have a domain specific kernel

$$K_{RBF}(x_1, x_2) = \exp\left\{ - \frac{\| x_1 - x_2 \|^2}{2\sigma^2} \right\}$$

euc. dist (x_1, x_2)

σ^2 ← hyper-param

$$K_{RBF}(x_1, x_2) = \exp\left(\frac{-d_{12}^2}{2\sigma^2}\right)$$

$$d_{12}^2 = \|x_1 - x_2\|^2$$

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:27

plot(exp(-x^2/(2*1*1))) - Goog

google.com/search?q=plot%28exp%28-x%5E2%2F%282*1*1%29%29%29&rlz=1C5CHFA_enIN958IN958&ei=gVGOYq6FH6iQseMPgv6esAI&ved=0ahUKEwjuybnZgfv3AhUoSGwGHQK_ByYQ4dUDCA4&uact=5&oq=plot%28exp%28-x%...

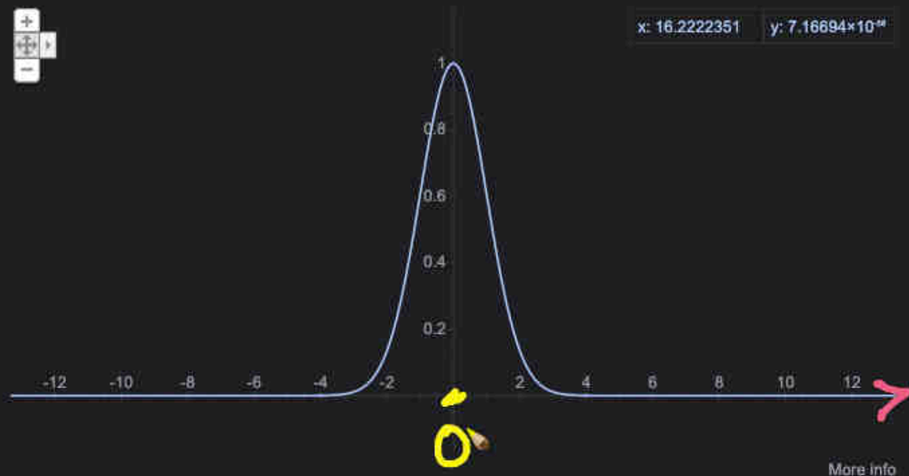
Google

plot(exp(-x^2/(2*1*1)))

All Shopping Maps Videos News More Tools

About 2,65,00,00,000 results (1.09 seconds)

Graph for exp((-(x^2))/(2*1*1))



More info

https://www.mathway.com › popular-problems › Calculus

Graph e^(-x^2) | Mathway

Graph e^(-x^2). **e−x2 e - x 2**. Find where the expression **e−x2 e - x 2** is undefined. The domain of the expression is all real numbers except where the ...

https://www.wolframalpha.com › examples › mathematics

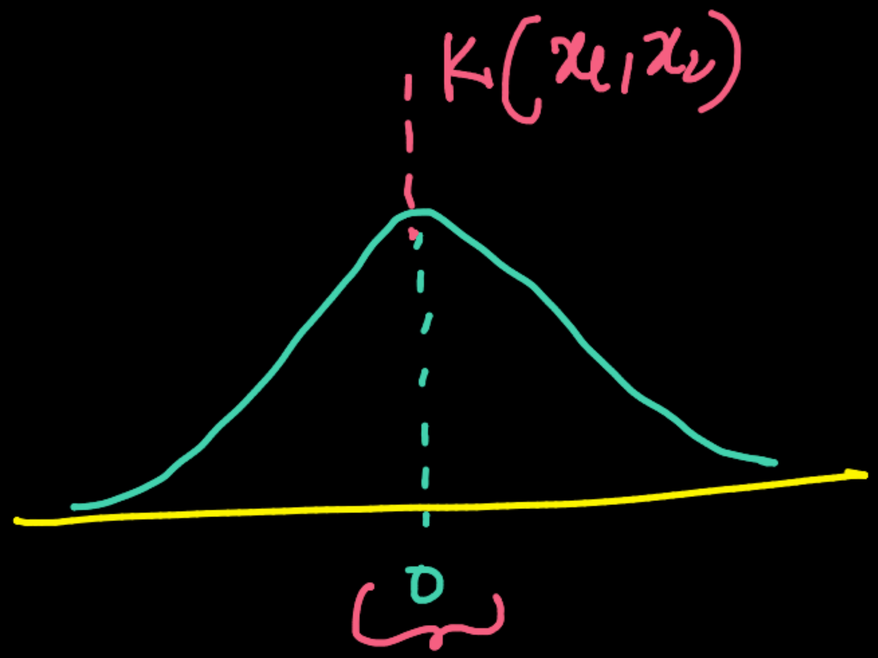
Plotting & Graphics - Wolfram|Alpha Examples

Use interactive calculators to **plot** and **graph** functions. Try 3D **plots**, equations, inequalities, polar and parametric **plots**. Specify ranges for variables.

Plot sin x, cos x, tan x · Plot x^2 y^3, x=-1..1, y=0..3 · Log-linear plot x^2 log x, x=1...

People also ask

11 / 11



K_{RBF}

$d_{12} = \|x_1 - x_2\|$

Similarity

$K(x_1, x_2)\uparrow$

x_1 is v. close to x_2 :

eucl. dist $(x_1, x_2)\uparrow$: $K(x_1, x_2)\downarrow$

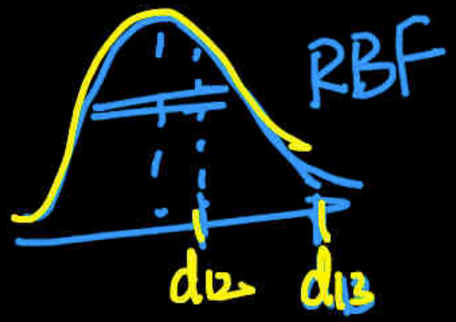
1. $\text{euc.dist}(x_1, x_2) \downarrow \Rightarrow K_{RBF}(x_1, x_2) \uparrow$ (Sim)

2. $\text{euc dist}(x_1, x_2) \uparrow \Rightarrow$ Sim or K_{RBF} falls exponentially

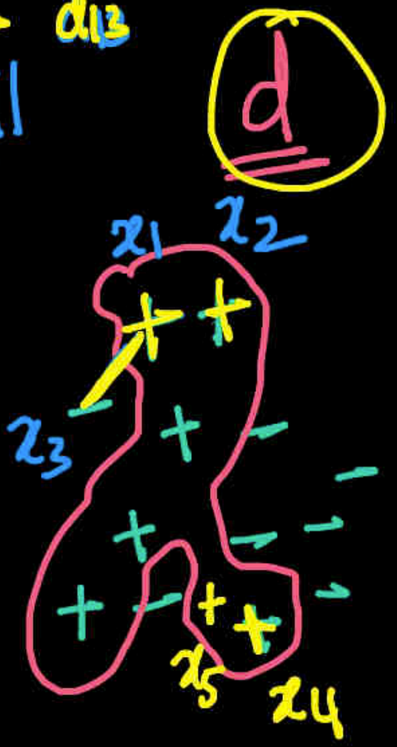
SVM + RBF

dual formulation → $x_i^T x_j$: $K_{RBF}(x_i, x_j)$

x_i is close to x_j →

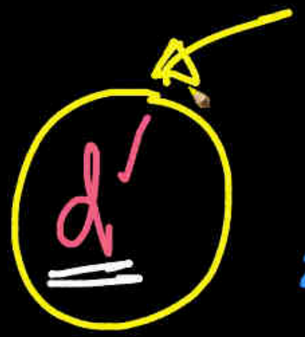


σ small



Kernel

RBF



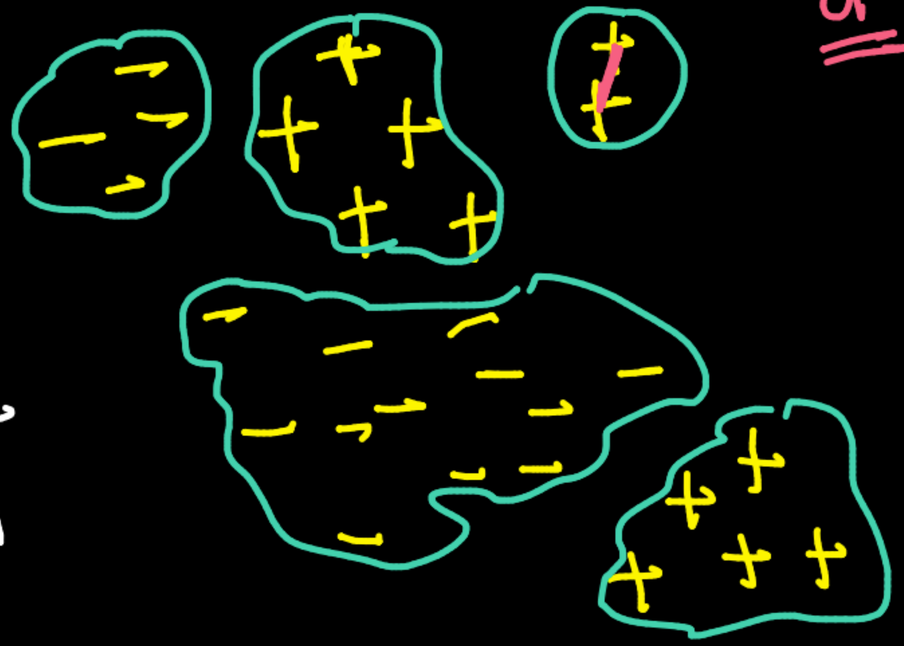
x_1' x_2'

x_3'

x_5' x_4'

$x_1'^T x_3'$: Small

d-dim



d'

σ RBF is small

KNN with small 'K'

≈

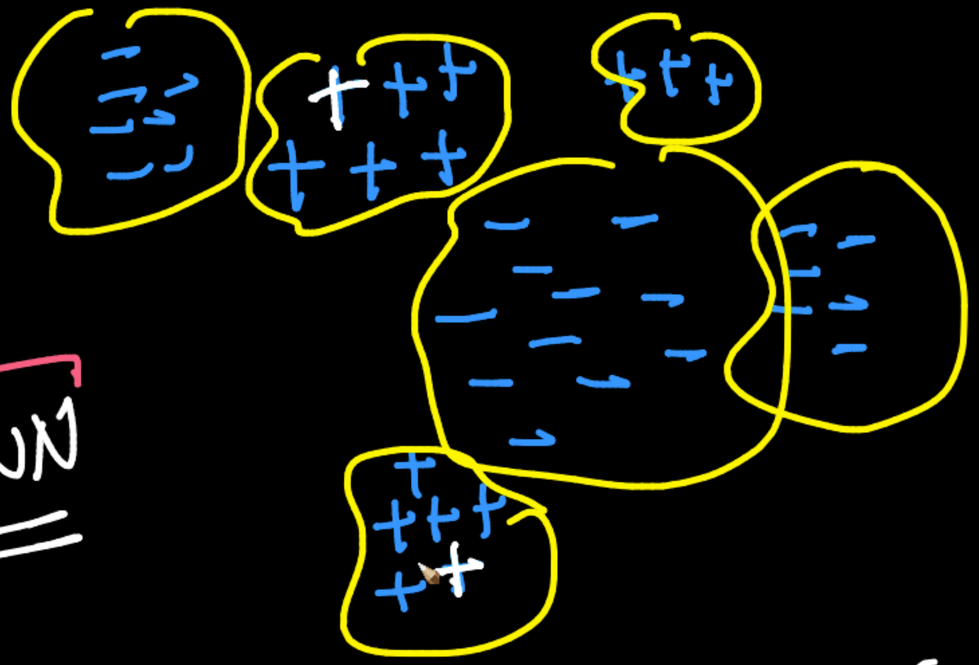
Proximity based Model

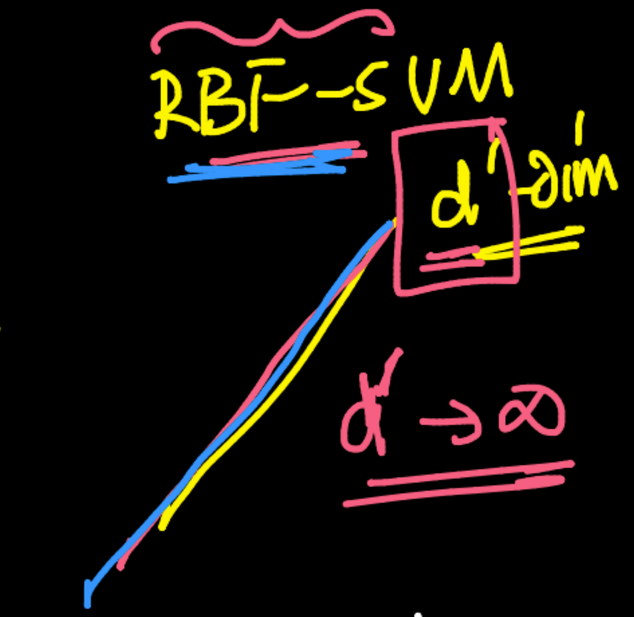
{ RBF-SVM with small σ

overfit

Outlier

overfit





✓ K-NN

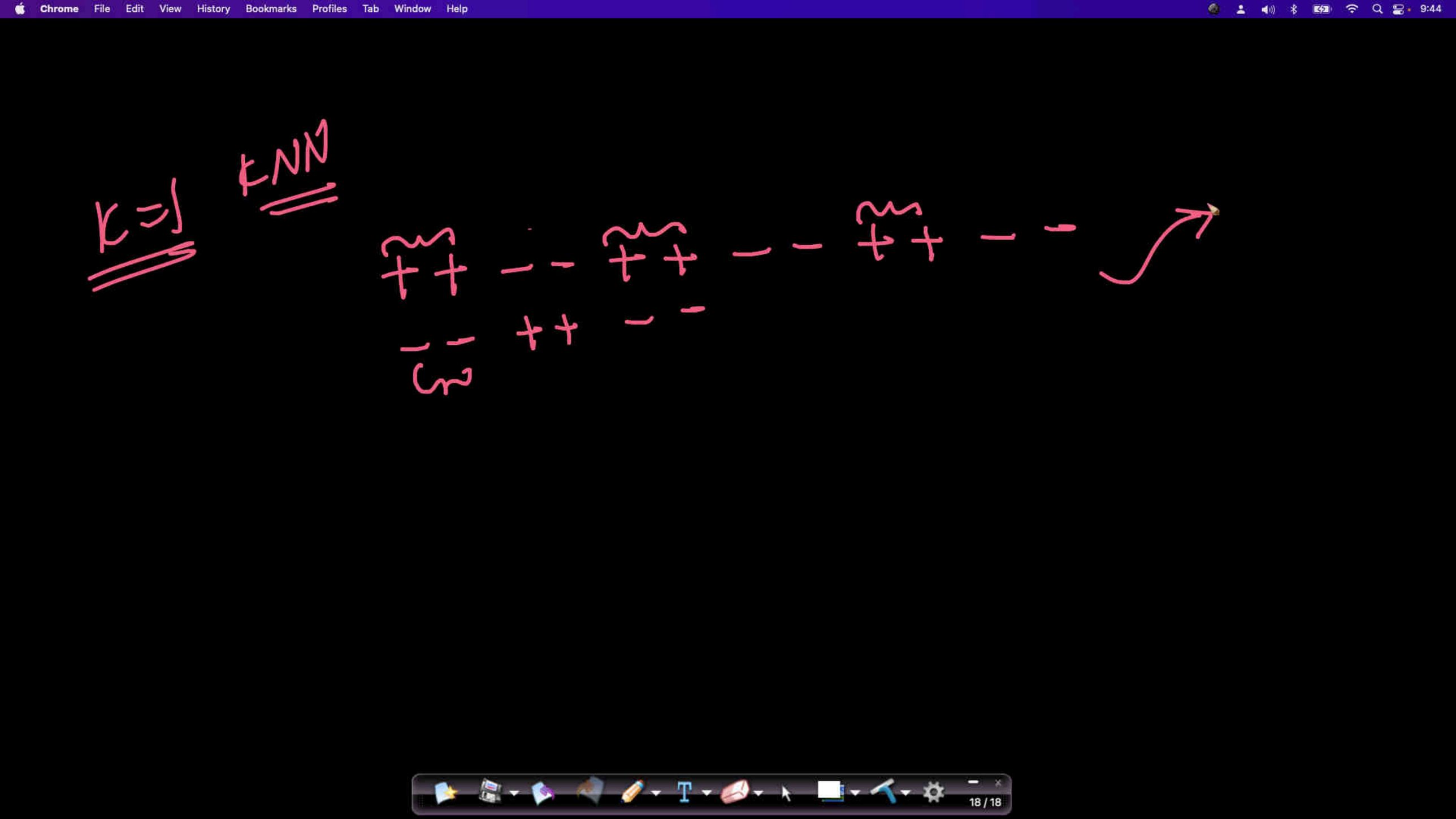
K

overfit

(proof)

∇ in RBF-SVM

overfit


K=1
KNN

KNN → d ↑

curse of dim

Euc. dist doesn't work

RBF ~ SUM !

↳ RBF

$\|x_1 - x_2\| = d_{12}$

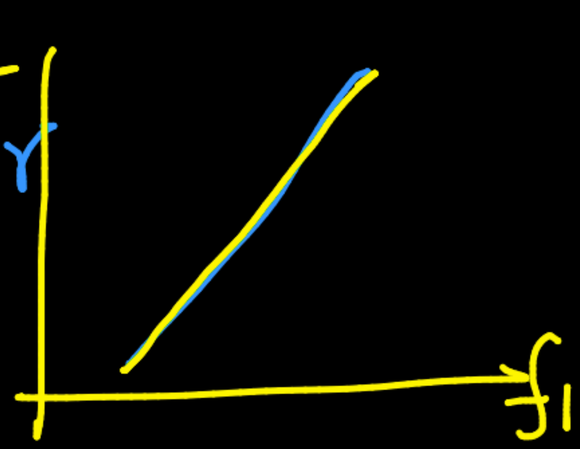
$func(d_{12})$

$x_i \in \mathbb{R}^d$ $d = 10$

INTUITIVE

Q

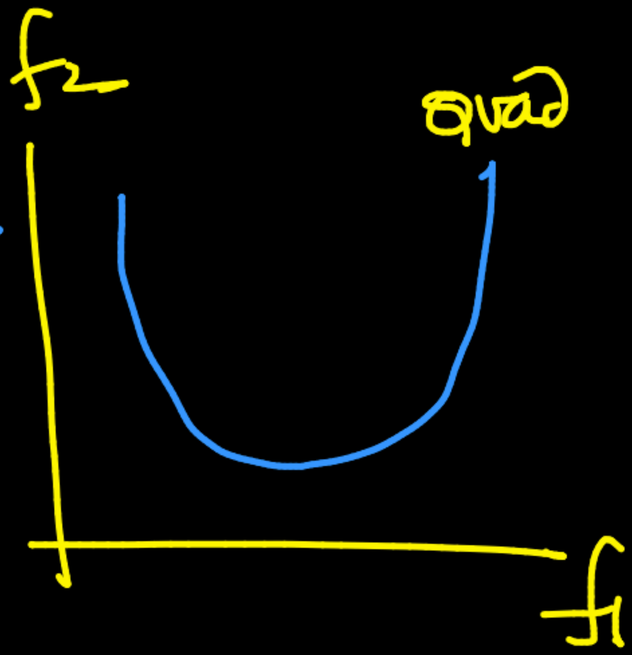
d → linear



Linear model

(2) → f_1, f_2

Quadratic curve

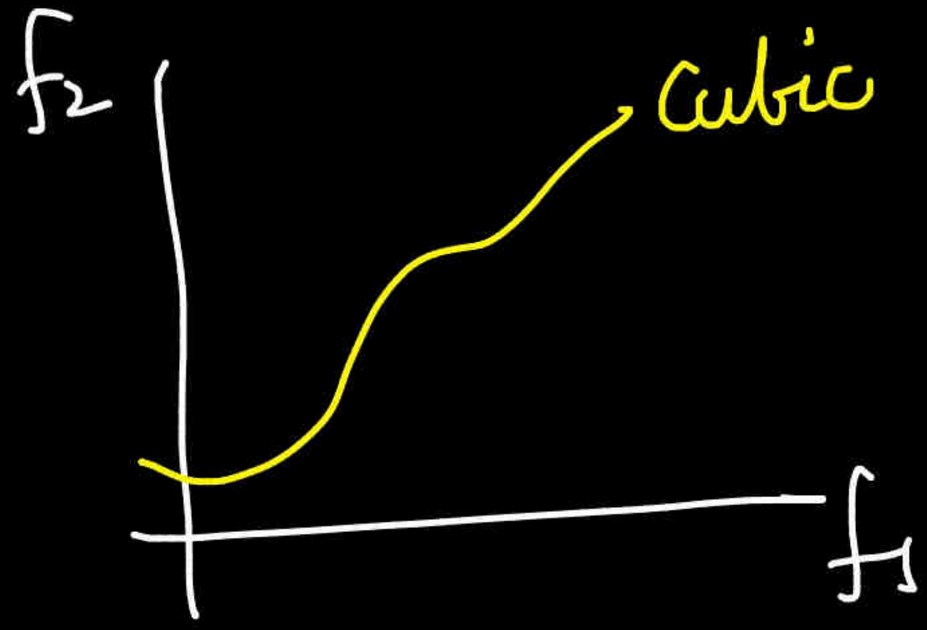


(5) $f_1, f_2, f_1^2, f_2^2, f_1 f_2$

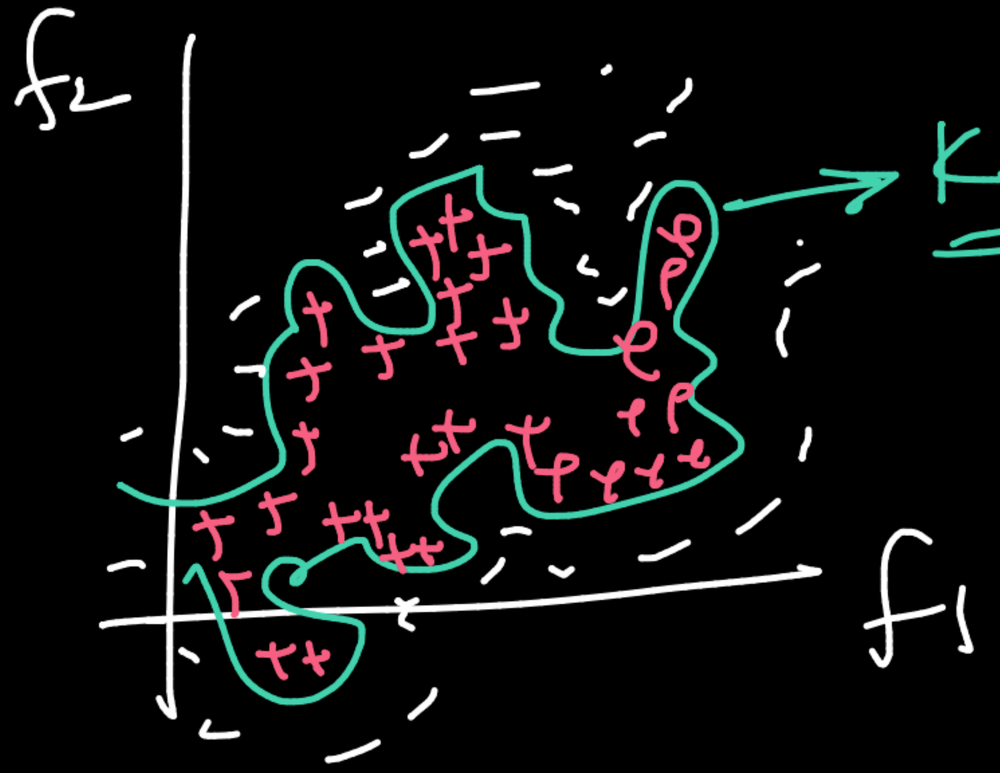
↓

linear model

9 → $f_1, f_2, f_1^2, f_2^2, f_1 f_2, f_1^3, f_2^3, f_1^2 f_2, f_2 f_1^2$



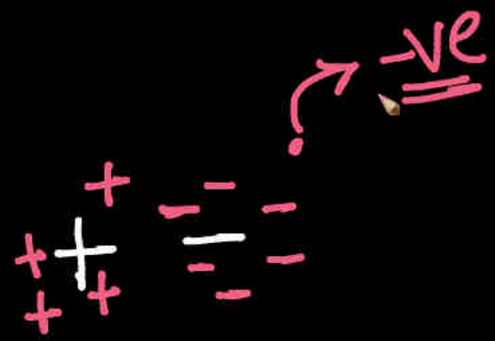
Do many:- $f_1, f_2, f_1^2, f_2^2, f_1^3 - - - - -$



KNN Small k

RBF-SVM Small σ

Q



KNN जैसे $K=1$
||
RBF-SVM smaller

pink : 2a)

white : train

Q

$K_{RBF}(x_1, x_2)$

d-dim

d

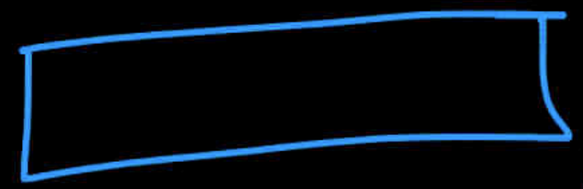
$+x_1$

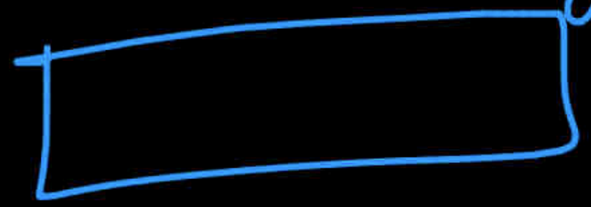
$+x_2$

SIM

d'

$+x_1'$

x_1' : 

x_2' : 

d'

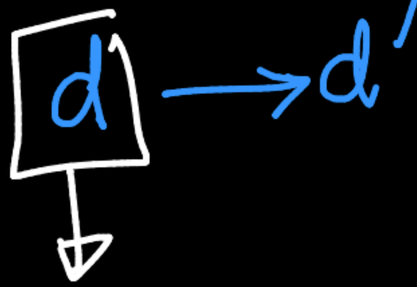
d'

IMPOSSIBLE

$d' \to \infty$ -laye

Q

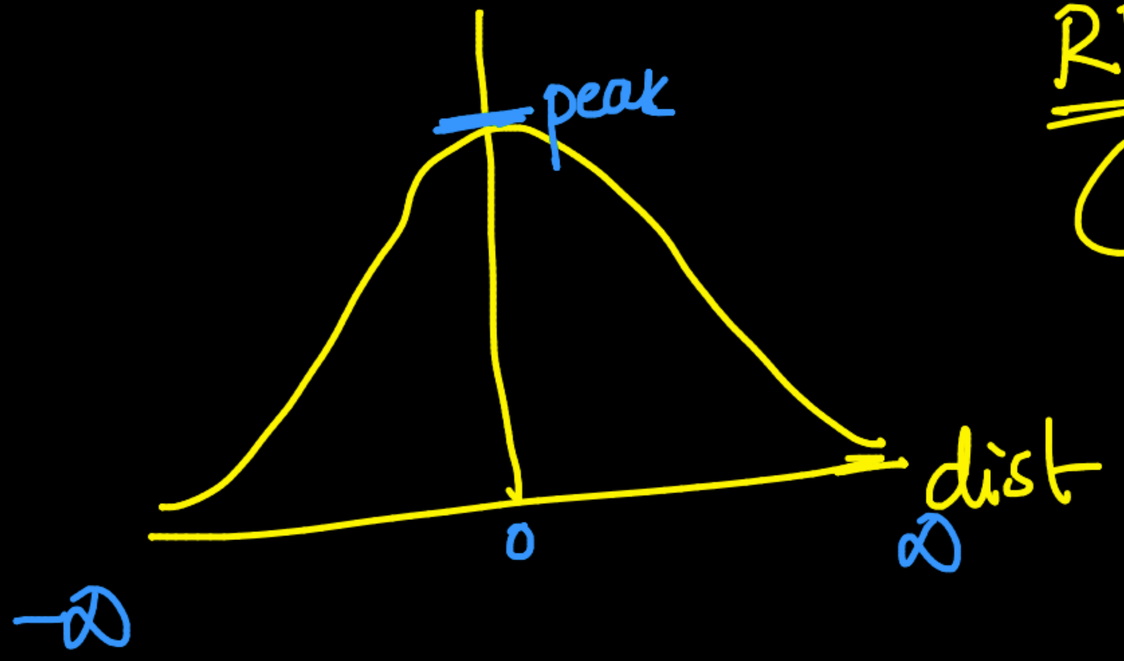
$-\infty \leq \text{dist}(x_1, x_2) \leq +\infty$



1000

(let)

euc. dist



RBF

↳ dist → RBF(dist)

$\|x_1 - x_2\|^2 = d_{12}^2$ (Case dalīm)

Summary

[RBF SVM $\xrightarrow[\text{Similar}]{\approx}$ KNN

$\sigma \downarrow$ $\quad$ $K \downarrow$

{ geometrically

Run-time Complx SVM:- (Test)

$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x_q) + b$

$\hat{y}_q$ $\quad$ d-dim

$\hookrightarrow O(n)$, $O(n^2)$ $O(d)$

$$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i K_{RBF}(x_i, x_q) + b$$

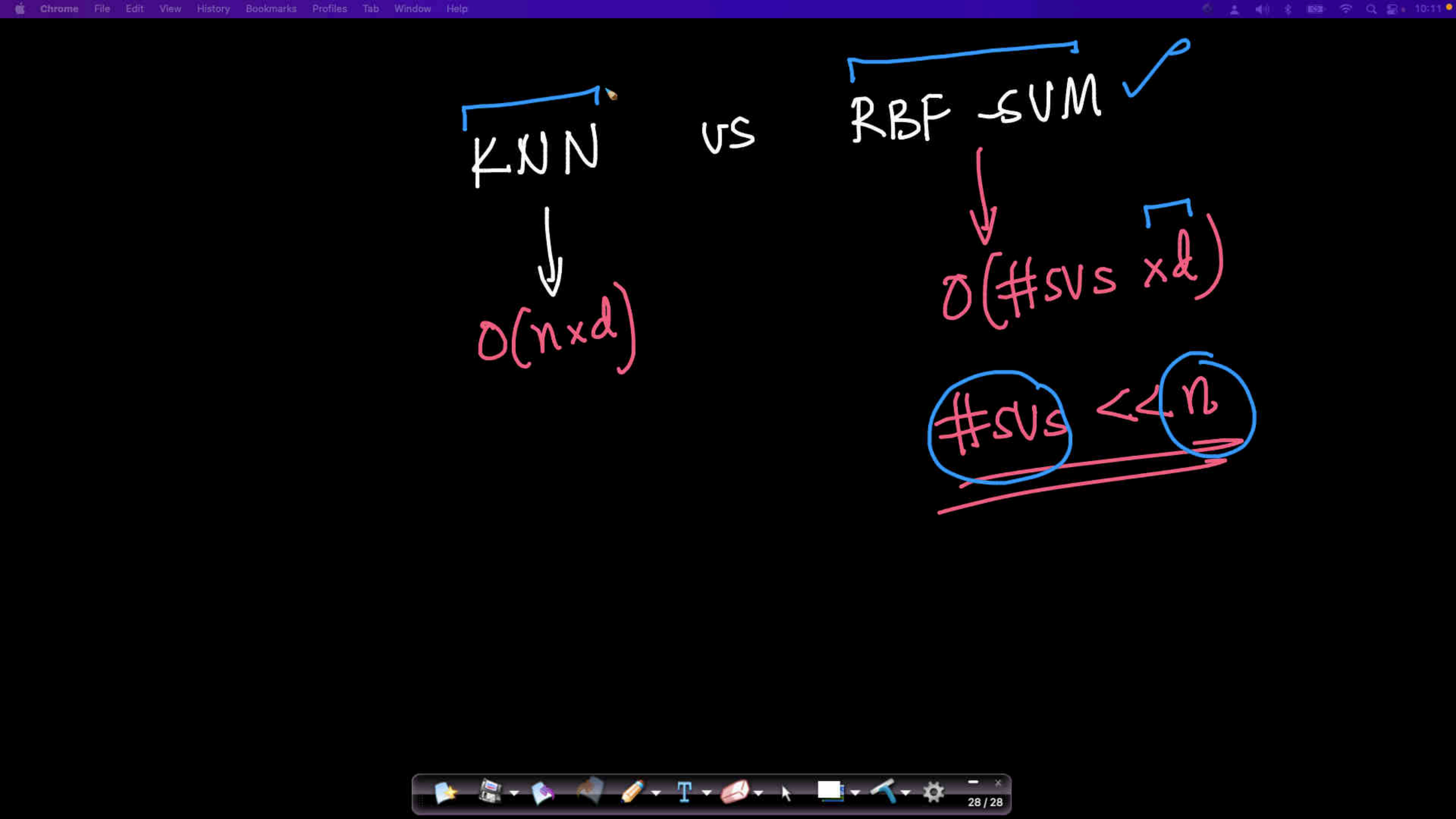
x_i, x_q: d-dim

$\alpha_i = 0$ for non SVs

$\alpha_i > 0$ for SVs

Operating in d-dim

Run-time $= O(\#SVs \times d)$


KNN
vs
RBF SVM
O(n x d)
O(#svs x d)
#svs << n

Summarize

lr-SVMs

1. Margin-max linear model
   - ↳ regularization
   - → hard-margin
   - → soft-margin
     - almost lr. sep
     - ↓
     - min hinge-loss

soft-margin lr-SVM ≈ logistic reg

(2) non-linear / Kernel SVMs

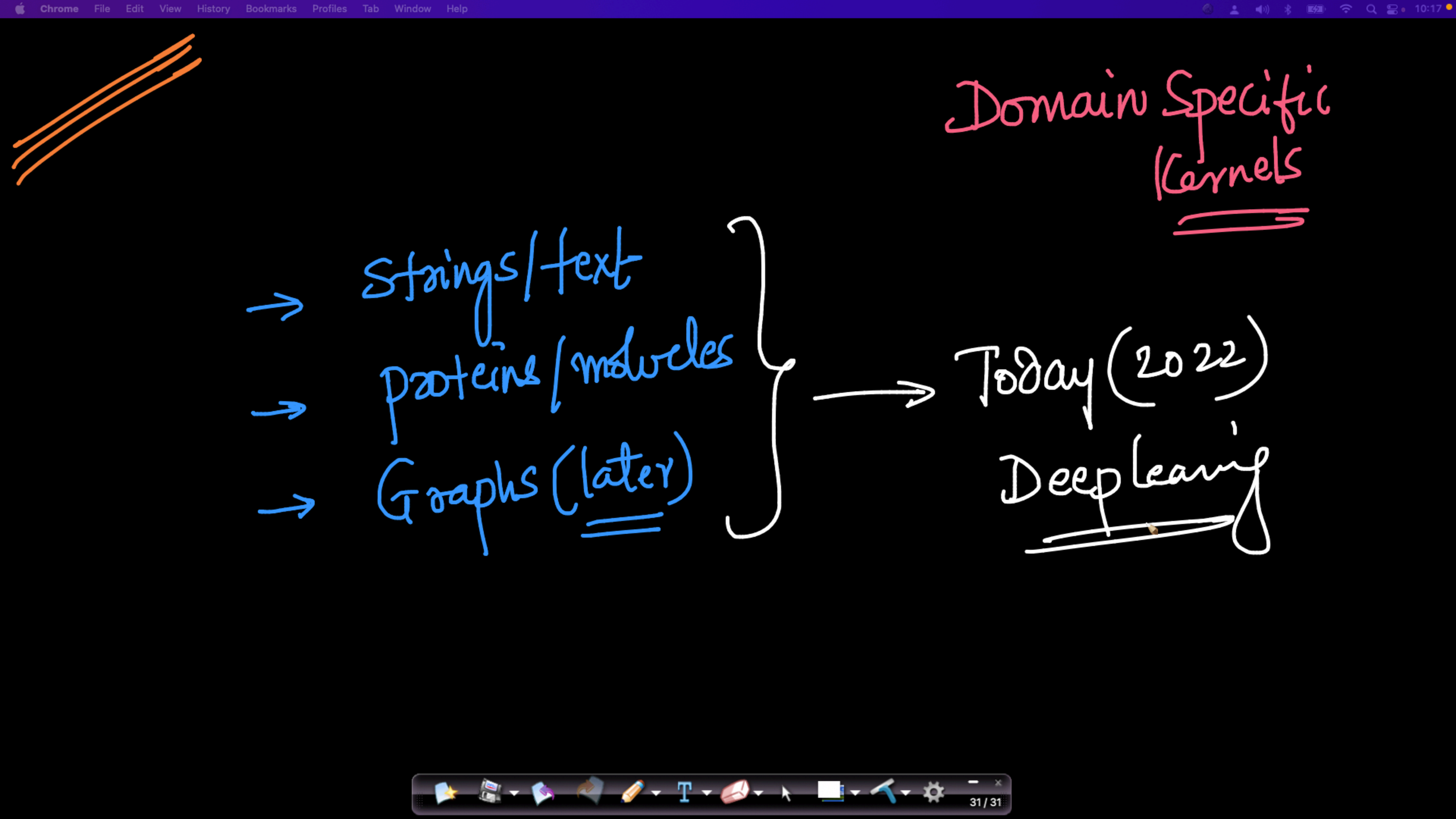
- primal — dual

$x_i^T x_j$ → $K(x_1, x_2)$: Similarity

- Quad. Kernel : 2d — 6d (explicitly)

- RBF / Gaussian Kernel → Geom:- $d \to d'$ (∞-dim)

RBF-SVM (σ) ↓ ≈ KNN (K) ↓

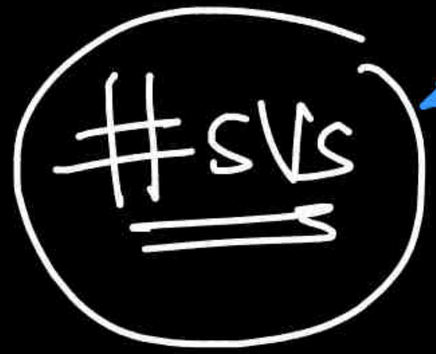

Domain Specific Kernels
Strings / text
Proteins / molecules
Graphs (later)
Today (2022)
Deep learning

SVMs → libsvm (C++) → sklearn internally

↳ SGD ; alternative GD (SMO)

SVMs! Training time :- $O(n^2)$ (improved later...on)

$n = 1B$

Runtime Complex:- 

Controlling it is hard

Concept:

intuition ?

Kernel - trick

SVMs

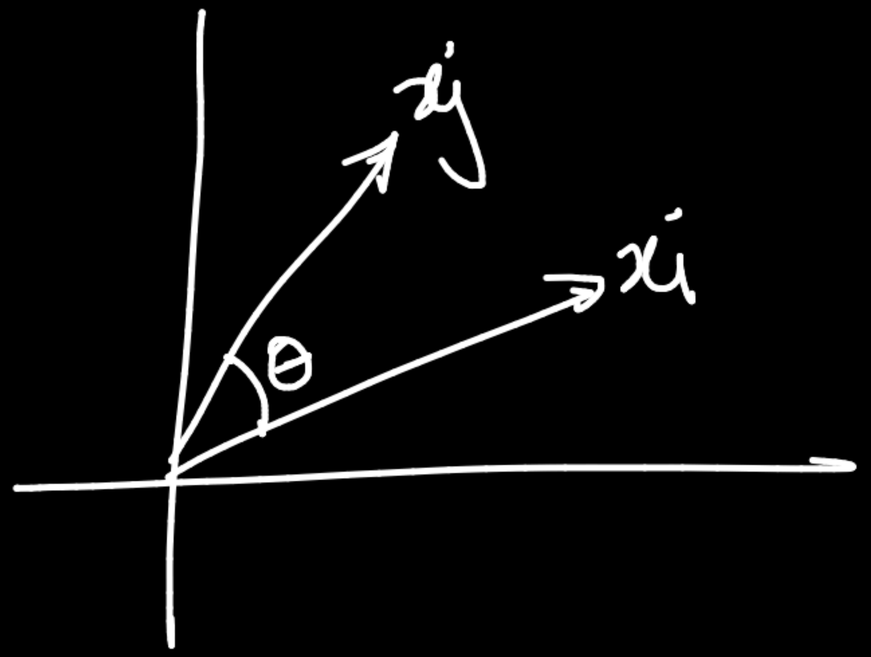
$d \longrightarrow d'$ (∞ - large)

Ques

$K(x_i, x_j)$

$x_i^T x_j = \|x_i\| \|x_j\| \cos\theta$

angular Sim



$$K_{Quad}(x_1, x_2) = (1 + x_1^T x_2)^2$$

$$K_{RBF}(x_1, x_2) = \exp\left\{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right\}$$

Simplest - formulation

not very popular

SV-R

(Kernelized -SVR)

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\epsilon$$

$y_i - \hat{y}_i$

$\hat{y}_i$

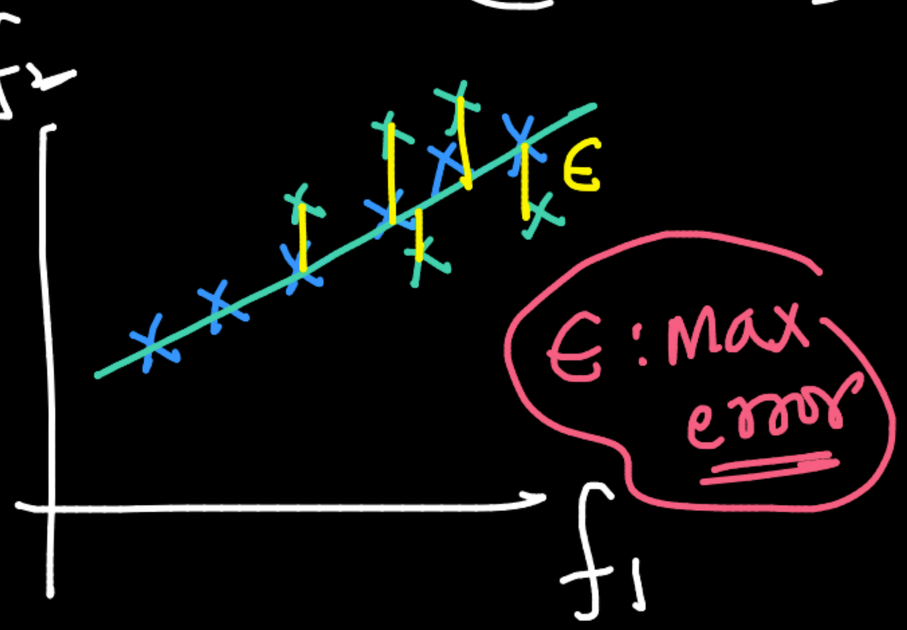
s.t $y_i - (w^T x_i + b) \leq \epsilon$

$(w^T x_i + b) - y_i \leq \epsilon$

$\epsilon \geq 0$

hard-SVR

Cl: $\xi_i \geq 0$



Hyper-params!

RBF-SVM ① $\sigma \downarrow$ overfit

Primal:-

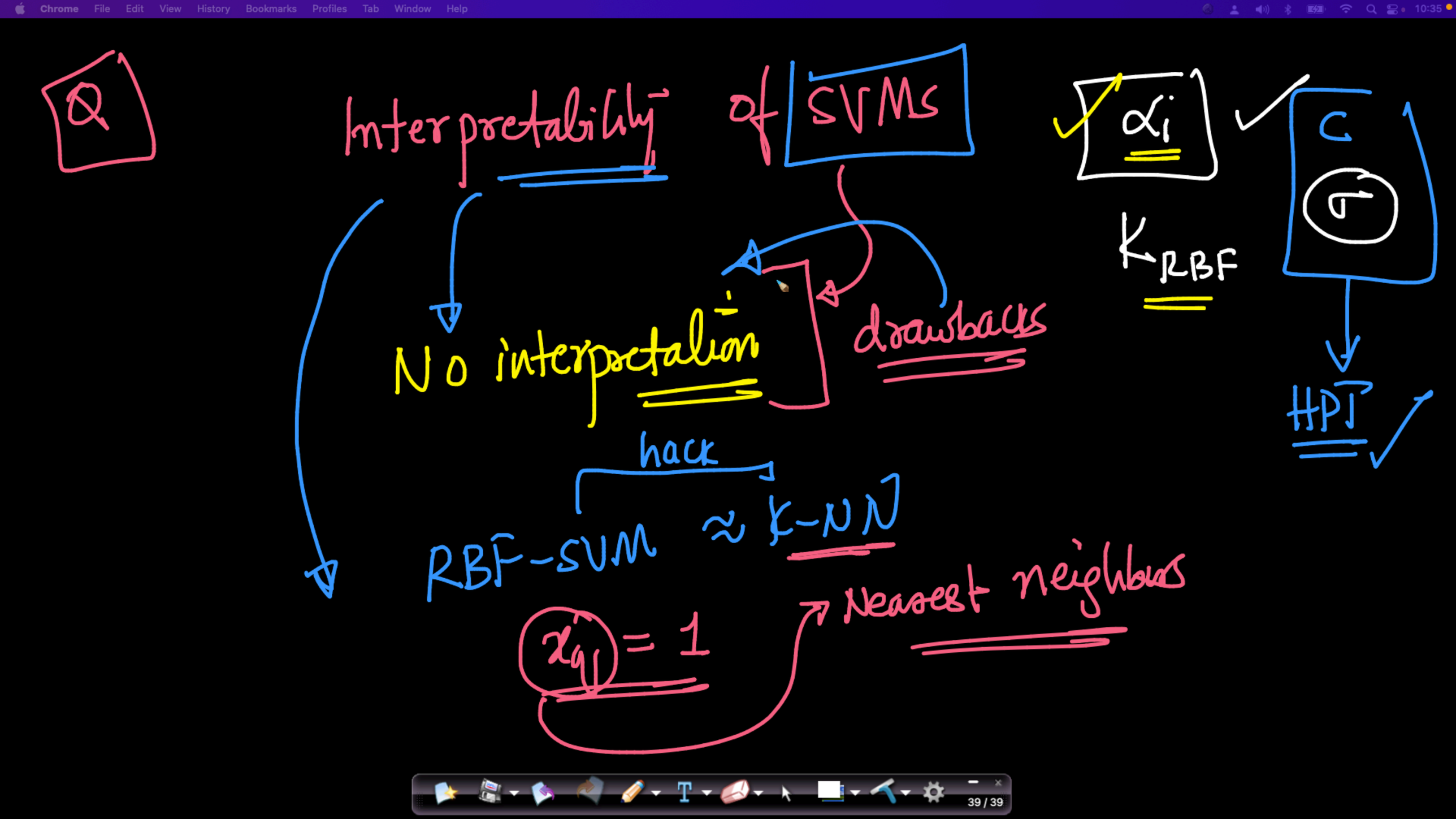
$$\min \quad \frac{1}{2}\|w\| + C \sum_{i=1}^{n} \xi_i$$

$$s.t.$$

reg ↑ loss ↑

② $C \uparrow$ overfit

log-reg: ↑ overfit


Q
Interpretability of SVMs
No interpretation
drawbacks
hack
RBF-SVM ≈ K-NN
$x_{qi} = 1$
Nearest neighbors
α_i
K_{RBF}
C
σ
HPT

sklearn.svm.SVC

class `sklearn.svm.`**SVC**(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=- 1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using **LinearSVC** or **SGDClassifier** instead, possibly after a **Nystroem** transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Pa

scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
sklearn.svm.SVC
Examples using sklearn.svm.SVC

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*

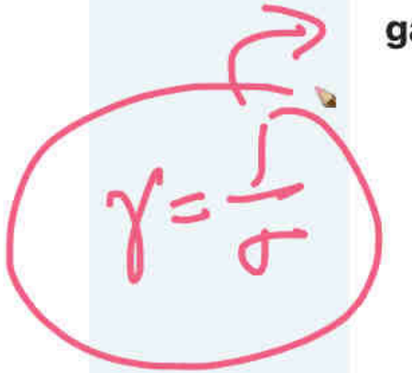
Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.



- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*

Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking : *bool, default=True*

Whether to use the shrinking heuristic. See the User Guide.

Toggle Menu

sklearn.svm.SVC

$w^T x_i + b$ w_0

class `sklearn.svm.`**SVC**(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=- 1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using **LinearSVC** or **SGDClassifier** instead, possibly after a **Nystroem** transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Pa

regularization is inversely proportional

sklearn.svm.SVC

class `sklearn.svm.`**SVC**(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=- 1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using **LinearSVC** or **SGDClassifier** instead, possibly after a **Nystroem** transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Pa

sklearn.svm.SVC

class `sklearn.svm.`**SVC**(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=- 1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using **LinearSVC** or **SGDClassifier** instead, possibly after a **Nystroem** transformer.

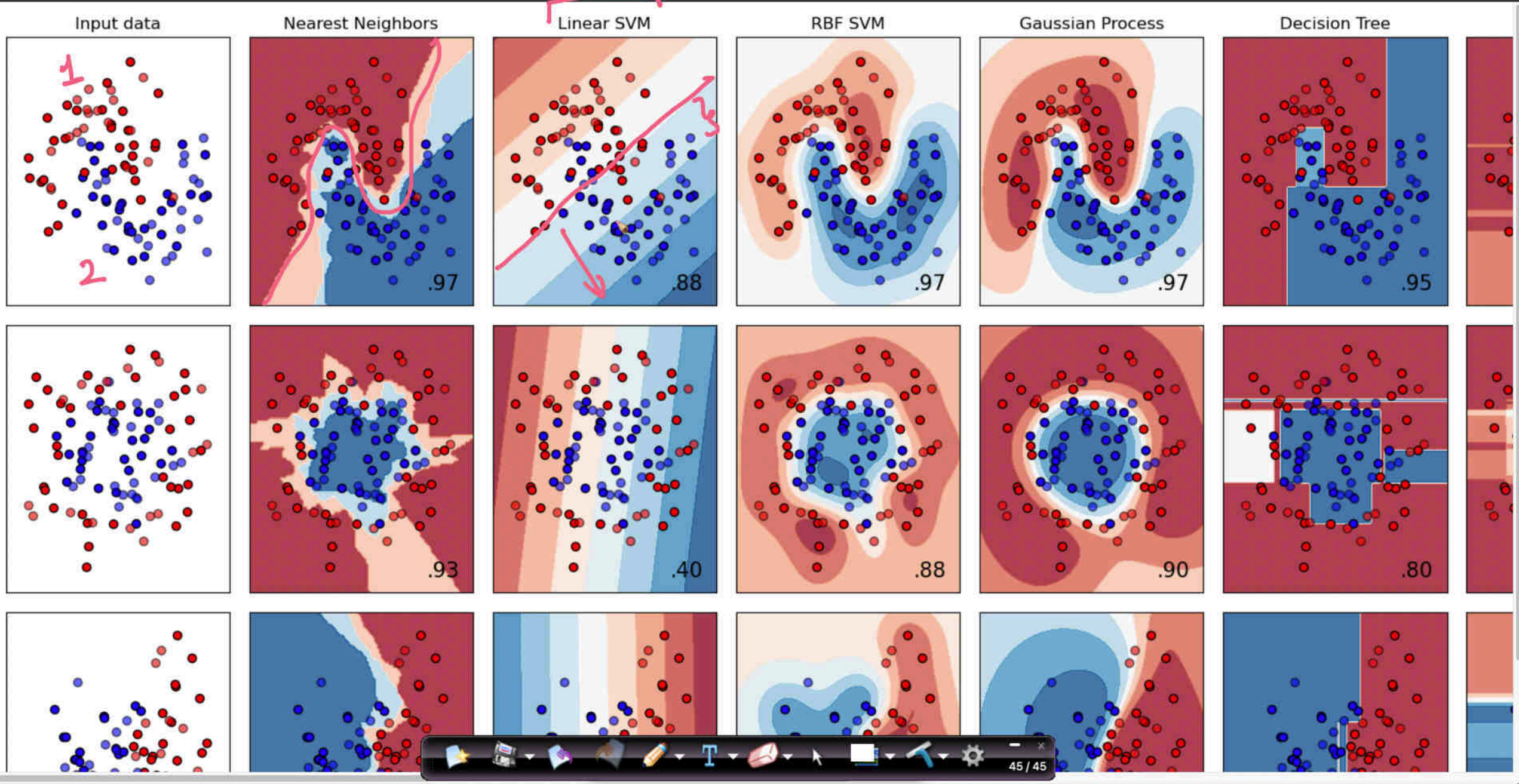
The multiclass support is handled according to a one-vs-one scheme.

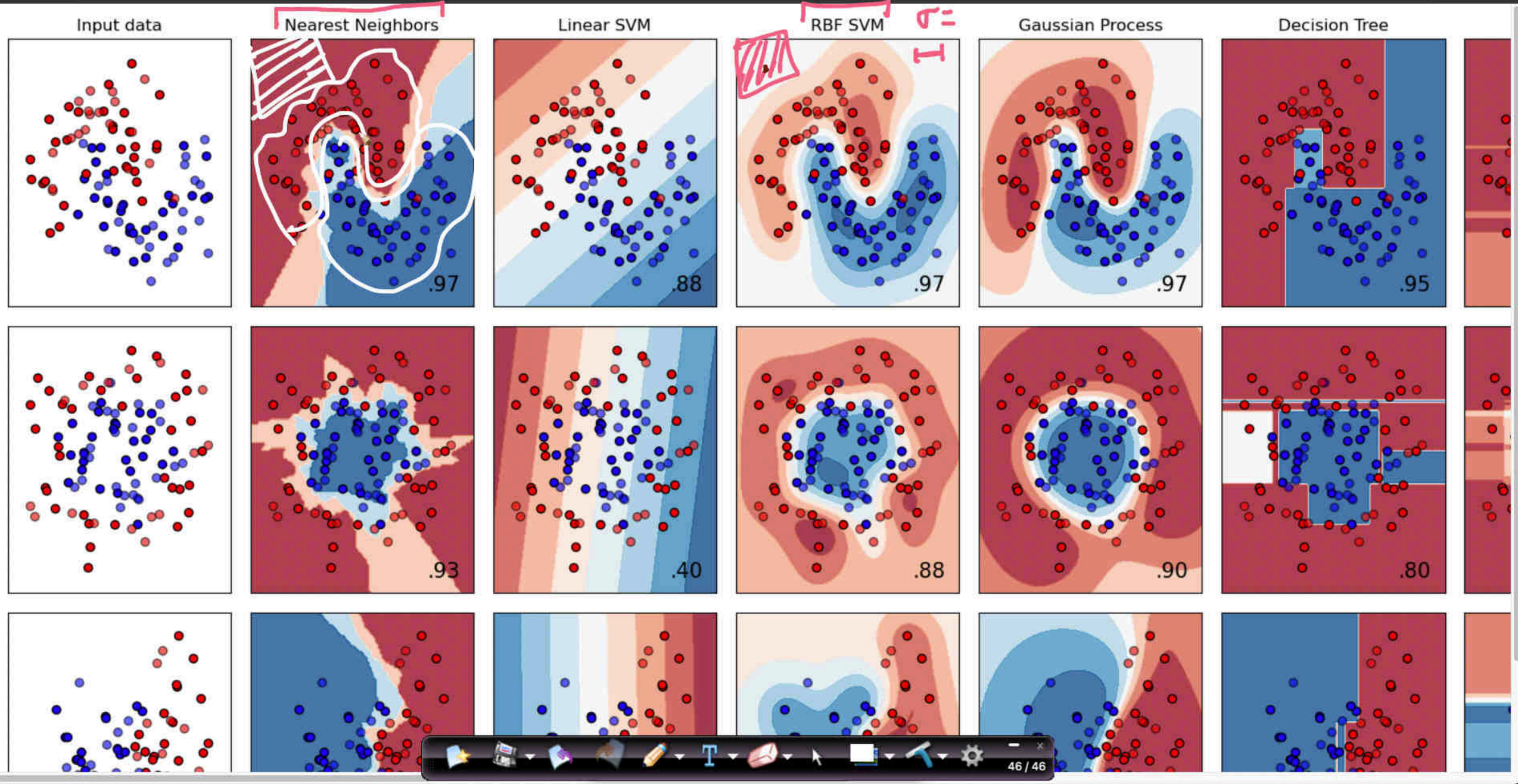
For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

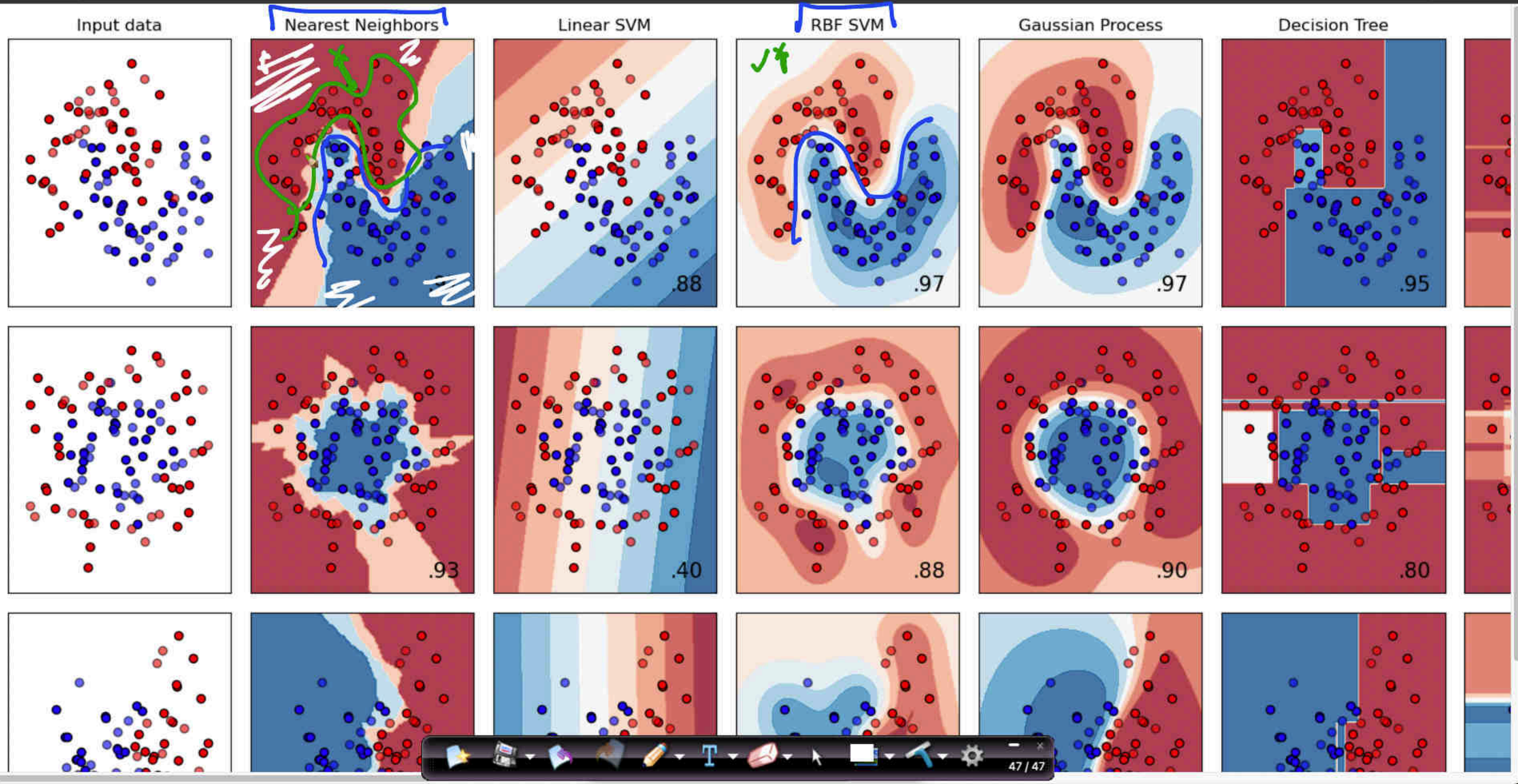
Read more in the User Guide.

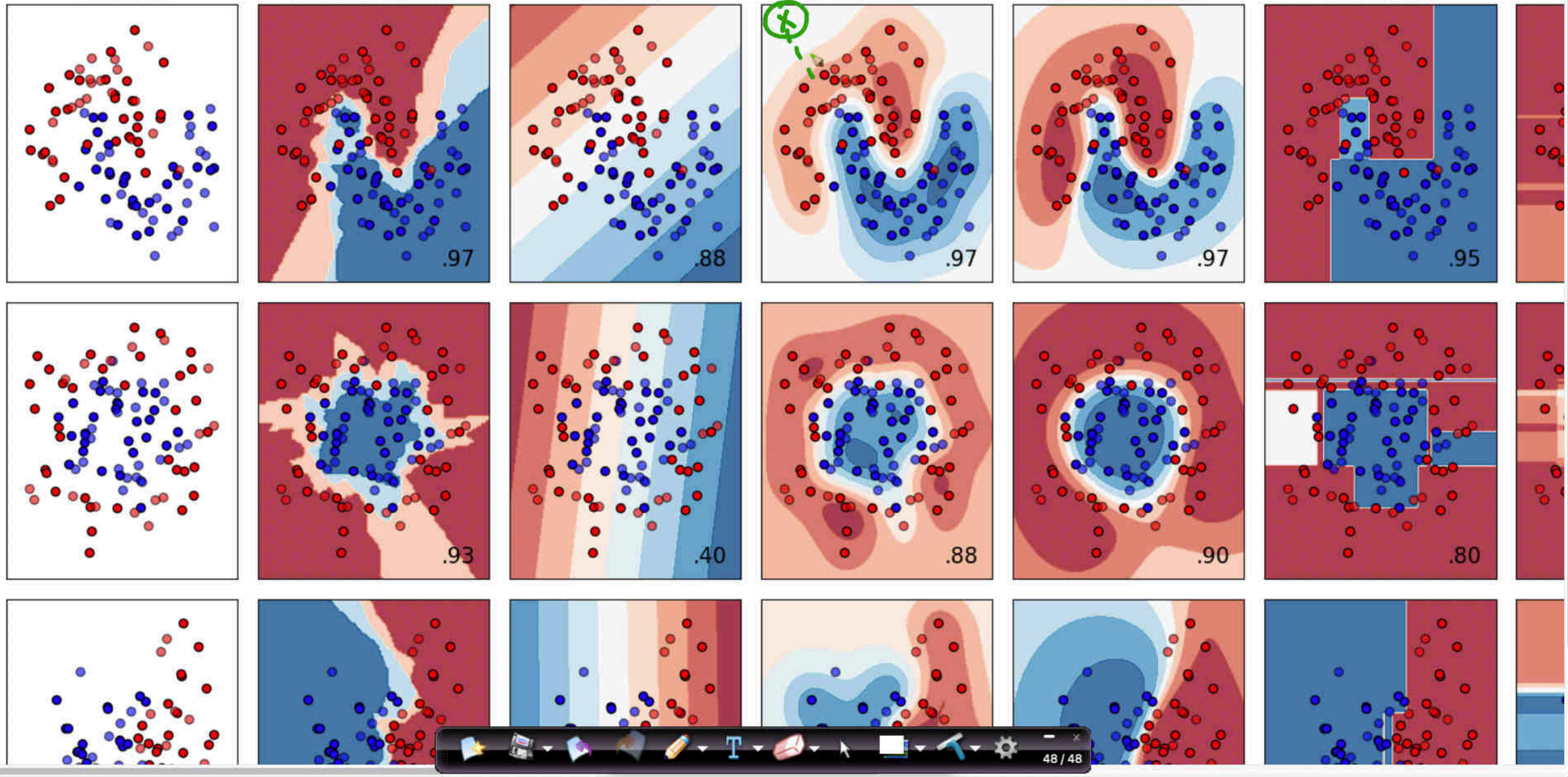
Pa

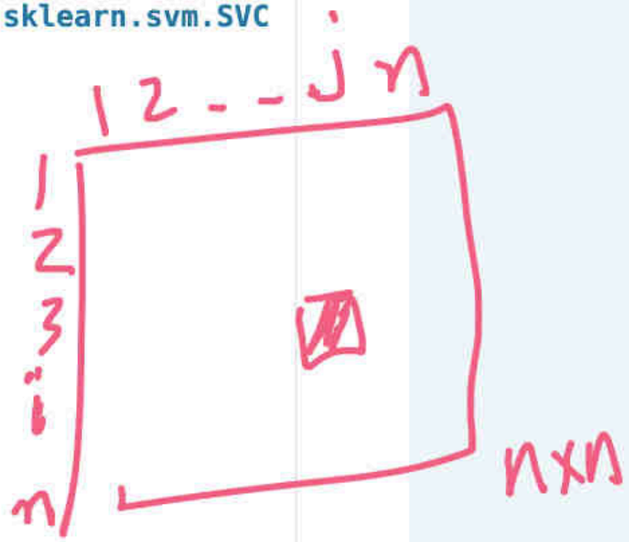
regularization is inversely proportional


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
1
2
3
.97
.88
.97
.97
.95
.93
.40
.88
.90
.80
45 / 45


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
.97
.88
.97
.97
.95
.93
.40
.88
.90
.80


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
.88
.97
.97
.95
.93
.40
.88
.90
.80


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
.97
.88
.97
.97
.95
.93
.40
.88
.90
.80



Parameters:

C : *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

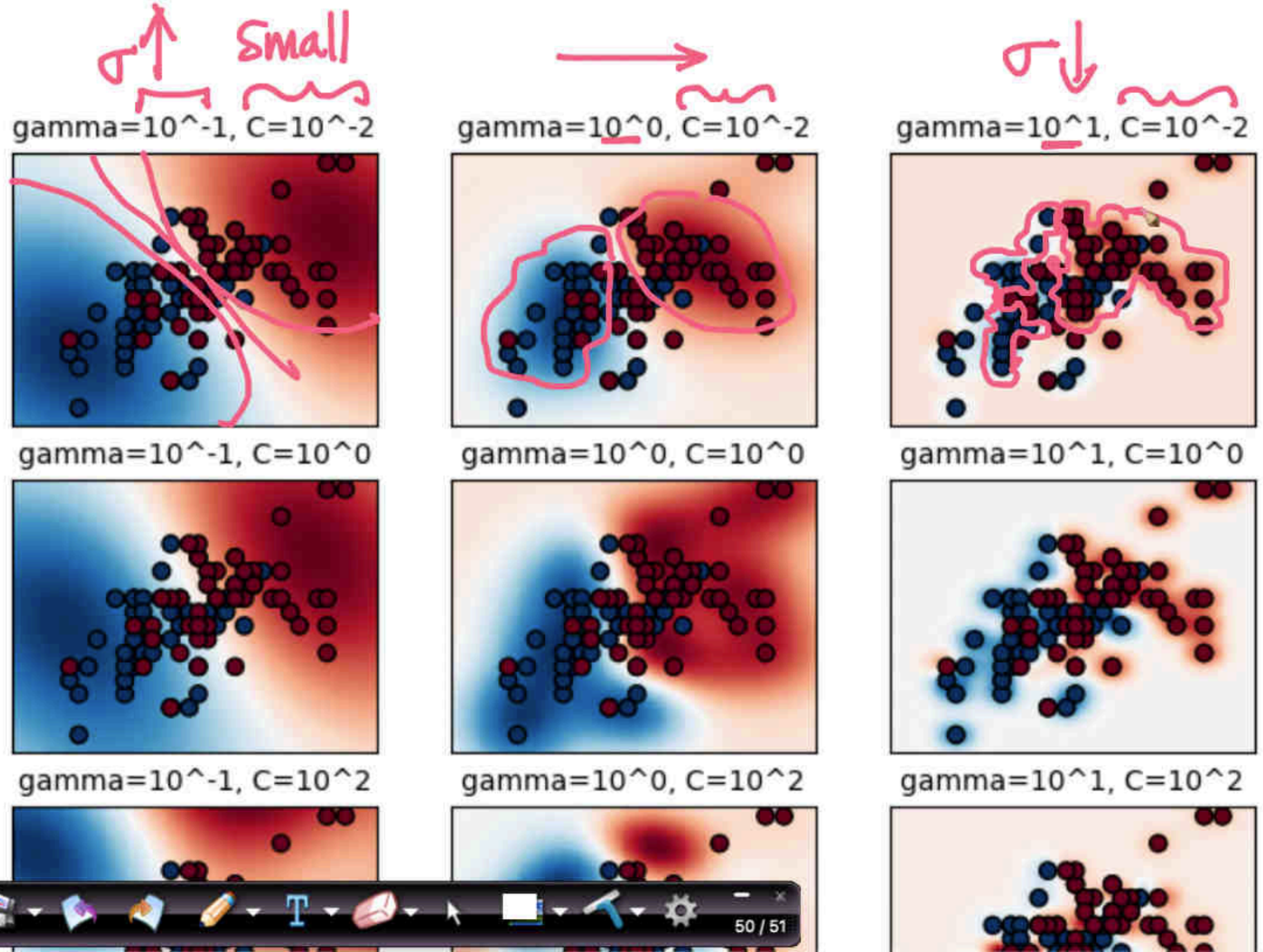
Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

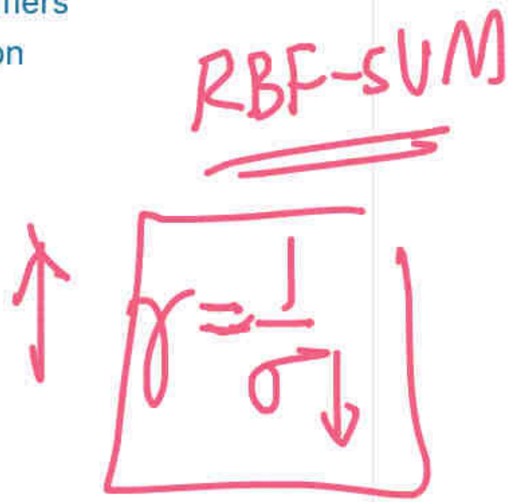
significant in 'poly' and 'sigmoid'.

scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
RBF SVM parameters
Load and prepare data set
Train classifiers
Visualization

```
    plt.yticks(())
    plt.axis("tight")

scores = grid.cv_results_["mean_test_score"].reshape(len(C_range), len(gamma_range))
```

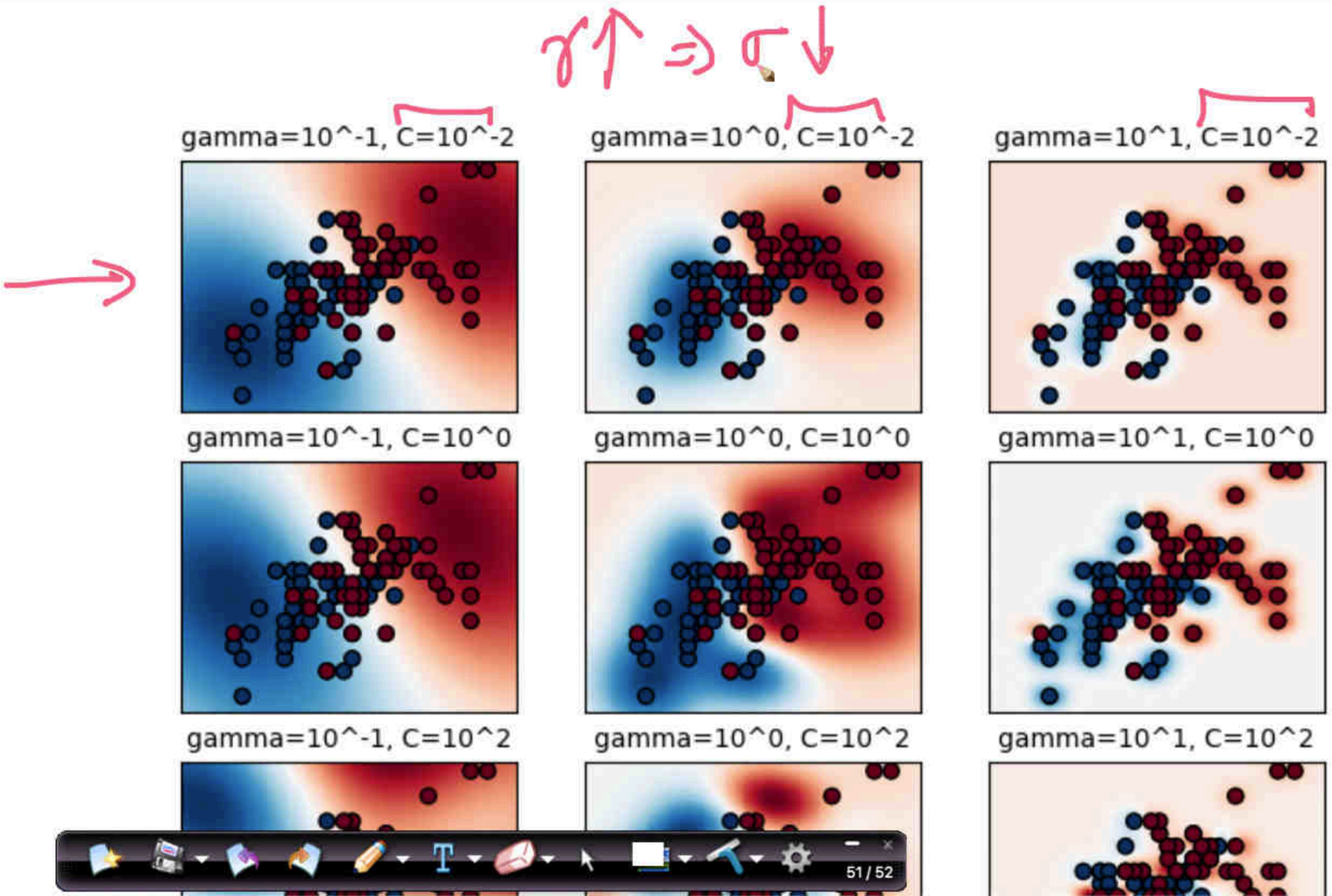


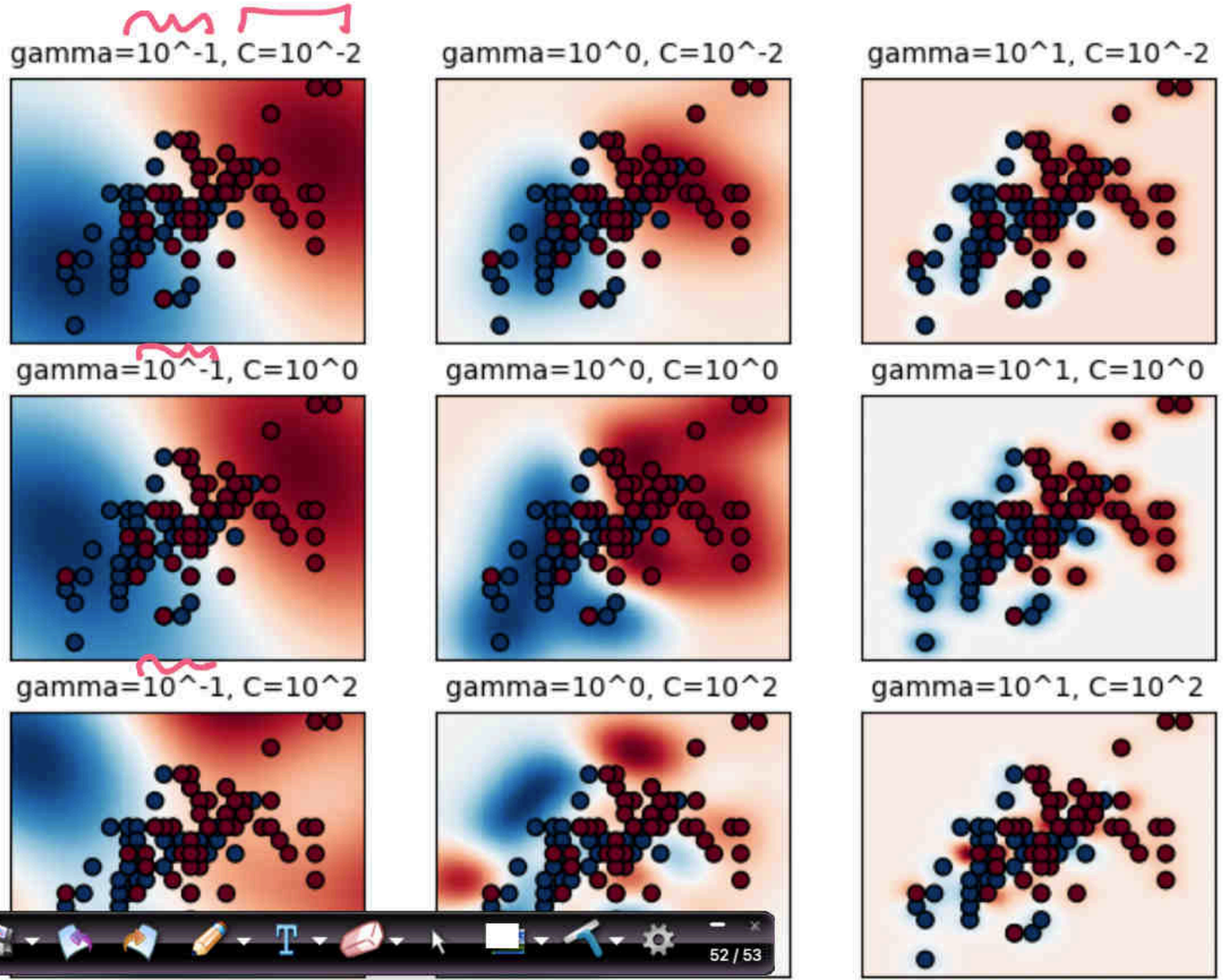


scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
RBF SVM parameters
Load and prepare data set
Train classifiers
Visualization

```
    plt.yticks(())
    plt.axis("tight")

scores = grid.cv_results_["mean_test_score"].reshape(len(C_range), len(gamma_range))
```




gamma=10^-1, C=10^-2
gamma=10^0, C=10^-2
gamma=10^1, C=10^-2
gamma=10^-1, C=10^0
gamma=10^0, C=10^0
gamma=10^1, C=10^0
gamma=10^-1, C=10^2
gamma=10^0, C=10^2
gamma=10^1, C=10^2
inc
C
52 / 53

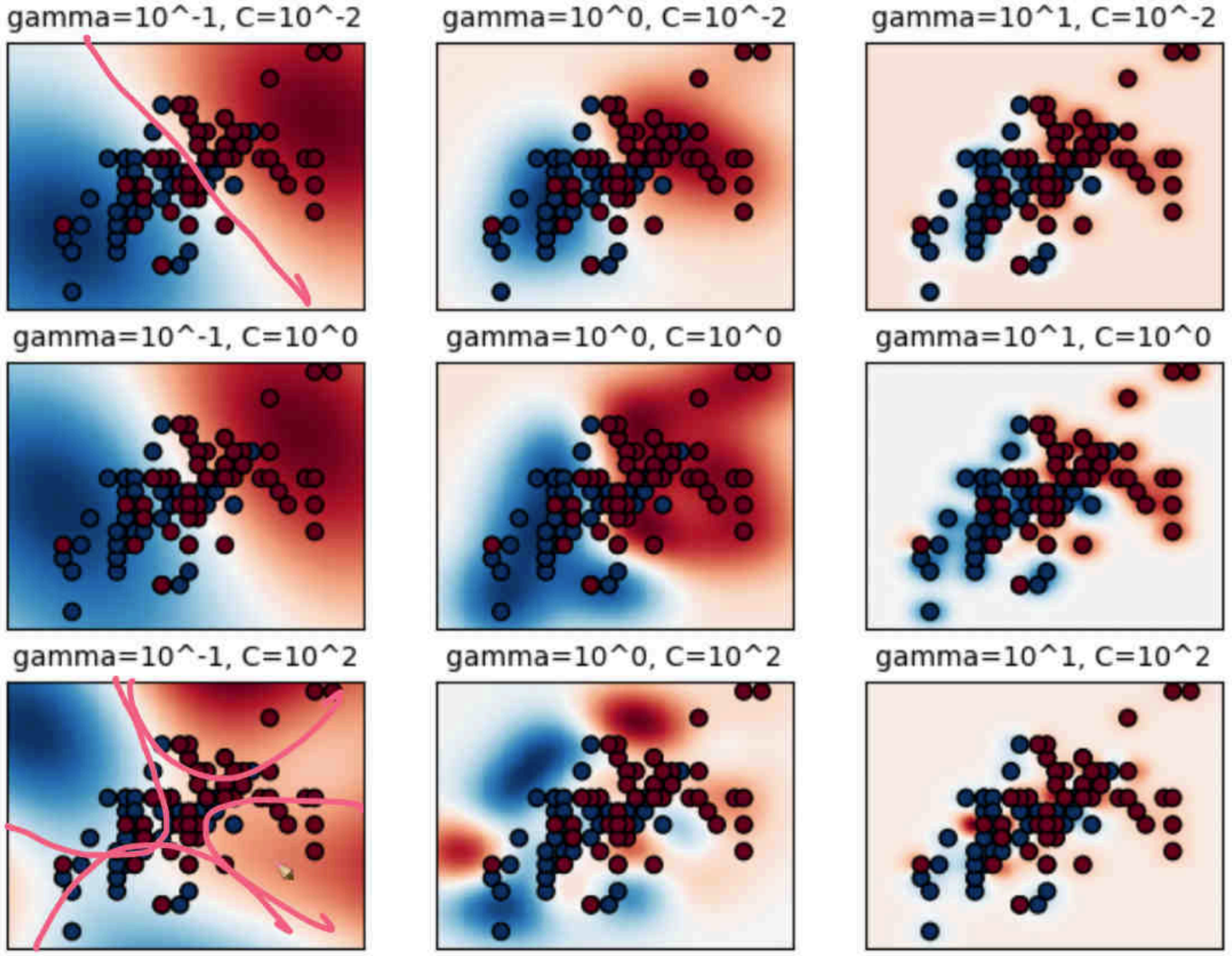
scikit learn

Prev Up Next

scikit-learn 1.1.1
Other versions

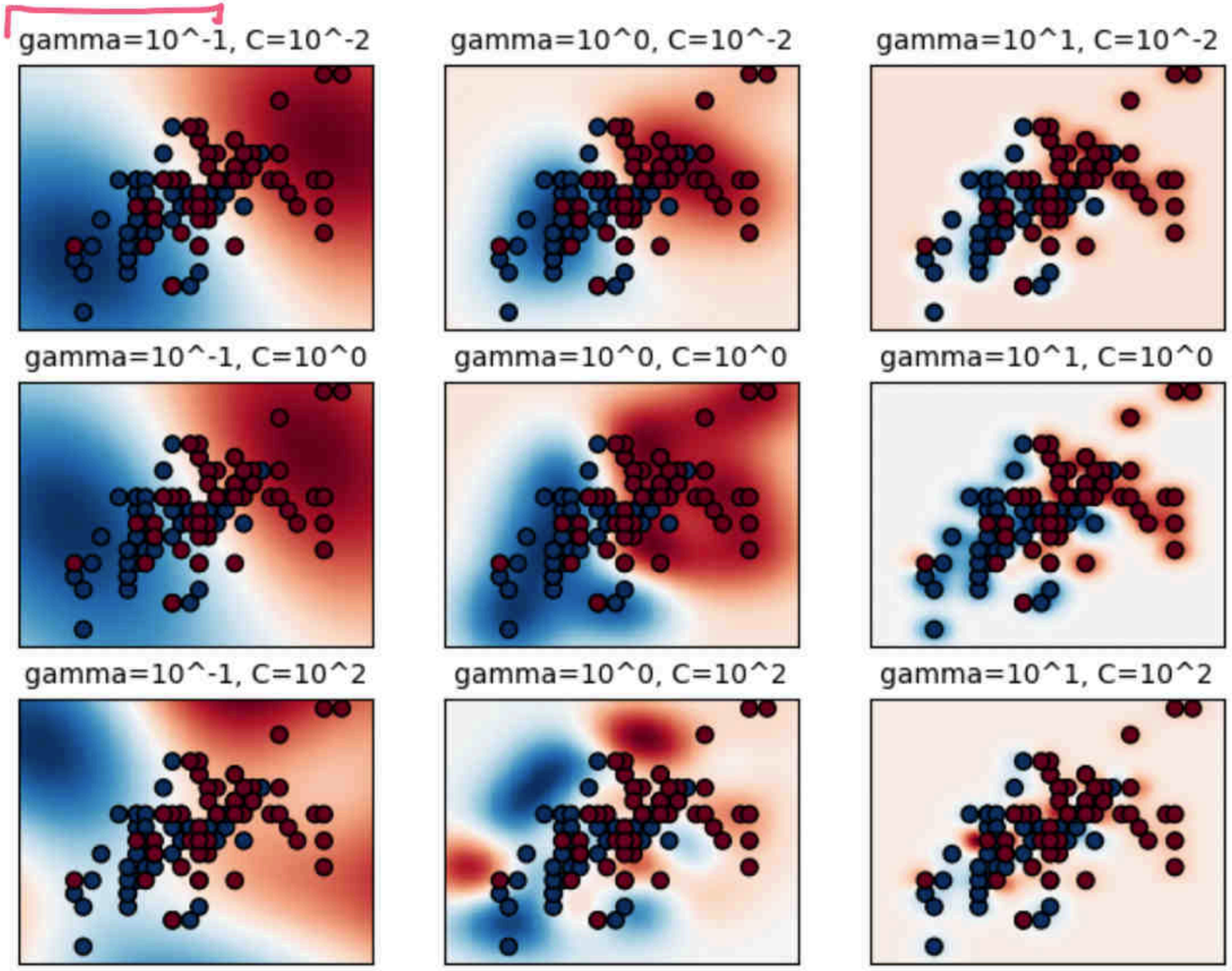
Please cite us if you use the software.

RBF SVM parameters
Load and prepare data set
Train classifiers
Visualization



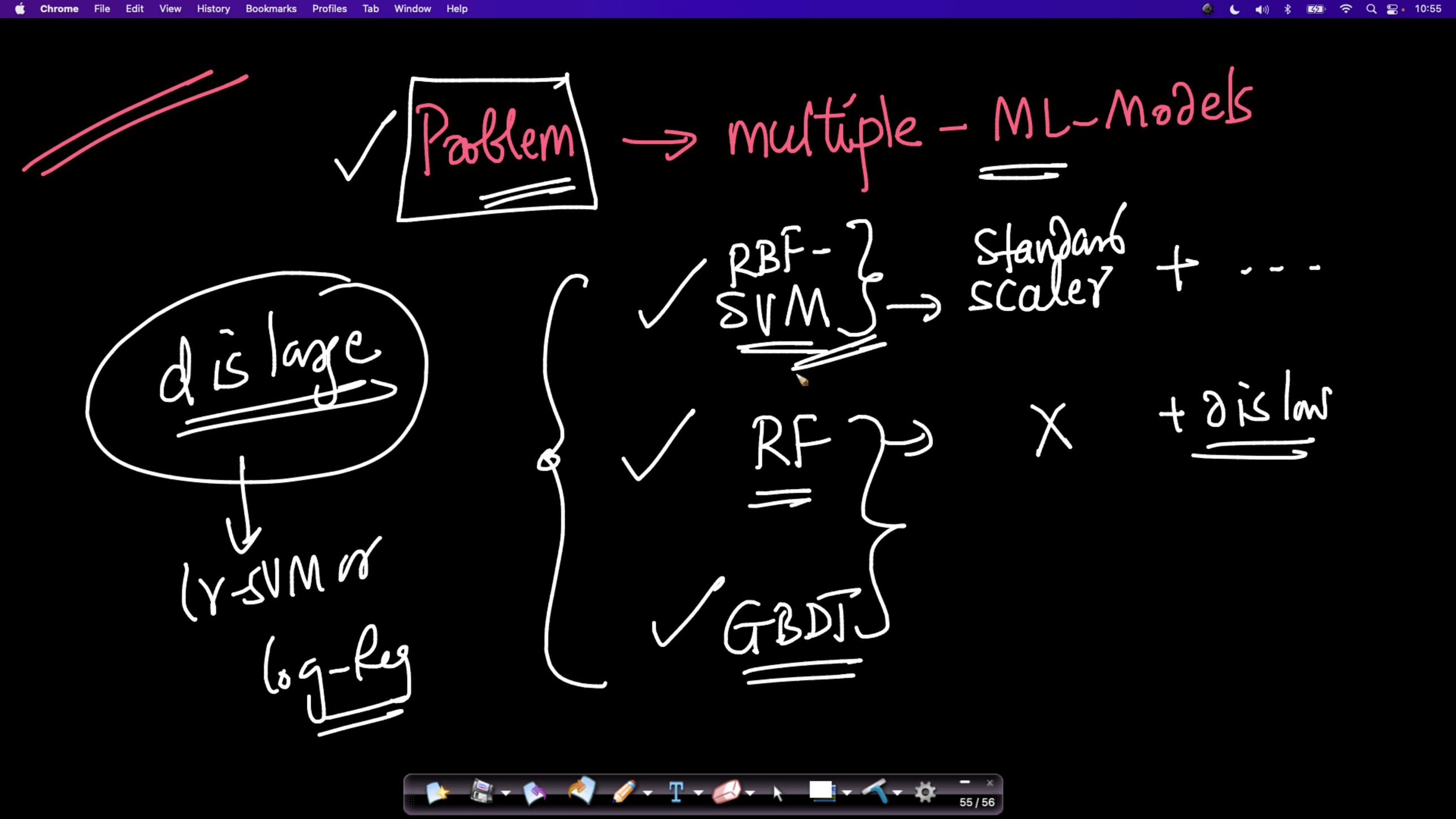
Toggle Menu

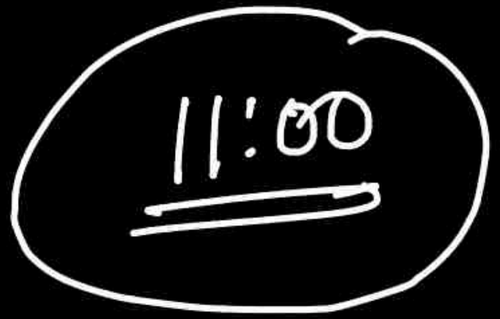
scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
RBF SVM parameters
Load and prepare data set
Train classifiers
Visualization



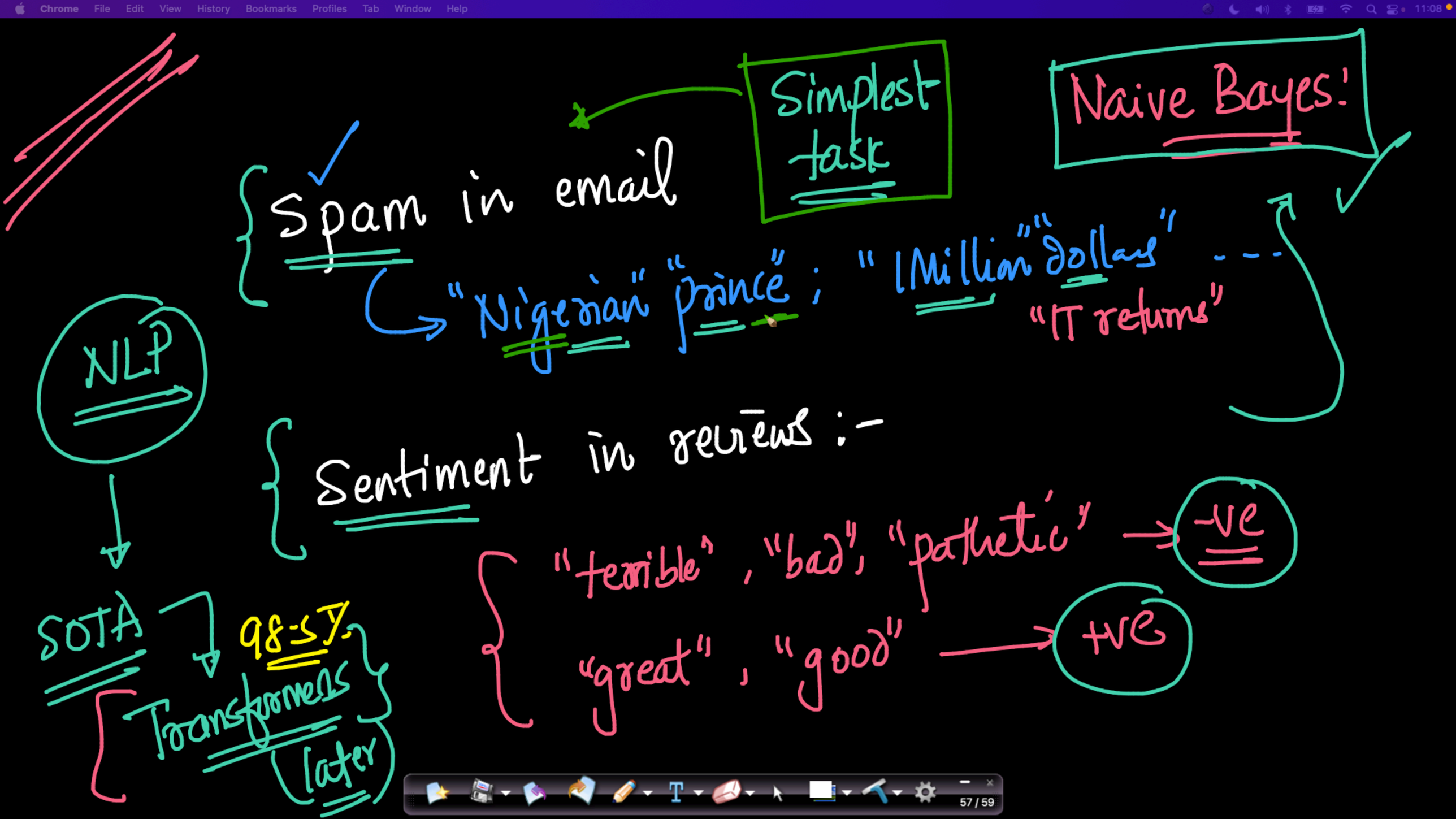
C↑
overfit

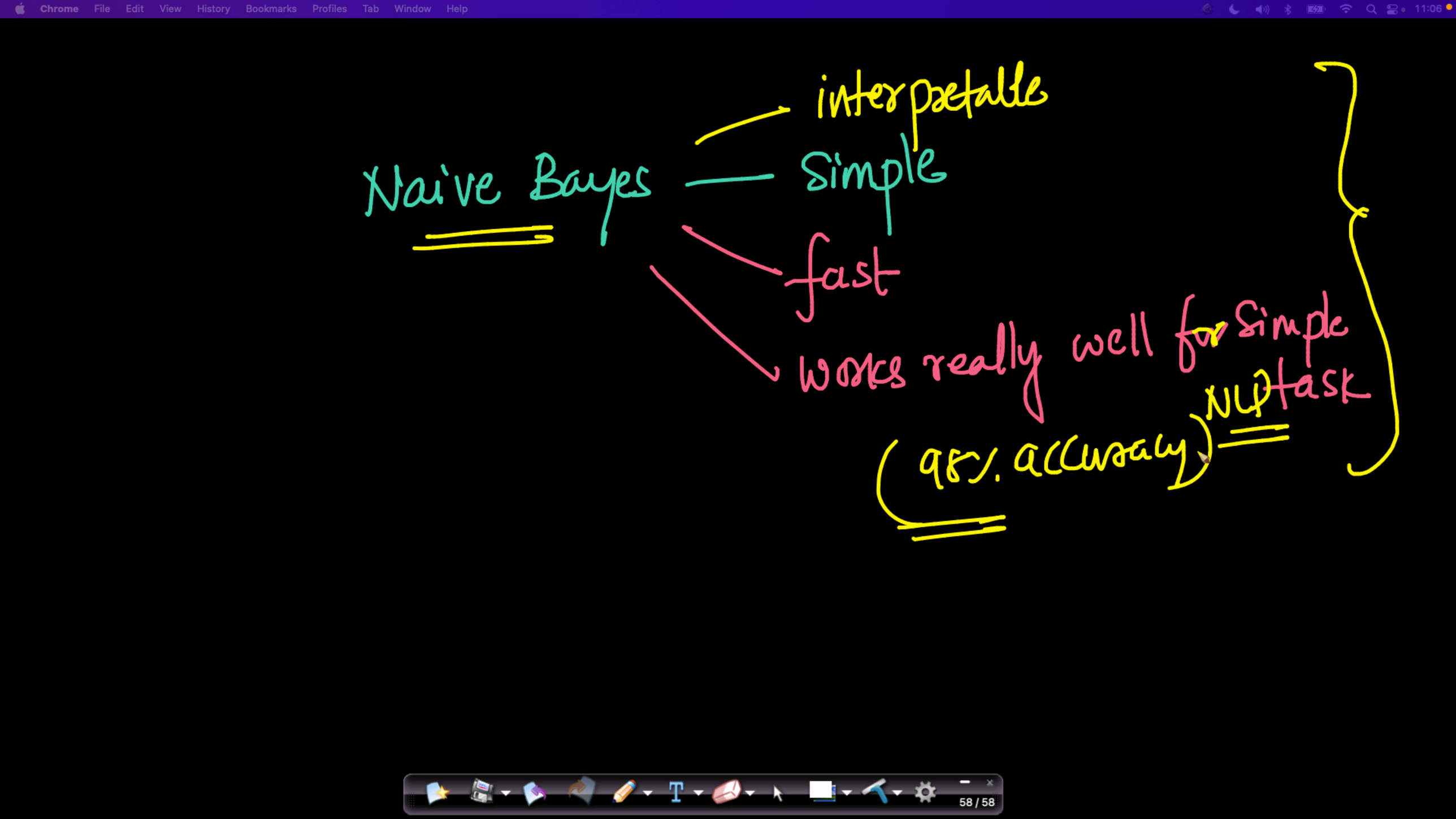
$\gamma = \frac{1}{\sigma}$


Problem → multiple - ML-Models
d is large
l1-SVM or
log-Reg
RBF-SVM
RF
GBDT
Standard scaler + ...
X
+ d is low

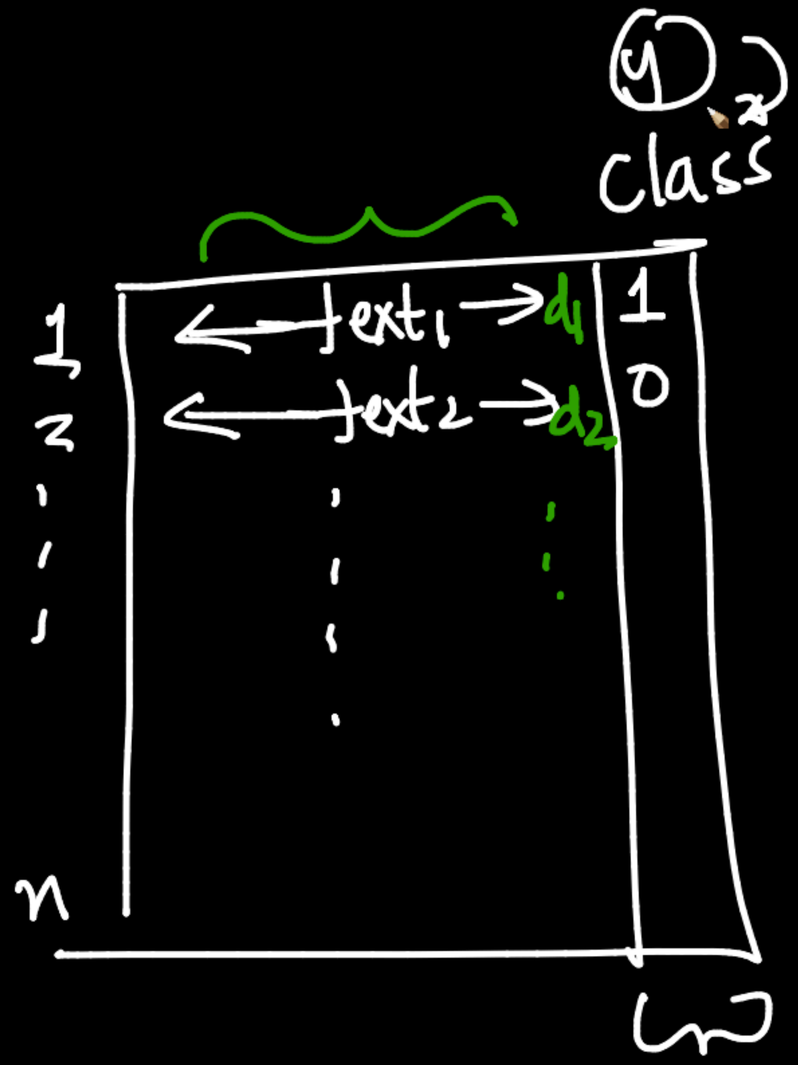


Naive Bayes


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Simplest task
Naive Bayes!
Spam in email
"Nigerian" "prince" ; "1 Million" "dollars"
"IT returns"
NLP
Sentiment in reviews :-
"terrible", "bad", "pathetic" → -ve
"great", "good" → +ve
SOTA
98-99%
Transformers
later
57 / 59


Naive Bayes
interpretable
simple
fast
works really well for simple NLP task
(98% accuracy)

Data:-



$text_i = w_1^i \; w_2^i \; w_3^i \cdots w_d^i$

text → remove stop-words ?

↳ of, the, in, I ...

$$P(y=1 \mid \text{text})$$

$$P(y=0 \mid \text{text})$$

$$P(y=1 \mid \text{text}) = \left[P\left(y=1 \mid \overbrace{w_1, w_2, w_3, \ldots w_d}^{X}\right) \right]$$

$$= \frac{P\left(y=1, \overbrace{w_1, w_2, w_3 \ldots w_d}^{X}\right)}{P(w_1, w_2, w_3 \ldots w_d)}$$

$$= \frac{P\left(\overbrace{w_1, w_2 \ldots w_d}^{X} \mid y=1\right) P(y=1)}{P(w_1, w_2 \ldots w_d)} \quad K$$

Bayes Thm

$P(A \wedge B)$
$= P(A \cap B)$
$= P(A \text{ and } B)$

$y=1$

$$P(y=1 \mid \overbrace{w_1 \dots w_3}^{x}) = \frac{P(\overbrace{w_1 w_2 \dots w_3}^{x} \mid y=1)\, P(y=1)}{\cancel{k}}$$

text

Compare

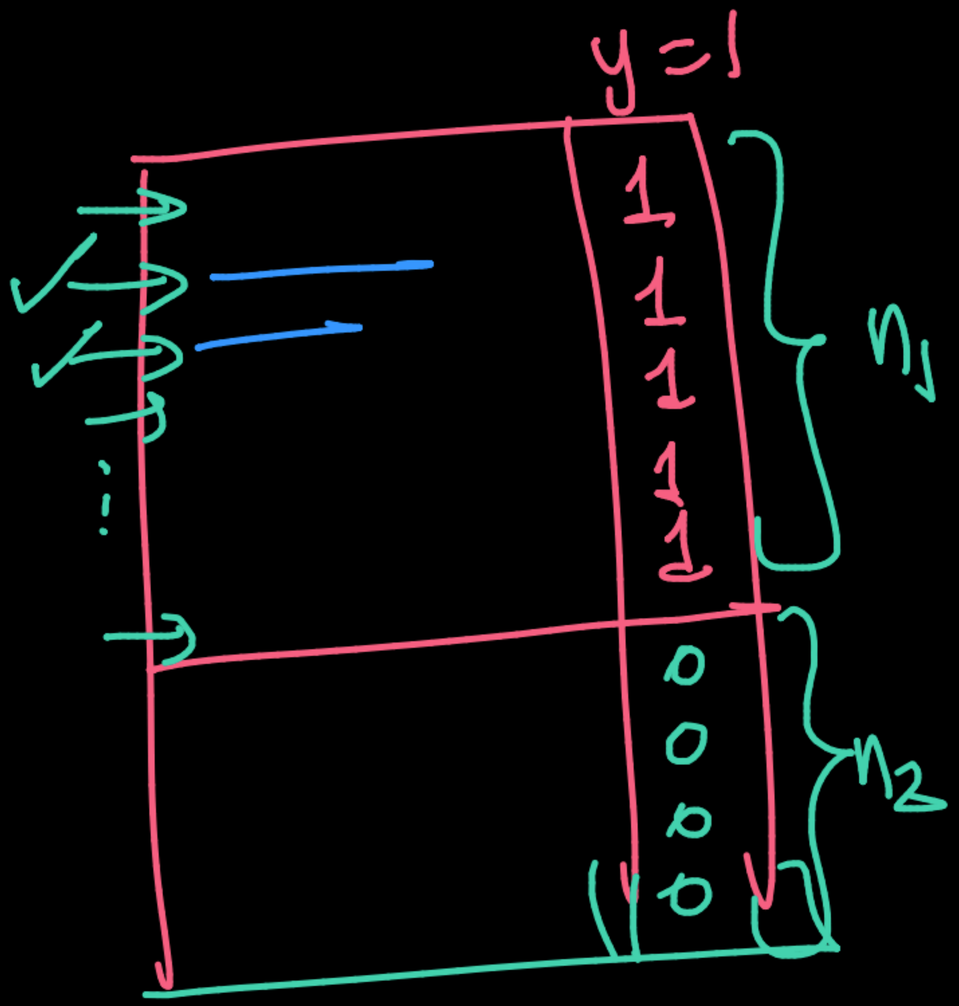
$$P(y=0 \mid \overbrace{w_1 \dots w_3}^{x}) = \frac{P(\overbrace{w_1 w_2 \dots w_3}^{x} \mid y=0)\, P(y=0)}{\cancel{k}}$$

$$P(y=1) \longrightarrow \frac{\#\text{ train pts with } y_i=1}{\text{total }\#\text{ of train pts}}$$

$$P(y=0) \longrightarrow \frac{\#\text{ train pts with } y_i=0}{\text{total }\#\text{ of train pts}}$$

$P(w_1, w_2, \ldots, w_d \mid y = 1)$

Naive



If A & B are indep → [indep ≠ mutually excl] ← ....

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

} defn

✓ { w_1 & w_2 are indep. conditional on the class

Naive assumption:

$$p(w_1, w_2, \mid y=1) \rightarrow \text{spam}$$

Nigerian prince

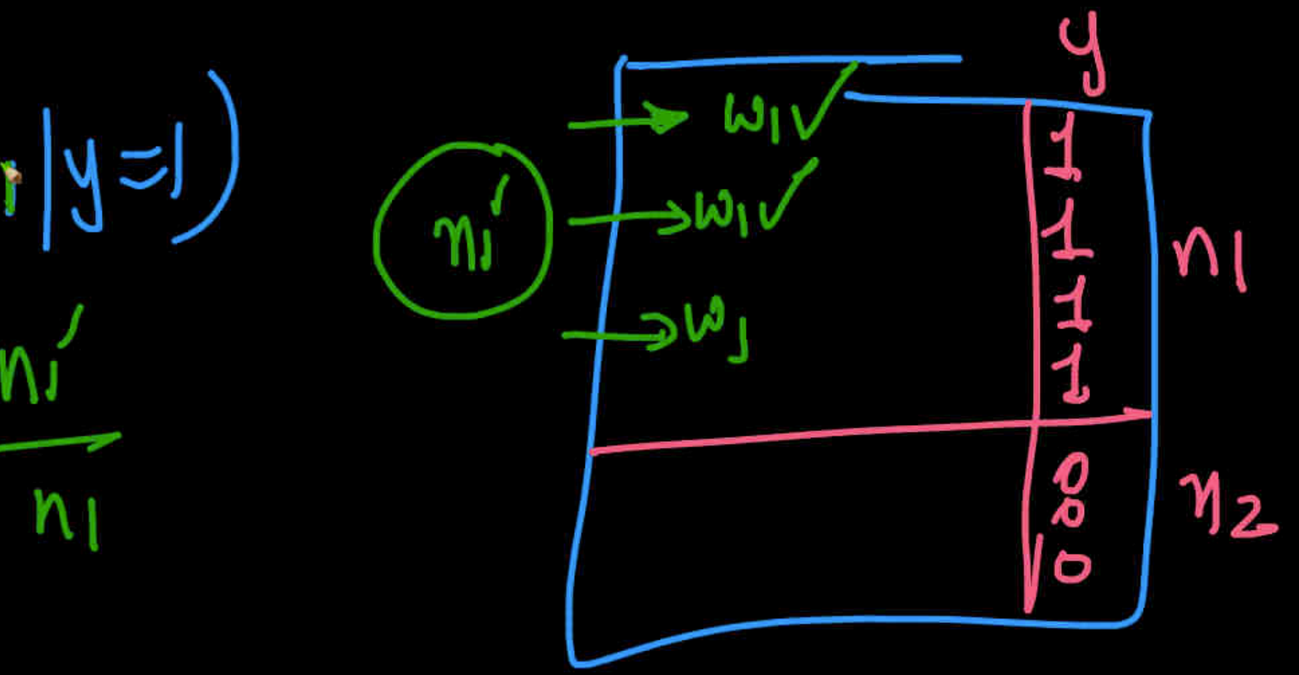
$$= p(w_1 \mid y_1) \cdot p(w_2 \mid y_1)$$

$$P(w_1 w_2 w_3 \ldots w_d \mid y = 1)$$

Naive

$$= P(w_1 \mid y = 1) \cdot P(w_2 \mid y = 1) \ldots P(w_d \mid y = 1)$$

$$P(w_i \mid y=1) = \frac{n_i'}{n_1}$$



$$p(y=1 \mid \underbrace{text}_{w_1 w_2 \cdots w_d}) = p(w_1 w_2 \cdots w_d \mid y=1)\, p(y=1)/K$$

$$= p(w_1 \mid y=1)\, p(w_2 \mid y=1) \cdots p(w_d \mid y=1) \cdot p(y=1)/K$$

$$= \prod_{i=1}^{d} p(w_i \mid y=1)\, p(y=1)/K$$

Σ: sum

Π: product

$$P(y=1 \mid \text{text}) = \prod_{i=1}^{d} p(w_i \mid y=1)\, p(y=1)$$

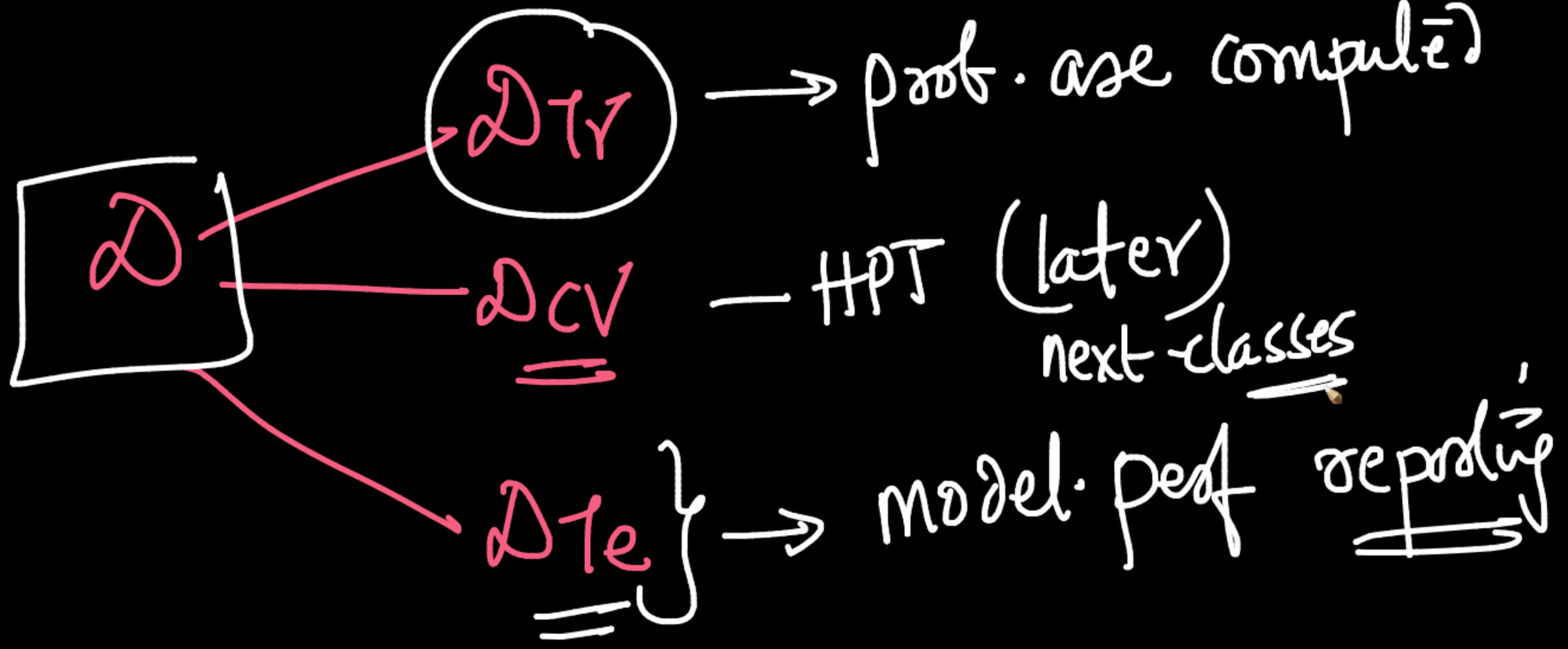
Compare

$$P(y=0 \mid \text{text}) = \prod_{i=1}^{d} p(w_i \mid y=0)\, p(y=0)$$

likelihoods

Class priors

Naive Bayes


D
Dtr
Dcv
Dte
→ prob. use compute
— HPT (later)
next classes
→ model perf reporting

log-reg :- 0 & 1 / +1, -1

SVM → +1, -1

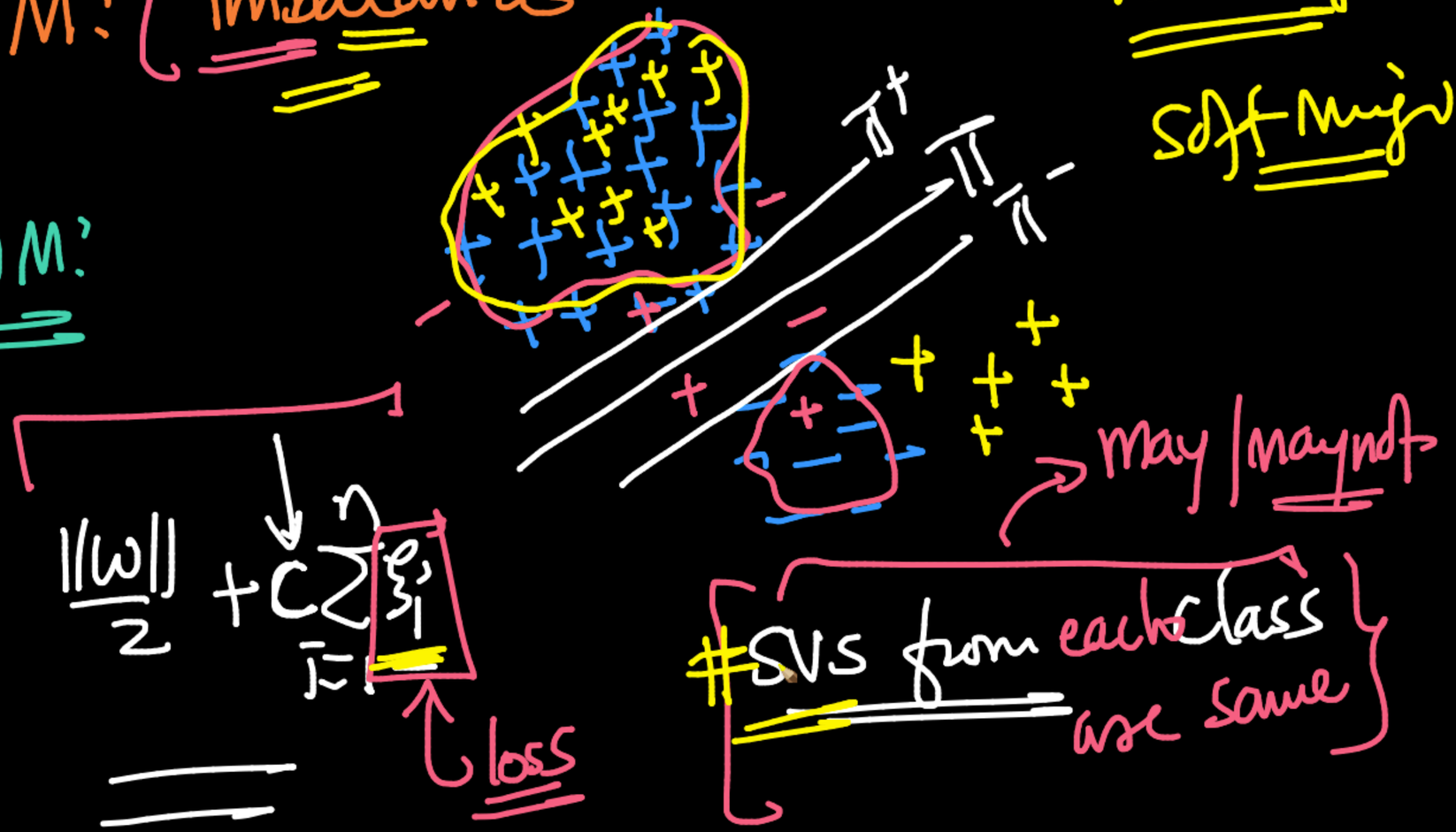
sklearn

Label-encoding

Q

SVM: { raw imbalance

Lr-SVM:



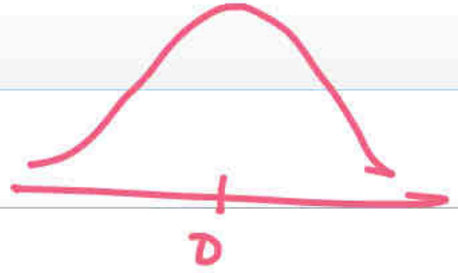
imbalance → Class weights ...
→ rebalance ...

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Radial basis function

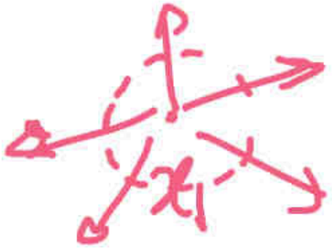


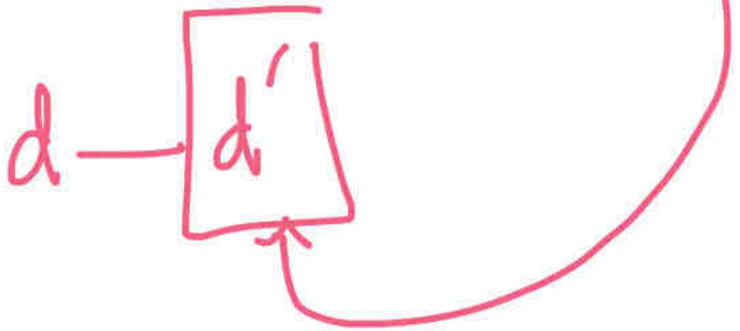
From Wikipedia, the free encyclopedia

A **radial basis function** (**RBF**) is a real-valued function φ whose value depends only on the distance between the input and some fixed point, either the origin, so that $\varphi(\mathbf{x}) = \hat{\varphi}(\|\mathbf{x}\|)$, or some other fixed point $\mathbf{c}$, called a *center*, so that $\varphi(\mathbf{x}) = \hat{\varphi}(\|\mathbf{x} - \mathbf{c}\|)$. Any function φ that satisfies the property $\varphi(\mathbf{x}) = \hat{\varphi}(\|\mathbf{x}\|)$ is a radial function. The distance is usually Euclidean distance, although other metrics are sometimes used. They are often used as a collection $\{\varphi_k\}_k$ which forms a basis for some function space of interest, hence the name.

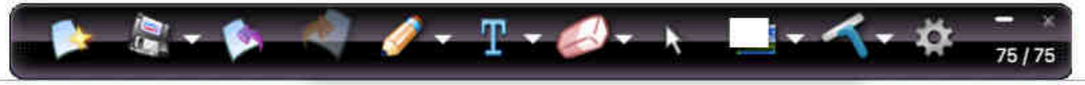
Sums of radial basis functions are typically used to approximate given functions. This approximation process can also be interpreted as a simple kind of neural network; this was the context in which they were originally applied to machine learning, in work by David Broomhead and David Lowe in 1988,[1][2] which stemmed from Michael J. D. Powell's seminal research from 1977.[3][4][5] RBFs are also used as a kernel in support vector classification.[6] The technique has proven effective and flexible enough that radial basis functions are now applied in a variety of engineering applications.[7][8]

Contents [hide]





Definition [edit]



Q.

SMOTE → D_{train} → Model

{ metric → F1-Score

D_{test} → Model → f1-score

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages

Radial basis function

From Wikipedia, the free encyclopedia

A **radial basis function** (**RBF**) is a real-valued function φ whose value depends only on the distance between the input and some fixed point, either the origin, so that $\varphi(\mathbf{x}) = \hat{\varphi}(\|\mathbf{x}\|)$, or some other fixed point $\mathbf{c}$, called a *center*, so that $\varphi(\mathbf{x}) = \hat{\varphi}(\|\mathbf{x} - \mathbf{c}\|)$. Any function φ that satisfies the property $\varphi(\mathbf{x}) = \hat{\varphi}(\|\mathbf{x}\|)$ is a radial function. The distance is usually Euclidean distance, although other metrics are sometimes used. They are often used as a collection $\{\varphi_k\}_k$ which forms a basis for some function space of interest, hence the name.

Sums of radial basis functions are typically used to approximate given functions. This approximation process can also be interpreted as a simple kind of neural network; this was the context in which they were originally applied to machine learning, in work by David Broomhead and David Lowe in 1988,[1][2] which stemmed from Michael J. D. Powell's seminal research from 1977.[3][4][5] RBFs are also used as a kernel in support vector classification.[6] The technique has proven effective and flexible enough that radial basis functions are now applied in a variety of engineering applications.[7][8]

Contents [hide]