Support Vector Machines & Kernels
Lecture 6

David Sontag
New York University

Slides adapted from Luke Zettlemoyer, Carlos Guestrin, and Vibhav Gogate

SVMs in the dual

Primal:

$$\text{minimize}_{\mathbf{w},b} \quad \tfrac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \;\; \forall j$$
$$\xi_j \geq 0, \;\; \forall j$$

Solve for **w**, b:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $C > \alpha_k > 0$

Dual:

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \tfrac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \underbrace{\mathbf{x}_i \mathbf{x}_j}_{\text{dot product}}$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

The dual is also a quadratic program, and can be efficiently solved to optimality

Support vectors

- **Complementary slackness** conditions:

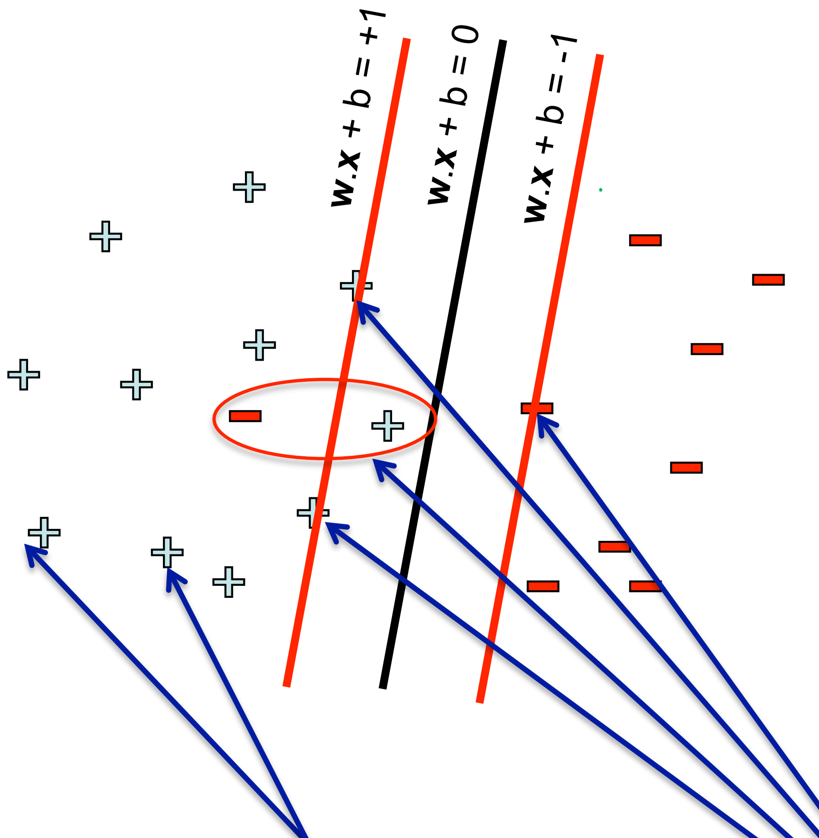
$$\alpha_j^* \left[y_j(\vec{w}^* \cdot \vec{x}_j + b) - 1 + \xi_j \right] = 0 \implies \alpha_j^* = 0 \ \vee \ y_j(\vec{w}^* \cdot \vec{x}_j + b) = 1 - \xi_j$$

$$\implies \alpha_j^* = 0 \ \vee \ y_j(\vec{w}^* \cdot \vec{x}_j + b) \leq 1$$

- **Support vectors**: points x_j such that $y_j(\vec{w}^* \cdot \vec{x}_j + b) \leq 1$ (includes all j such that $\alpha_j^* > 0$, but also additional points where $\alpha_j^* = 0 \wedge y_j(\vec{w}^* \cdot \vec{x}_j + b) = 1$)

- Note: the SVM dual solution may not be unique!

Dual SVM interpretation: Sparsity



$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

Final solution tends to be sparse

- α_j=0 for most j
- don't need to store these points to compute w or make predictions

Non-support Vectors:
- α_j=0
- moving them will not change w

Support Vectors:
- $\alpha_j \geq 0$

Classification rule using dual solution

$$y \leftarrow \mathrm{sign}(\vec{w} \cdot \vec{x} + b)$$

Using dual solution

$$y \leftarrow \mathrm{sign}\left[\sum_i \alpha_i y_i \underbrace{(\vec{x}_i \cdot \vec{x})} + b\right]$$

dot product of feature vectors of new example with support vectors

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $C > \alpha_k > 0$

SVM with kernels

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

- Never compute features explicitly!!!
  – Compute dot products in closed form
- $O(n^2)$ time in size of dataset to compute objective
  – much work on speeding up

Predict with:

$$y \leftarrow \text{sign}\left[\sum_i \alpha_i y_i K(x_i, x) + b\right]$$

Efficient dot-product of polynomials

Polynomials of degree exactly *d*

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$$

d=2

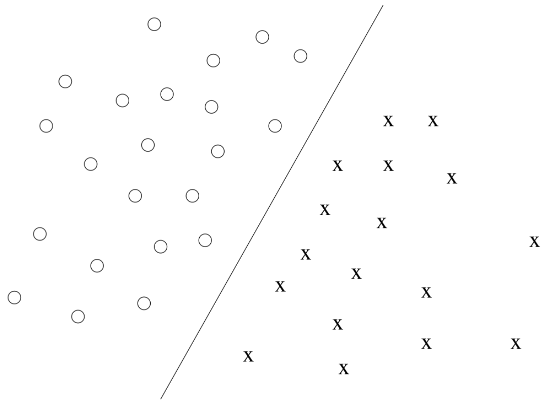
$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} . \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} \begin{aligned} &= u_1^2 v_1^2 + 2 u_1 v_1 u_2 v_2 + u_2^2 v_2^2 \\ &= (u_1 v_1 + u_2 v_2)^2 \\ &= (u.v)^2 \end{aligned}$$

For any *d (we will skip proof):*

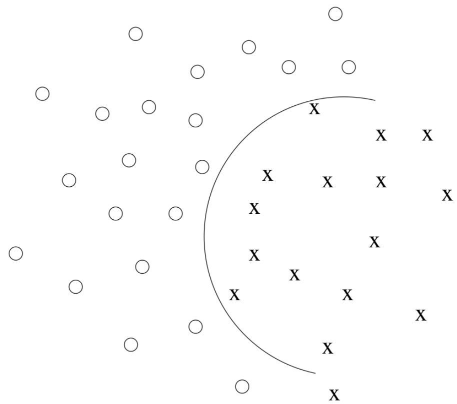
$$\phi(u).\phi(v) = (u.v)^d$$

- Cool! Taking a dot product and exponentiating gives same results as mapping into high dimensional space and then taking dot product

Quadratic kernel



Linear separator in the feature ϕ-space



Non-linear separator in the original $\mathbf{x}$-space

[Tommi Jaakkola]

Quadratic kernel

$$\begin{aligned}
k(\mathbf{x},\mathbf{z}) &= (\mathbf{x}^T\mathbf{z}+c)^2 = \left(\sum_{j=1}^n x^{(j)}z^{(j)}+c\right)\left(\sum_{\ell=1}^n x^{(\ell)}z^{(\ell)}+c\right) \\
&= \sum_{j=1}^n\sum_{\ell=1}^n x^{(j)}x^{(\ell)}z^{(j)}z^{(\ell)} + 2c\sum_{j=1}^n x^{(j)}z^{(j)} + c^2 \\
&= \sum_{j,\ell=1}^n (x^{(j)}x^{(\ell)})(z^{(j)}z^{(\ell)}) + \sum_{j=1}^n (\sqrt{2c}x^{(j)})(\sqrt{2c}z^{(j)}) + c^2,
\end{aligned}$$

Feature mapping given by:

$$\mathbf{\Phi}(\mathbf{x}) = [x^{(1)2}, x^{(1)}x^{(2)}, ..., x^{(3)2}, \sqrt{2c}x^{(1)}, \sqrt{2c}x^{(2)}, \sqrt{2c}x^{(3)}, c]$$

[Cynthia Rudin]

Common kernels

- Polynomials of degree exactly *d*

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to *d*

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

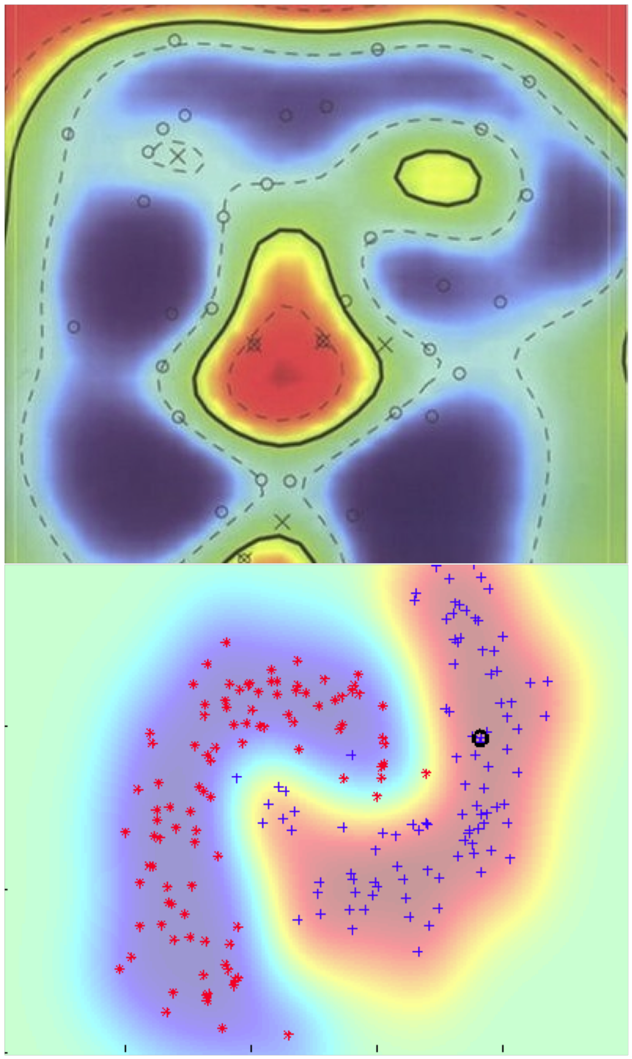
- Gaussian kernels

$$K(\vec{u}, \vec{v}) = \exp\left(-\frac{||\vec{u} - \vec{v}||_2^2}{2\sigma^2}\right)$$

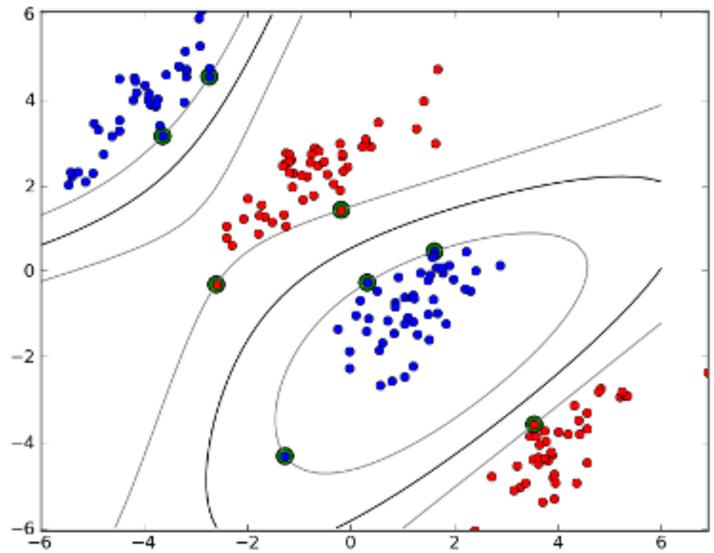
Euclidean distance, squared

- And many others: very active area of research!
(e.g., structured kernels that use dynamic programming to evaluate)

Gaussian kernel



[Cynthia Rudin]



[mblondel.org]

Kernel algebra

kernel composition	**feature composition**
a) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v}) + k_b(\mathbf{x},\mathbf{v})$	$\boldsymbol{\phi}(\mathbf{x}) = (\boldsymbol{\phi}_a(\mathbf{x}), \boldsymbol{\phi}_b(\mathbf{x})),$
b) $k(\mathbf{x},\mathbf{v}) = f k_a(\mathbf{x},\mathbf{v}),\ f > 0$	$\boldsymbol{\phi}(\mathbf{x}) = \sqrt{f}\boldsymbol{\phi}_a(\mathbf{x})$
c) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v}) k_b(\mathbf{x},\mathbf{v})$	$\phi_m(\mathbf{x}) = \phi_{ai}(\mathbf{x})\phi_{bj}(\mathbf{x})$
d) $k(\mathbf{x},\mathbf{v}) = \mathbf{x}^T A \mathbf{v}$, A positive semi-definite	$\boldsymbol{\phi}(\mathbf{x}) = L^T\mathbf{x}$, where $A = LL^T$.
e) $k(\mathbf{x},\mathbf{v}) = f(\mathbf{x}) f(\mathbf{v}) k_a(\mathbf{x},\mathbf{v})$	$\phi(\mathbf{x}) = f(\mathbf{x})\phi_a(\mathbf{x})$

Q: How would you prove that the “Gaussian kernel” is a valid kernel?
A: Expand the Euclidean norm as follows:

$$\exp\left(-\frac{\|\vec{u}-\vec{v}\|_2^2}{2\sigma^2}\right) = \exp\left(-\frac{\|\vec{u}\|_2^2}{2\sigma^2}\right)\exp\left(-\frac{\|\vec{v}\|_2^2}{2\sigma^2}\right)\exp\left(\frac{\vec{u}\cdot\vec{v}}{\sigma^2}\right)$$

Then, apply (e) from above

To see that this is a kernel, use the Taylor series expansion of the exponential, together with repeated application of (a), (b), and (c):

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

The feature mapping is infinite dimensional!

[Justin Domke]

Overfitting?

- Huge feature space with kernels: should we worry about overfitting?
  - SVM objective seeks a solution with large **margin**
    - Theory says that large margin leads to good generalization (we will see this in a couple of lectures)
  - But everything overfits sometimes!!!
  - Can control by:
    - Setting C
    - Choosing a better Kernel
    - Varying parameters of the Kernel (width of Gaussian, etc.)

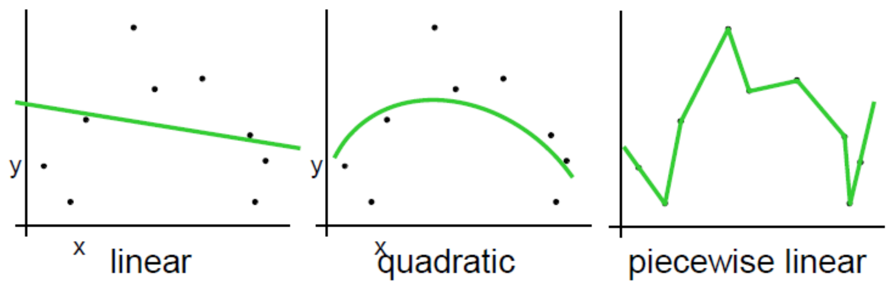
Software

- SVM^{light}: one of the most widely used SVM packages. Fast optimization, can handle very large datasets, C++ code.
- LIBSVM
- Both of these handle multi-class, weighted SVM for unbalanced data, etc.
- There are several new approaches to solving the SVM objective that can be much faster:
  - Stochastic subgradient method (discussed in a few lectures)
  - Distributed computation (also to be discussed)
- See http://mloss.org, "machine learning open source software"

Machine learning methodology: Cross Validation

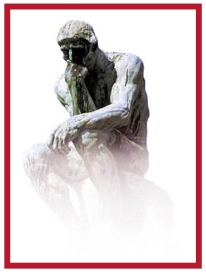
Choosing among several hypotheses

- Suppose you are considering between several different hypotheses, e.g.



regression

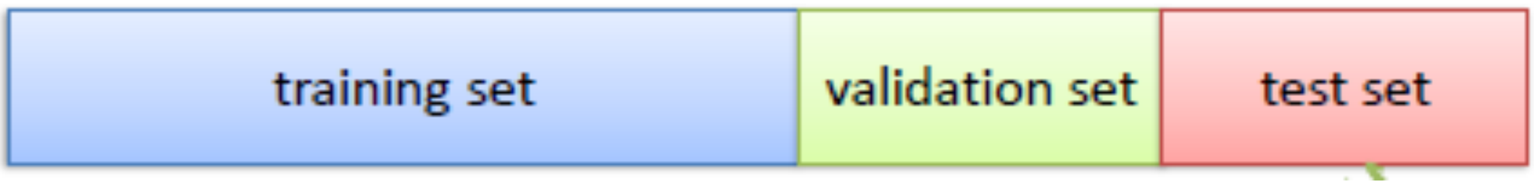
- For the SVM, we get one linear classifier for each choice of the regularization parameter C



- How do you choose between them?

General strategy

Split the data up into three parts:

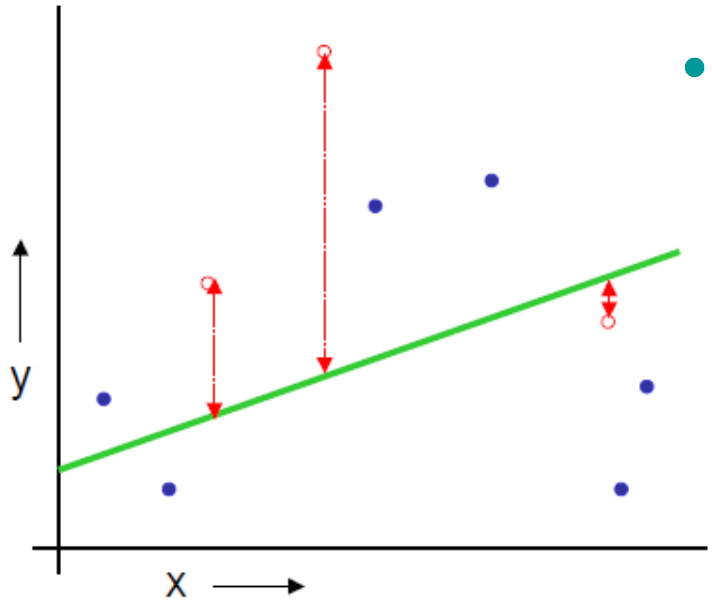


Assumes that the available data is randomly allocated to these three, e.g. 60/20/20.

Typical approach

- Learn a model from the training set (e.g., fix a C and learn the SVM)

- Estimate your future performance with the validation data

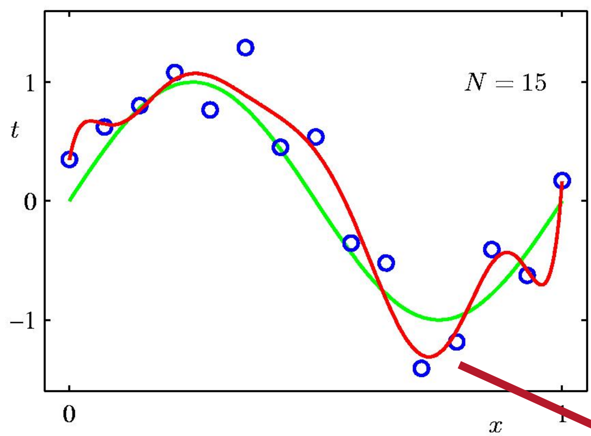


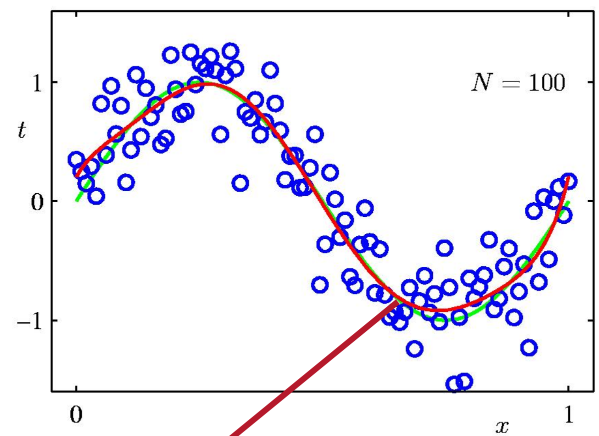
This the model you learned.

More data is better

With more data you can learn better

Blue: Observed data
Red: Predicted curve
True: Green true distribution





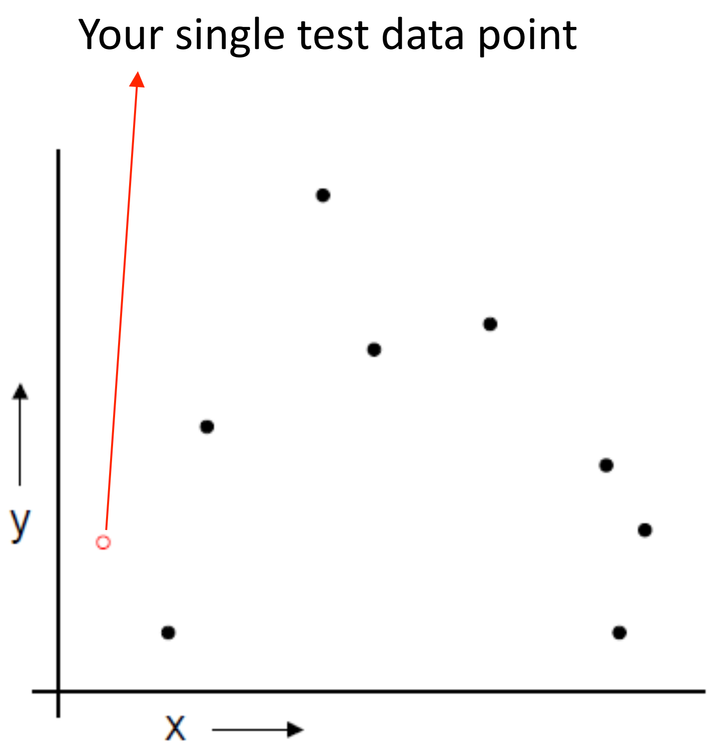
Compare the predicted curves

Cross Validation

Recycle the data!

Use (almost) all of this for training:

LOOCV (Leave-one-out Cross Validation)



Lets say we have N data points
k indices the data points, i.e. k=1...N

Let (x_k, y_k) be the k^{th} example

Temporarily remove (x_k, y_k) from the dataset

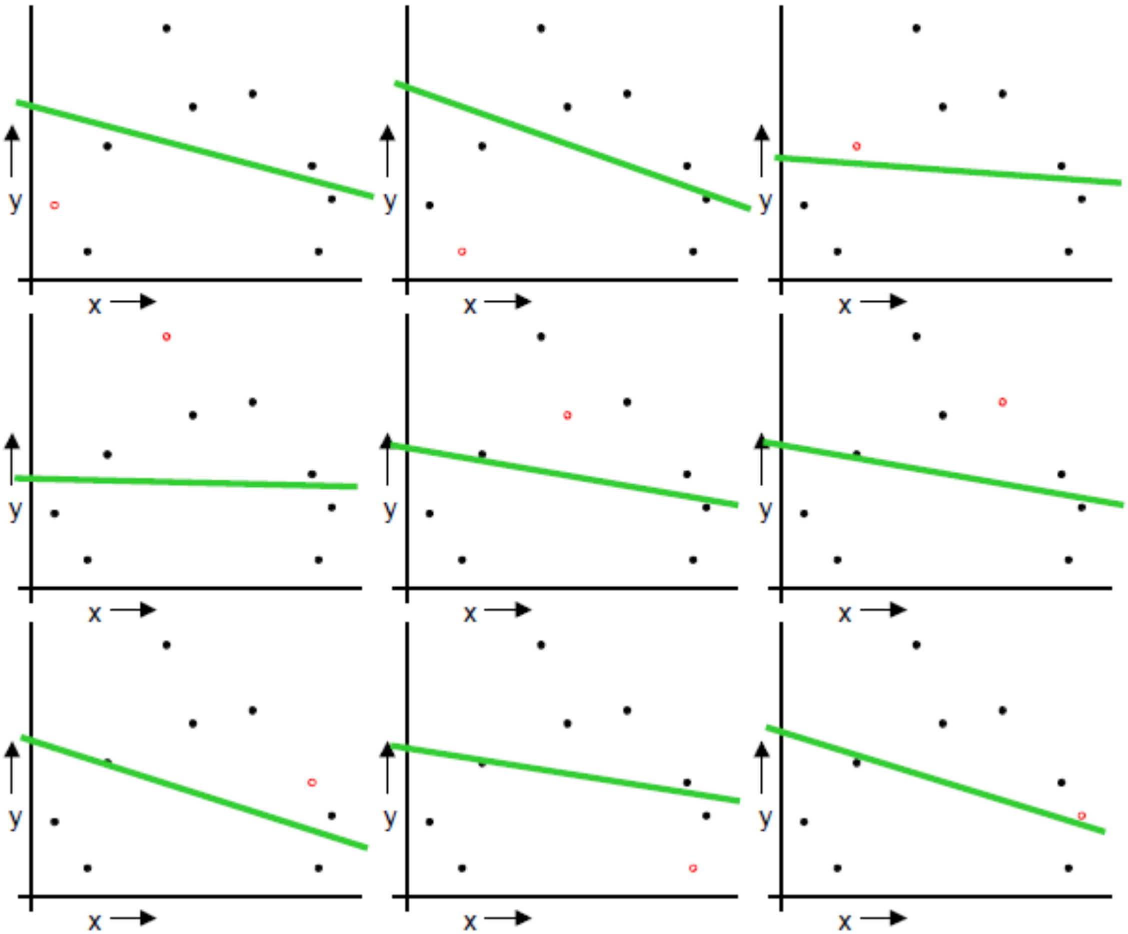
Train on the remaining N-1 data points

Test your error on (x_k, y_k)

Do this for each k=1..N and report the average error

Once the best parameters (e.g., choice of C for the SVM) are found, re-train using all of the training data

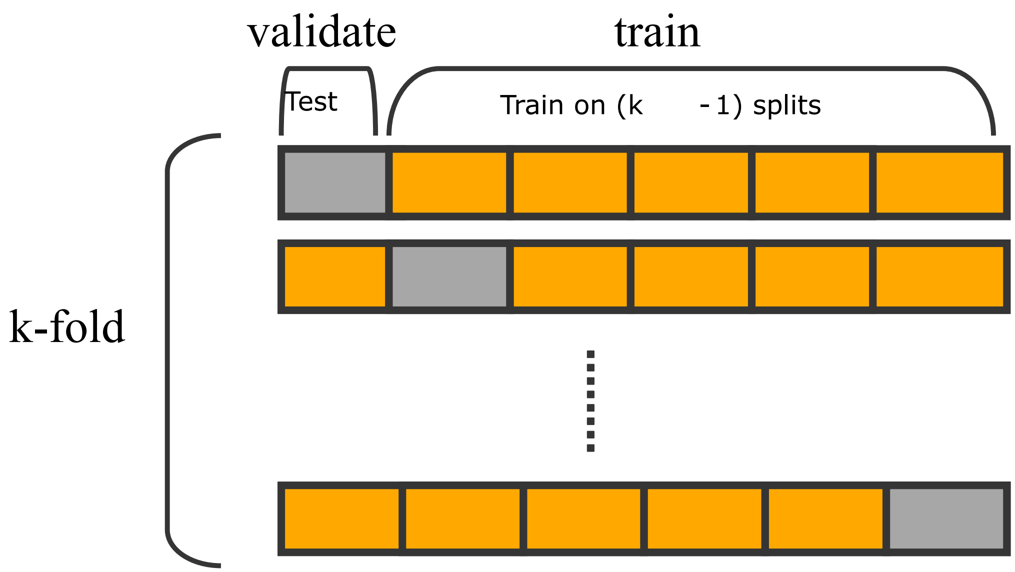
LOOCV (Leave-one-out Cross Validation)



There are N data points.
Repeat learning N times.

Notice the test data (shown in red) is changing each time

K-fold cross validation



In 3 fold cross validation, there are 3 runs.
In 5 fold cross validation, there are 5 runs.
In 10 fold cross validation, there are 10 runs.

the error is averaged over all runs