Lecture
Support Vector Machines cont

PROF. LINDA SELLIE

SOME SLIDES FROM PROF. RANGAN

The “Kernel Trick”

- For some transformations, we can compute
  $K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Phi(\mathbf{x}')$
  without transforming the features. Instead, we perform operations in the original feature space to compute the value of $K(\mathbf{x}, \mathbf{x}')$.
- In class, we will discuss two different types of transformations:
  1. Polynomial transformation
  2. RBF

SVM and feature transformation

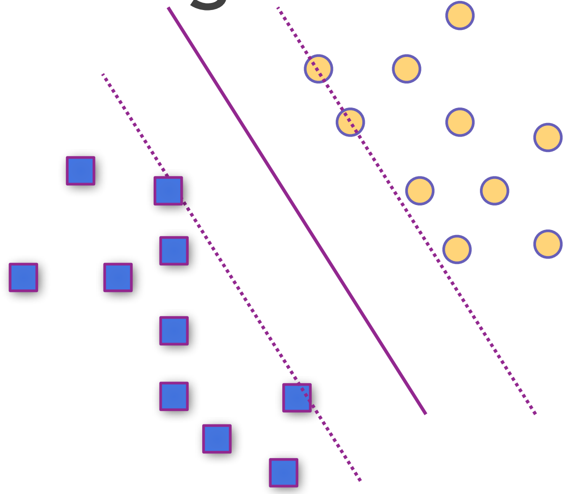
- Feature transformation
- Support vectors
- The kernel trick
  (polynomial and RBF)

Fundamental Insight

SVM's are typically implemented using the **dual form.** The kernel trick is especially efficient in the dual form. The dual form is outside the scope of the class.

We expressed $\mathbf{w}$ as a linear combination of the training examples!

$$\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \Phi(\mathbf{x}^{(i)})$$



Keep track of a single variable per data point $\alpha^{(i)}$.

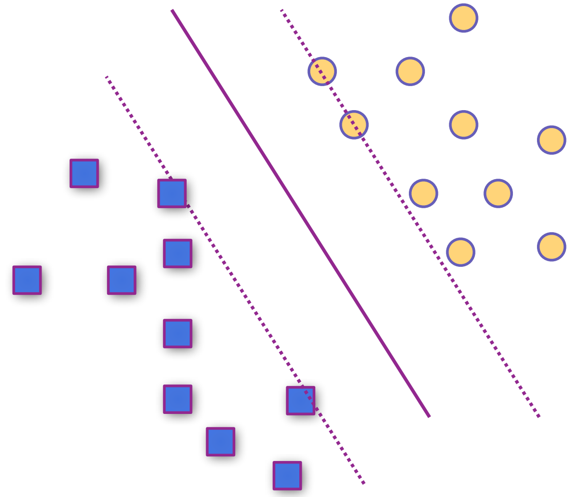
Predict for example $\mathbf{x}$: $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}\left(\left(\underbrace{\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \Phi(\mathbf{x}^{(i)})^T}_{\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \Phi(\mathbf{x}^{(i)})} \Phi(\mathbf{x})\right) + w_0\right)$

SVM support vectors

The optimal $\mathbf{w}$ is a linear combination of the training examples whose $\alpha^{(i)} \neq 0$ (i.e. examples on the margin or inside the margin)

$$\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$$

$\alpha^{(i)} = 0$ if $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) > 1$
All points correctly classified outside the margin



These examples are called the support vectors

Example (with rounded numbers….)

$$\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$$

Given the set of training examples
((1,2.5), 1), ((0, 0.75), -1), ((1, 1), -1), ((2,2), 1), ((3,1), 1), ((2,3), 1), …

α^T = [0.9, 0, 1.4, 0, 0.5, 0, …, 0]

What is $\mathbf{w}$?

$$\mathbf{w} = 0.9(1)\begin{bmatrix}1\\2.5\end{bmatrix} + 1.4(-1)\begin{bmatrix}1\\1\end{bmatrix} + 0.5(1)\begin{bmatrix}3\\1\end{bmatrix} = 0.9\begin{bmatrix}1\\2.5\end{bmatrix} + -1.4\begin{bmatrix}1\\1\end{bmatrix} + 0.5\begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}1\\1.35\end{bmatrix}$$

What is w_0?

No examples are inside the margin. Thus every support vector has a functional margin of one.

$$(1)\left(w_0 + \mathbf{w}^T\begin{bmatrix}3\\1\end{bmatrix}\right) = 1$$

$$w_0 + 4.35 = 1$$

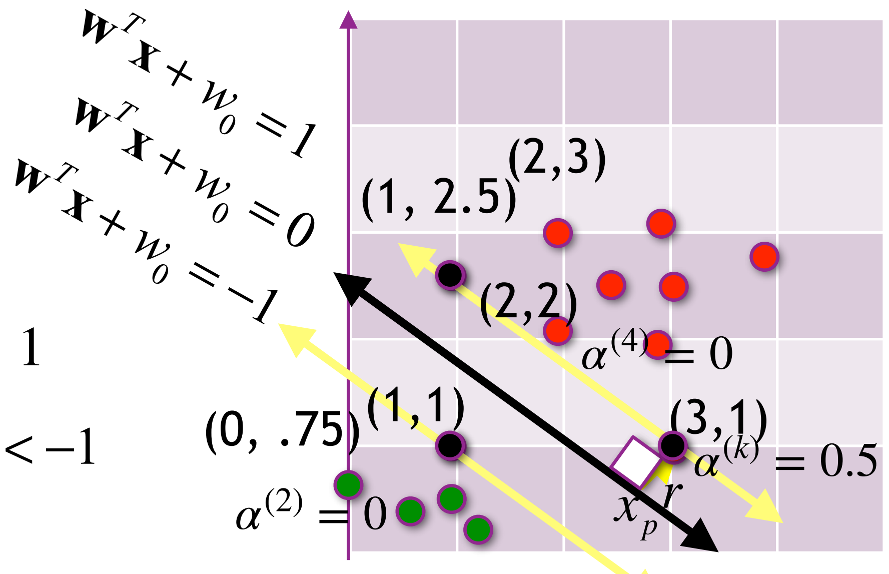
$$w_0 = -3.35$$

Note… I rounded these numbers, so… (If I hadn't rounded $w_2 = 1.33, w_0 = -3.33$)

$$\mathbf{w}^T(3,1)^T + w_0 = 1$$

$$\mathbf{w}^T(0,0.75)^T + w_0 < -1$$

$$\mathbf{w}^T(2,3)^T + w_0 > 1$$


$\mathbf{w}^T\mathbf{x} + w_0 = 1$
$\mathbf{w}^T\mathbf{x} + w_0 = 0$
$\mathbf{w}^T\mathbf{x} + w_0 = -1$
(2,3)
(1, 2.5)
(2,2)
$\alpha^{(4)} = 0$
(0, .75)
(1,1)
(3,1)
$\alpha^{(k)} = 0.5$
$\alpha^{(2)} = 0$
x_p
r

Prediction of $\begin{bmatrix}1\\0\end{bmatrix}$

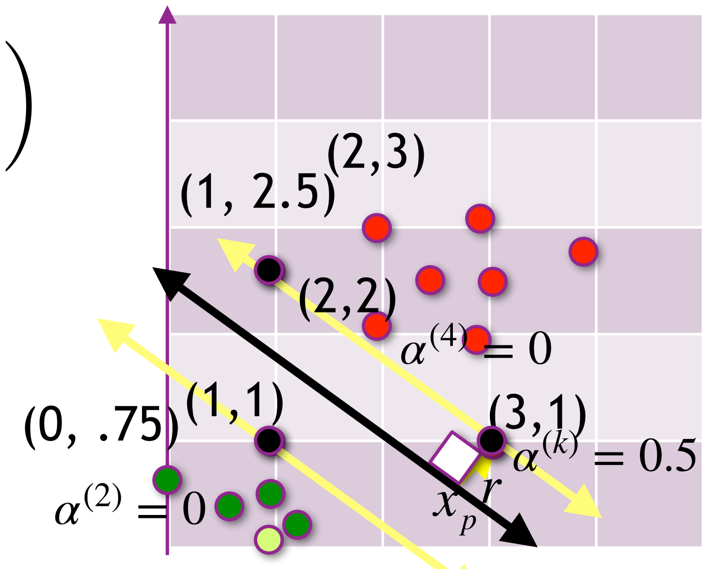
$$\mathbf{w} = \begin{bmatrix}1\\1.35\end{bmatrix} = 0.9\begin{bmatrix}1\\1.25\end{bmatrix} + -1.4\begin{bmatrix}1\\1\end{bmatrix} + 0.5\begin{bmatrix}3\\1\end{bmatrix}, \quad w_0 = -3.35$$

$$\text{sign}\left(\mathbf{w}^T\begin{bmatrix}1\\0\end{bmatrix} + w_0\right) = \text{sign}\left(\begin{bmatrix}1\\1.35\end{bmatrix}^T\begin{bmatrix}1\\0\end{bmatrix} - 3.35\right) = 1 - 3.35 = \text{sign}(-2.25) = -1$$

$$= \text{sign}\left(\left(0.9\begin{bmatrix}1\\1.25\end{bmatrix} + -1.4\begin{bmatrix}1\\1\end{bmatrix} + 0.5\begin{bmatrix}3\\1\end{bmatrix}\right)^T\begin{bmatrix}1\\0\end{bmatrix} - 3.35\right)$$

$$= \text{sign}\left(0.9\underbrace{K\left(\begin{bmatrix}1\\1.25\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right)}_{K(\mathbf{x}^{(1)}, \mathbf{x})} + -1.4\underbrace{K\left(\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right)}_{K(\mathbf{x}^{(3)}, \mathbf{x})} + 0.5\underbrace{K\left(\begin{bmatrix}3\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right)}_{K(\mathbf{x}^{(5)}, \mathbf{x})} - 3.35\right)$$

Linear kernel



How did this save us any time?

Avoiding paying the time/space cost of feature transformation!

Key takeaway

If we can efficiently compute the inner product, we can *implicitly* work in high dimensional space

$$K(\mathbf{x}^{(i)}, \mathbf{x}) = \Phi(\mathbf{x}^{(i)})^T \Phi(\mathbf{x})$$

SVM and feature transformation

- Feature transformation
- Support vectors
- Kernel trick (polynomial and RBF)

Standard Choices of Kernel functions

- Dot product (no transformation):

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \mathbf{x}^{(i)T}\mathbf{x}^{(j)}$$

- Polynomial kernels:

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (1 + \mathbf{x}^{(i)T}\mathbf{x}^{(j)})^d$$

- Radial basis function:

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\gamma\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2\right)$$

Polynomial transformation of degree exactly 2

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$$

$$K(\mathbf{x}^T\mathbf{x}') = \Phi(\mathbf{x})^T\Phi(\mathbf{x}') = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1x_2 \end{bmatrix} \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2}x_1'x_2' \end{bmatrix}$$

$$= x_1^2 \cdot x_1'^2 + x_2^2 \cdot x_2'^2 + \sqrt{2}x_1x_2 \cdot \sqrt{2}x_1'x_2' = \left([x_1 \quad x_2] \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} \right)^2 = (\mathbf{x}^T\mathbf{x}')^2$$

If d = 200, then $\tilde{d} = d + \dfrac{d(d-1)}{2}$ = 20100

For 200 features, I only need to perform 201 multiplications (and 199 additions)

Polynomial kernel

$$\Phi_2(\mathbf{x})^T = [\underline{1}, \underline{x_1, x_2, \ldots, x_d}, \underline{x_1^2, x_2^2, \ldots, x_d^2, x_1 x_2, \ldots, x_{d-1} x_d}]$$

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}')$$

If d=200 then in z-space we have a feature vector of size approx 20,000

Kernel Motivation

$$g(\mathbf{x}) = \operatorname{sign}\left(\sum_{i \in I} \alpha^{(i)} y^{(i)} (\mathbf{x}^{(i)T} \mathbf{x}) + w_0 \right)$$

$$\mathbf{x} = [x_1, x_2]^T$$

$$\Phi_2 : R^2 \rightarrow R^6$$

$$\Phi_2(\mathbf{x}) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

$$\Phi_2(-0.75, -0.25) = (1, -0.75, -0.25, (-0.75) \cdot (-0.25), (-0.75)^2, (-0.25)^2)$$

$$\Phi_2(-1, 1) = (1, -1, 1, -1 \cdot 1, (-1)^2, 1^2)$$

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2] \cdot [1, x_1', x_2', x_1' x_2', x_1'^2, x_2'^2]$$

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') = 1 + x_1 x_1' + x_2 x_2' + x_1 x_2 x_1' x_2' + x_1{}^2 x_1'^2 + x_2{}^2 x_2'^2$$

$$\Phi_2(-0.75, -0.25) \cdot \Phi_2(-1, 1) = 1 + 0.75 - 0.25 + 0.75 \cdot (-0.25) + (-0.75)^2 + (-0.25)$$

Kernel motivation

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)} y^{(i)} (\mathbf{x}^{(i)T} \mathbf{x}) + w_0 \right)$$

But if we use a different (but similar) transformation:

$$\Phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2]^T$$

$$\Phi(-0.75, -0.25) = \left(1, \sqrt{2} \cdot (-0.75), \sqrt{2} \cdot (-0.25), \sqrt{2}(-0.75) \cdot (-0.25), (-0.75)^2, (-0.25)^2\right)$$

$$\Phi(-1, 1) = (1, -\sqrt{2}, \sqrt{2}, -\sqrt{2}, (-1)^2, 1^2)$$

$$K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = 1 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_2 x_1' x_2' + {x_1}^2 {x_1'}^2 + {x_2}^2 {x_2'}^2$$

$$= (1 + \mathbf{x} \cdot \mathbf{x}')^2$$

$$K((-1,1), (-0.75, -0.25)) = \underline{2.25}$$

Class exercise

$$= 1 + 2(0.75) + 2(-0.25) - 2(0.75)(0.25) + (-0.75)^2 + (-0.25)^2$$

$$= \left(1 + (0.75 - 0.25)\right)^2$$

So, instead of computing $\Phi(\mathbf{x})^T \Phi(\mathbf{x}')$, we compute $\mathbf{x} \cdot \mathbf{x}'$ in the original feature space, and then compute $(1 + \mathbf{x} \cdot \mathbf{x}')^2$

Learn more at: https://en.wikipedia.org/wiki/Polynomial_kernel

General form

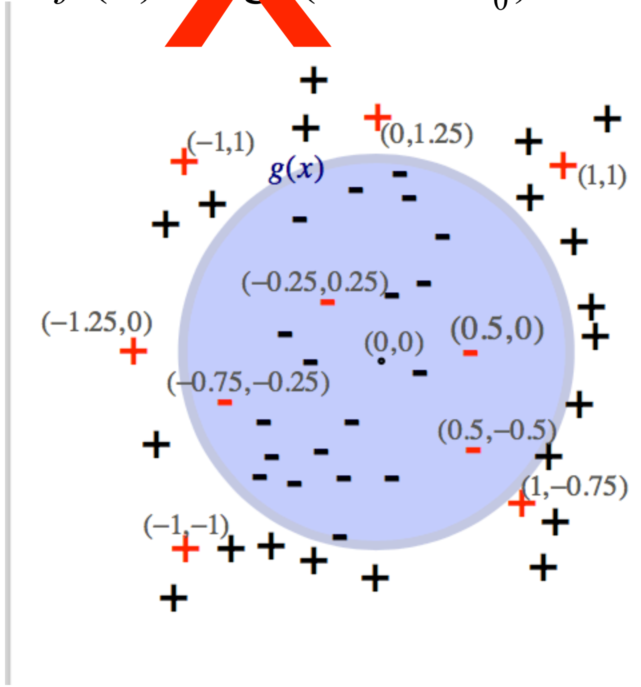
The polynomial kernel can be generalized to higher dimensions. A degree k polynomial is $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^k$

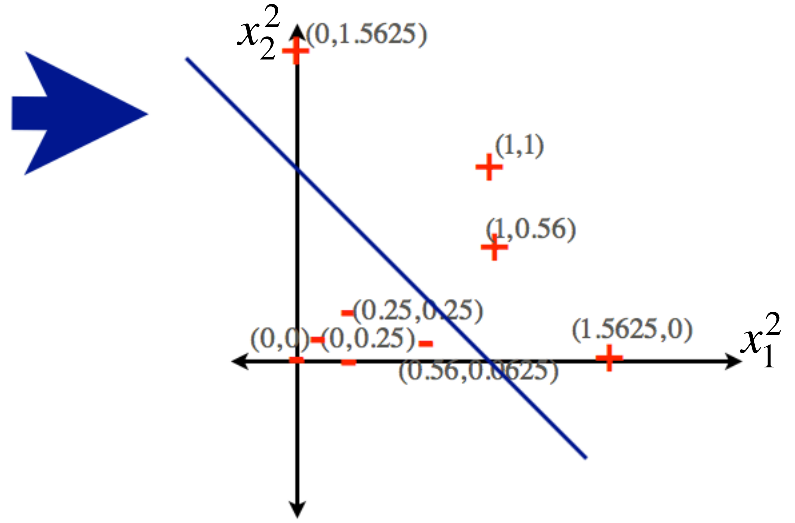
Example from the previous slide of $\Phi_2(\mathbf{x})$:
If d=200, then in z-space, we have a feature vector of size approx 20,000, but we do roughly the same amount of work to compute K(**x**,**x**') as if we hadn't transformed the features

Example

linearly separable?

$f(\mathbf{x}) = \cancel{\text{sign}}(\mathbf{w}^T\mathbf{x} + w_0)$





If we transform the features using $\Phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T$

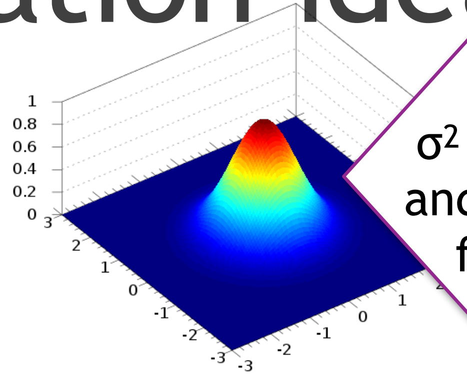
$$\text{sign}\left(\left(\sum_{i \in I} \alpha^{(i)} y^{(i)} \Phi(\mathbf{x}^{(i)})\right)^T \Phi(\mathbf{x}) + w_0\right) \Rightarrow g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)} y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + w_0\right) \Leftarrow K(\mathbf{x}^{(i)}, \mathbf{x}) = (1 + \mathbf{x}^{(i)T}\mathbf{x})^2$$

I contains the indexes of the non-zero $\alpha^{(i)}$

Radial Basis Function Kernel (RBF)

Another feature transformation idea

picture from https://en.wikipedia.org/wiki/Gaussian_function



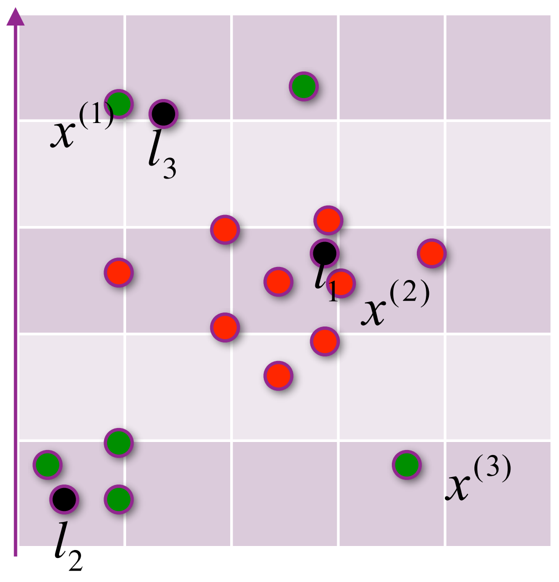
if
σ^2 is small, then lower bias and higher variance since the features taper off quicker

- Define new features, f_1, f_2, and f_3 for every example on how close it is to chosen "landmarks" $l^{(1)}$, $l^{(2)}$, and $l^{(3)}$
- For an example **x** we create new features z_1, z_2, z_3:

$$z_1 = exp\left(-\frac{\left\|\mathbf{x} - l^{(1)}\right\|_2^2}{2\sigma^2}\right)$$

$$z_2 = exp\left(-\frac{\left\|\mathbf{x} - l^{(2)}\right\|_2^2}{2\sigma^2}\right)$$

$$z_3 = exp\left(-\frac{\left\|\mathbf{x} - l^{(3)}\right\|_2^2}{2\sigma^2}\right)$$



- $\Phi(x^{(1)})=(z_1, z_2, z_3)$ approx (0, 0, 1)
- $\Phi(x^{(2)})=(z_1, z_2, z_3)$ approx (1, 0, 0)
- The larger the sigma - the more weight is given to features far away from the landmark

This slide is adapted from: https://www.coursera.org/learn/machine-learning/lecture/YOMHn/kernels-i

https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Another feature transformation

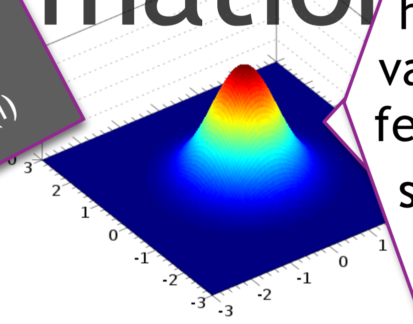
- Define new features, f_1, f_2, and f_3 for every
  how close it is to chosen "landmarks" $l^{(1)}$
- For an example **x** we create new feat ... z_3:

$$z_1 = exp\left(-\frac{\left\|\mathbf{x} - l^{(1)}\right\|_2^2}{2\sigma^2}\right)$$

$$z_2 = exp\left(-\frac{\left\|\mathbf{x} - l^{(2)}\right\|_2^2}{2\sigma^2}\right)$$

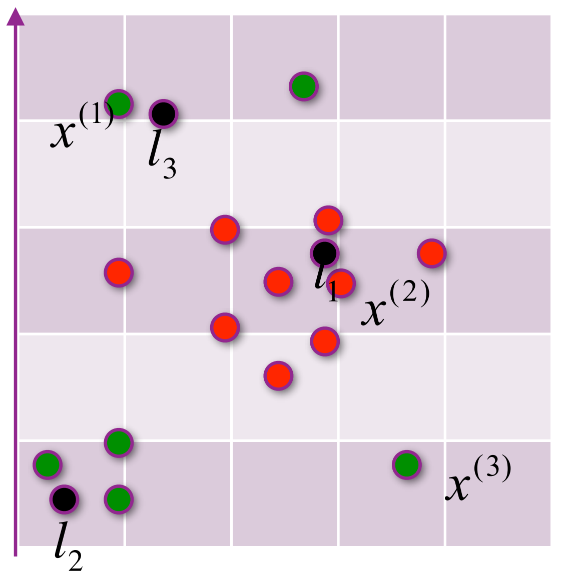
$$z_3 = exp\left(-\frac{\left\|\mathbf{x} - l^{(3)}\right\|_2^2}{2\sigma^2}\right)$$

$||\mathbf{x} - l^{(i)}||_2^2$ Square of Euclidean distance of x and $l^{(i)}$



if σ^2 is large, then higher bias and lower variance since the features taper off slower. σ^2 is the radius of influence

ias
e the
ker



- $\Phi(x^{(1)})=(z_1, z_2, z_3)$ approx (0, 0, 1)
- $\Phi(x^{(2)})=(z_1, z_2, z_3)$ approx (1, 0, 0)
- The larger the sigma - the more weight is given to features far away from the landmark

This slide is adapted from: https://www.coursera.org/learn/machine-learning/lecture/YOMHn/kernels-i

https://en.wikipedia.org/wiki/Radial_basis_function_kernel

You will not be tested on this slide

Gausian-RBF Kernel

$$\Phi(\mathbf{x}) = exp\left(-\frac{(\mathbf{x})^2}{2\sigma^2}\right)\left(1, \sqrt{\frac{2^1}{1!}}(\mathbf{x})^1, \sqrt{\frac{2^2}{2!}}(\mathbf{x})^2, \sqrt{\frac{2^3}{3!}}(\mathbf{x})^3, \ldots\right)$$

infinite-dimensions transformation

The Taylor series expansion of e^x

$$e^{\mathbf{x}} = \sum_{k=0}^{\infty} \mathbf{x}^k / k!$$

see: http://people.math.sc.edu/girardi/m142/handouts/10sTaylorPolySeries.pdf

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = exp\left(-\frac{\left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|_2^2}{2\sigma^2}\right) = exp\left(-\frac{(\mathbf{x}^{(i)})^2}{2\sigma^2}\right) exp\left(\frac{2\mathbf{x}^{(i)}\mathbf{x}^{(j)}}{2\sigma^2}\right) exp\left(-\frac{(\mathbf{x}^{(j)})^2}{2\sigma^2}\right)$$

Taylor series

$$= exp\left(-\frac{(\mathbf{x}^{(i)})^2}{2\sigma^2}\right)\left(\sum_{k=0}^{\infty} \frac{2^k (\mathbf{x}^{(i)})^k (\mathbf{x}^{(j)})^k}{k!}\right) exp\left(-\frac{(\mathbf{x}^{(j)})^2}{2\sigma^2}\right)$$

$$= \Phi(\mathbf{x}^{(i)})^T \Phi(\mathbf{x}^{(j)})$$

To extend this idea see: https://peekaboo-vision.blogspot.com/2012/12/kernel-approximations-for-efficient.html https://pdfs.semanticscholar.org/b986/6f91d6ce82384e18e8e99f803cf817787f74.pdf

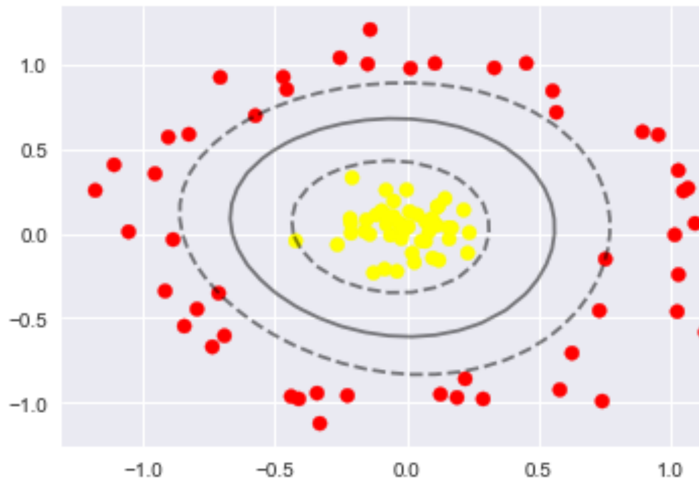
Gaussian Radial Basis Function

- For every example make a landmark: $l^{(1)}= \mathbf{x}^{(1)}$, $l^{(2)}= \mathbf{x}^{(2)}$,, …, $l^{(N)}= \mathbf{x}^{(N)}$
- Then for every x, we have created N new features $z_1, z_2, \ldots, z_n$:

$$K(\mathbf{x}^{(i)}, l^{(j)}) = z_j^{(i)} = exp\left(-\frac{\left\|\mathbf{x}^{(i)} - l^{(j)}\right\|_2^2}{2\sigma^2}\right)$$

$$z = \begin{bmatrix} \exp(\left\|\mathbf{x} - l^{(1)}\right\|_2^2 / 2\sigma^2) \\ \exp(\left\|\mathbf{x} - l^{(2)}\right\|_2^2 / 2\sigma^2) \\ \vdots \\ \exp(\left\|\mathbf{x} - l^{(N-1)}\right\|_2^2 / 2\sigma^2) \\ \exp(\left\|\mathbf{x} - l^{(N)}\right\|_2^2 / 2\sigma^2) \end{bmatrix}$$

- The number of features is now N



```
support vectors
[-0.7116806 , -0.35318537]
[-0.57297015,  0.69623782]
[ 0.7511019 , -0.1515738 ]
[-0.42044048, -0.04330787]
```

A great description is given at: https://www.coursera.org/learn/machine-learning/lecture/hxdcH/kernels-ii

Gaussian Radial Basis Function

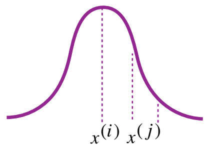
- For every example make a landı
  $l^{(1)} = \mathbf{x}^{(1)}$, $l^{(2)} = \mathbf{x}^{(2)}$,, …, $l^{(N)} = \mathbf{x}^{(N)}$
- Then for every x, we have created N new features $z_1, z_2, \ldots, z_n$:

Perform feature scaling!

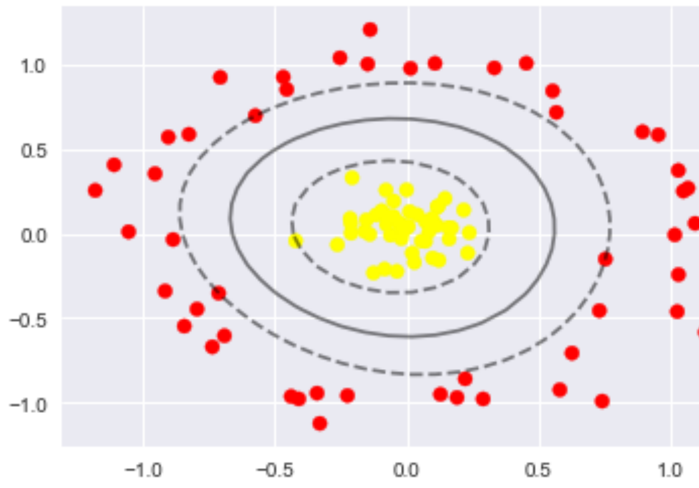
$$K(\mathbf{x}^{(i)}, l^{(j)}) = z_j^{(i)} = exp\left(-\frac{\left\|\mathbf{x}^{(i)} - l^{(j)}\right\|_2^2}{2\sigma^2}\right)$$

$$\begin{bmatrix} \exp(\left\|\mathbf{x} - l^{(1)}\right\|_2^2 / 2\sigma^2) \\ \ldots \sigma^2) \\ \ldots 2\sigma^2) \\ \exp(\left\|\mathbf{x} - l^{(N)}\right\|_2^2 / 2\sigma^2) \end{bmatrix}$$

Similarity measure. $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$ are "very similar" if they are at most σ distance

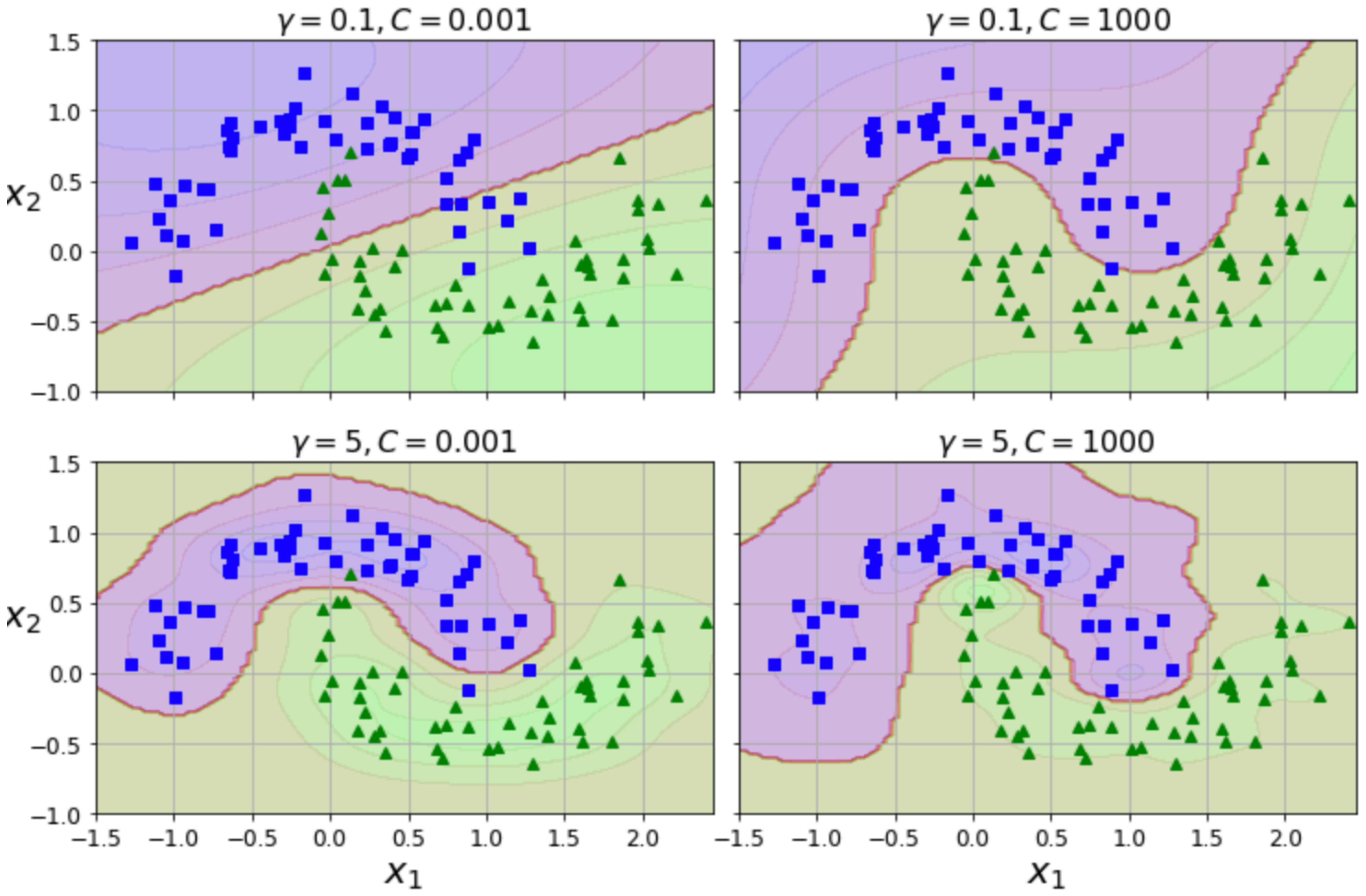


- The number of features is now N



```
support vectors
[-0.7116806 , -0.35318537]
[-0.57297015,  0.69623782]
[ 0.7511019 , -0.1515738 ]
[-0.42044048, -0.04330787]
```

A great description is given at: https://www.coursera.org/learn/machine-learning/lecture/hxdcH/kernels-ii



$$\gamma = \frac{1}{2\sigma^2}$$

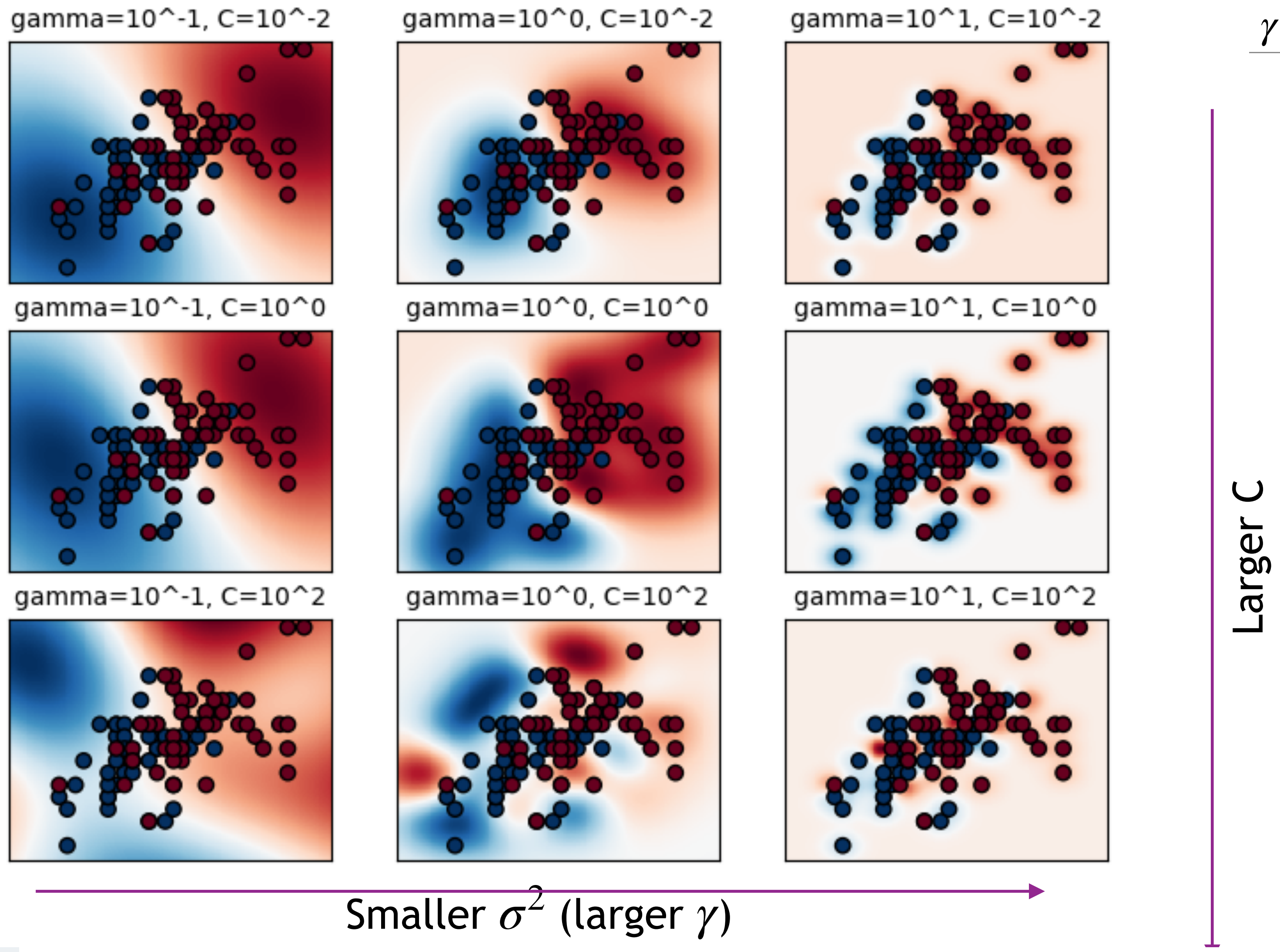
$$\exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|_2^2\right)$$

Smaller γ, gives more weight for points further away

Graph from Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow

Example from http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

$\gamma = 1/(2\sigma^2)$



"Intuitively, the `gamma` parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'."

SVMs

- Maximizes distance of training data to boundary (built in regularization)
- Generalizes to nonlinear decision boundaries (I.e. feature transformation)
- Performs well with high dimensional data

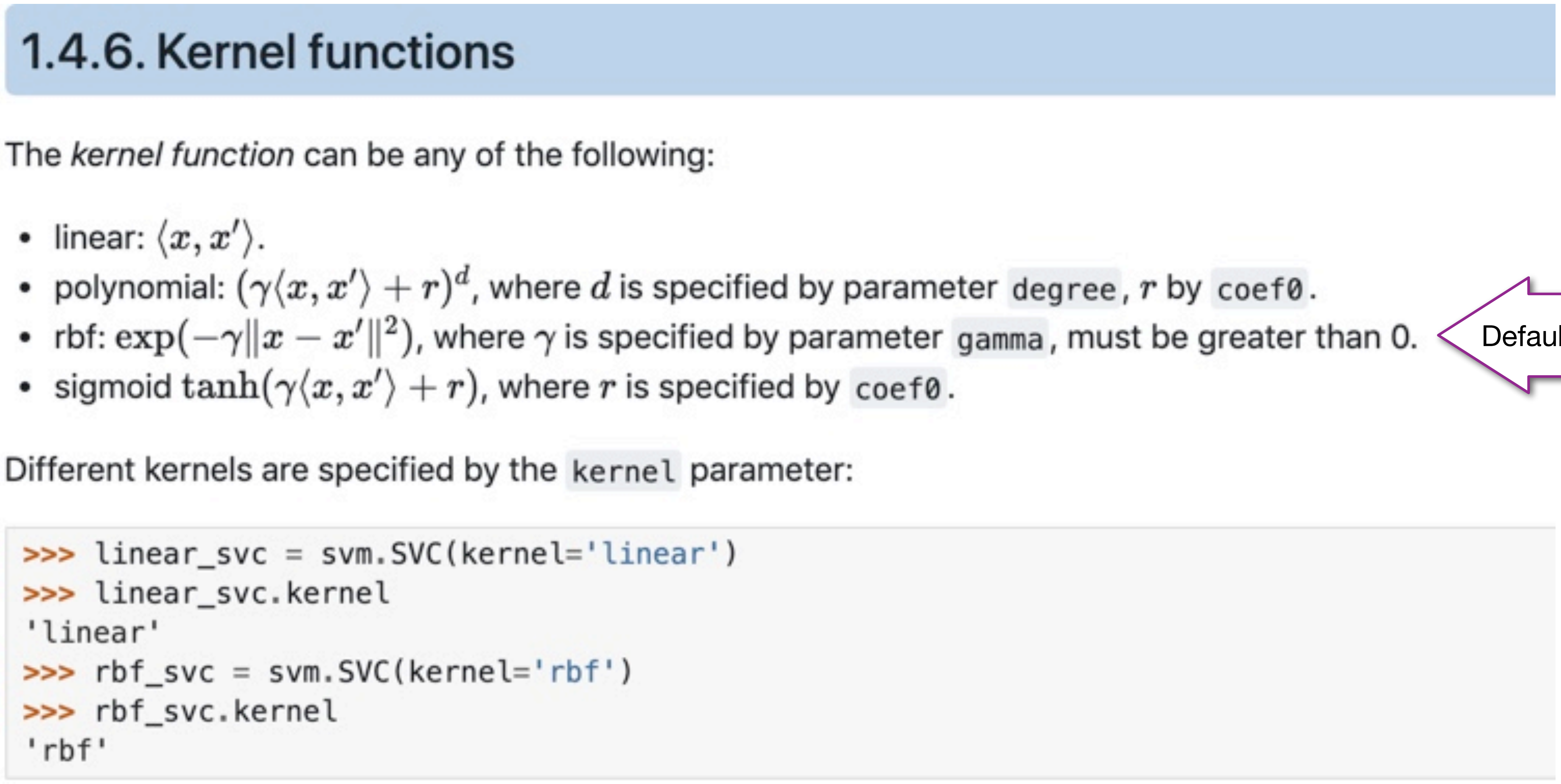
Sklearn

LinearSVC: optimizer for the primal problem $O(N \times d)$

SVC: optimizer for the dual problem

typically between $O(N^2 \times d)$ and $O(N^3 \times d)$

From: https://scikit-learn.org/dev/modules/svm.html#svm-kernels



1.4.6. Kernel functions

The *kernel function* can be any of the following:

- linear: $\langle x, x'\rangle$.
- polynomial: $(\gamma\langle x, x'\rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma\|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma\langle x, x'\rangle + r)$, where r is specified by `coef0`.

Default kernel

Different kernels are specified by the `kernel` parameter:

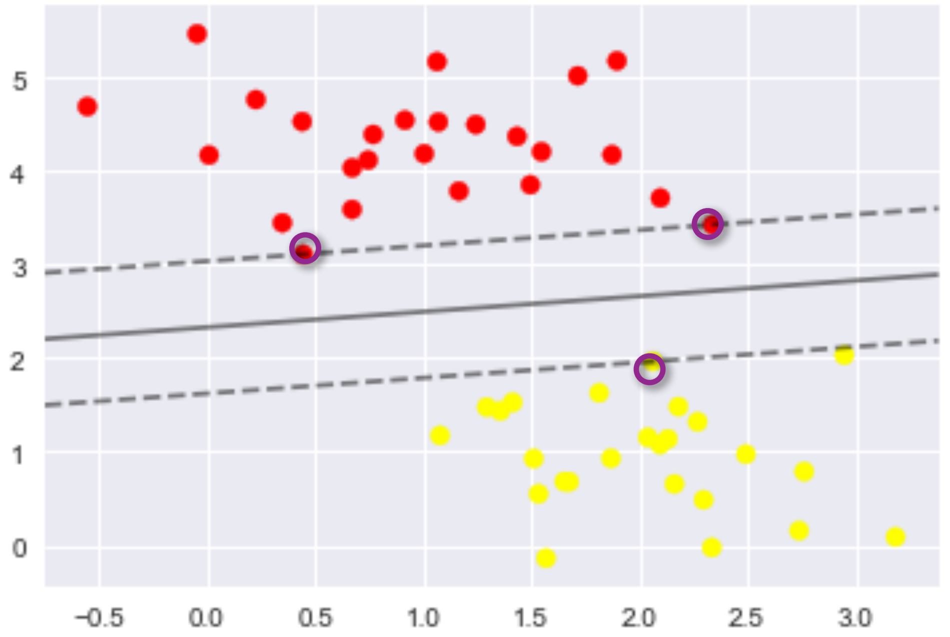
```
>>> linear_svc = svm.SVC(kernel='linear')
>>> linear_svc.kernel
'linear'
>>> rbf_svc = svm.SVC(kernel='rbf')
>>> rbf_svc.kernel
'rbf'
```

Using sklearn's support vector classifier to find the maximum margin on the training examples X

```
from sklearn.svm import SVC # "Support vector
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

By choosing the parameter C to be very large we assure no misclassification points

The classifier finds the optimal model for the training examples



We can observe what the support vectors are:

```
model.support_vectors_
```

```
array([[ 0.44359863,  3.11530945],
       [ 2.33812285,  3.43116792],
       [ 2.06156753,  1.96918596]])
```

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

When using the SVM, we don't need to compute the coefficient vector.

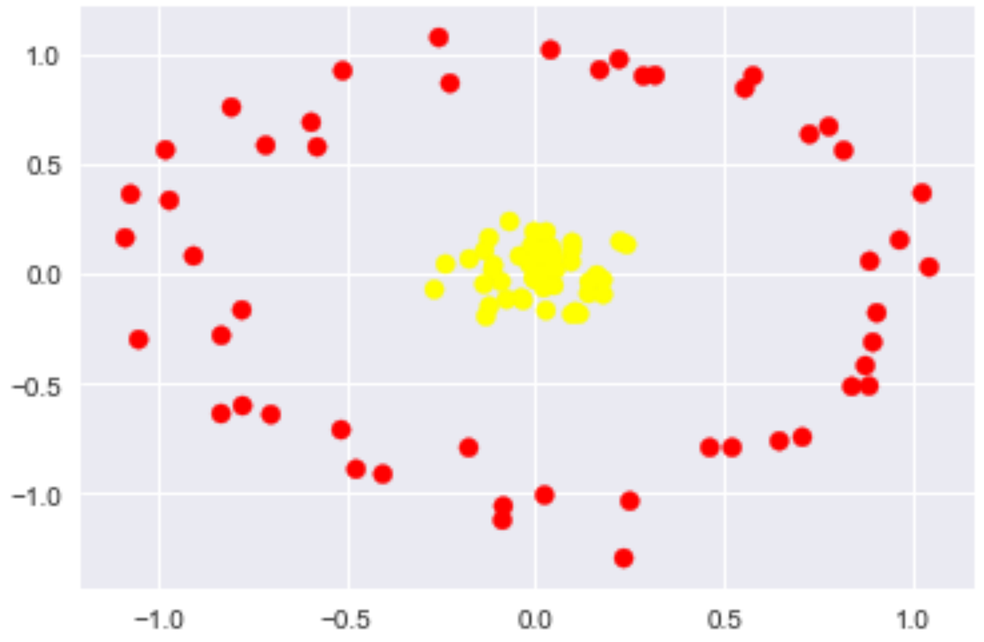
But, if we wanted to find it, how can we determine $\mathbf{w}$?

1) We could use the formula $\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$

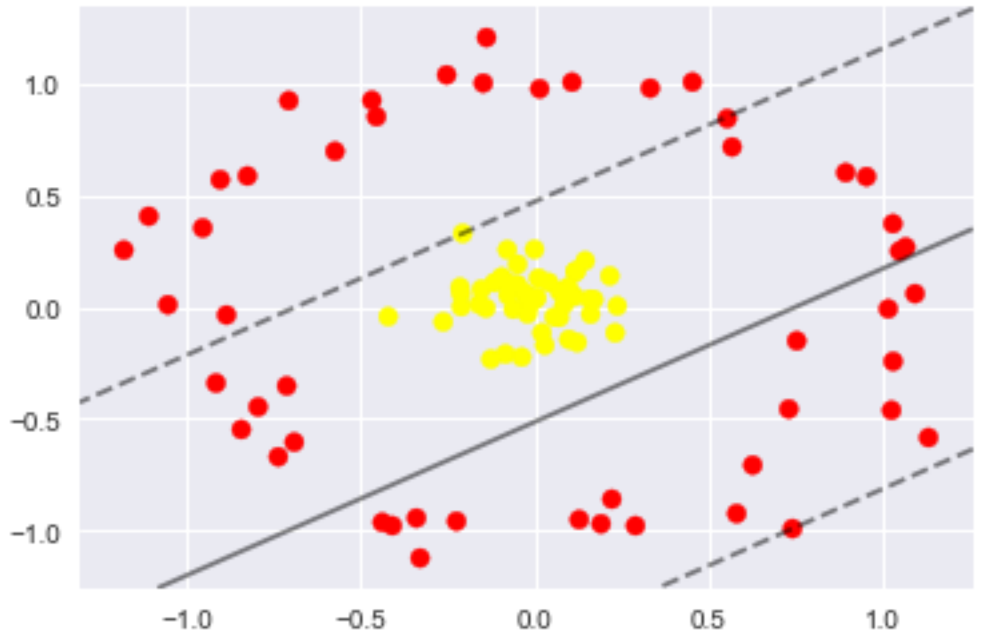
2) Or we retrieve the information from Sklearn if we are using a linear kernel: model.coef_

Suppose the data is not linearly separable

Lets try to fit it with a linear support vector machine (we will allow for some misclassification)…



```
clf = SVC(kernel='linear').fit(X, y)
```



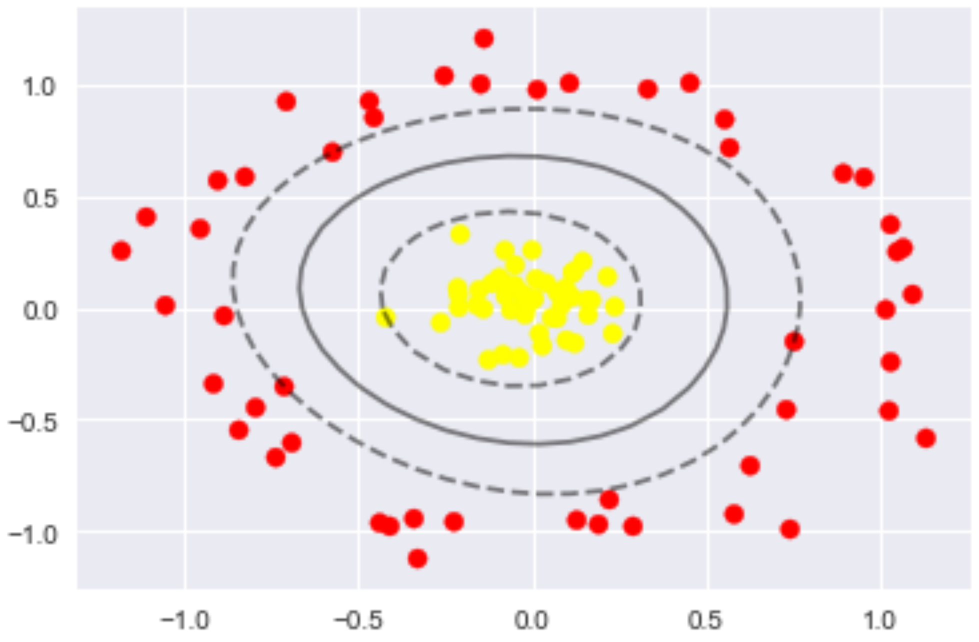
https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

How can we find the right point to transform the points to become linearly separable?

Easiest answer is to compute the distance to every point in the data set!
Problem? The number of features becomes N, the number of training examples

In sklearn, we can apply the radial basis function kernel (RBF kernel)

```
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

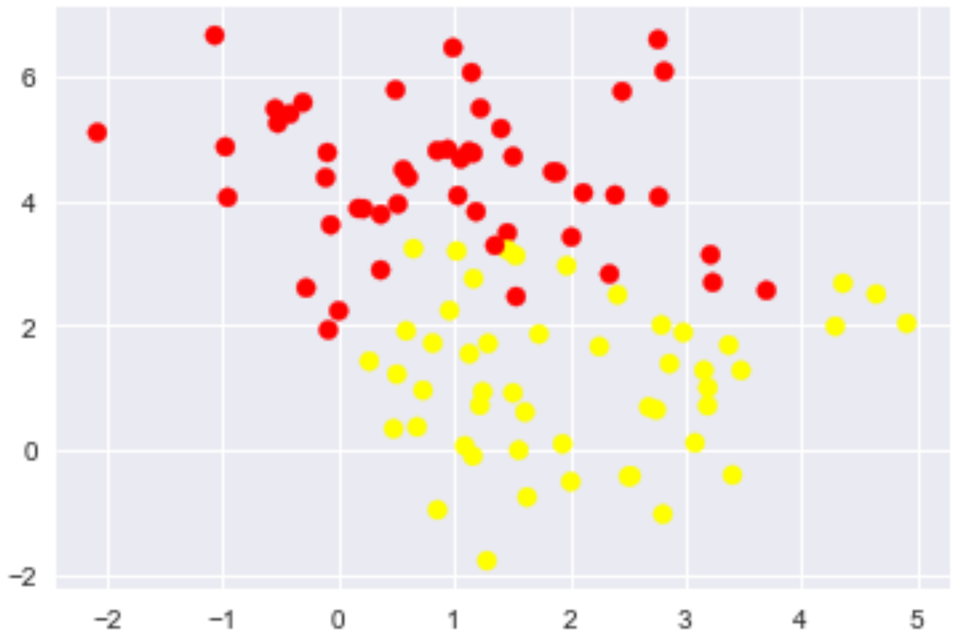


```
clf.support_vectors_
```

```
array([[-0.7116806 , -0.35318537],
       [-0.57297015,  0.69623782],
       [ 0.7511019 , -0.1515738 ],
       [-0.42044048, -0.04330787]])
```

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

Another way to work with non separable data is to allowing for errors - Soft Margin SVM



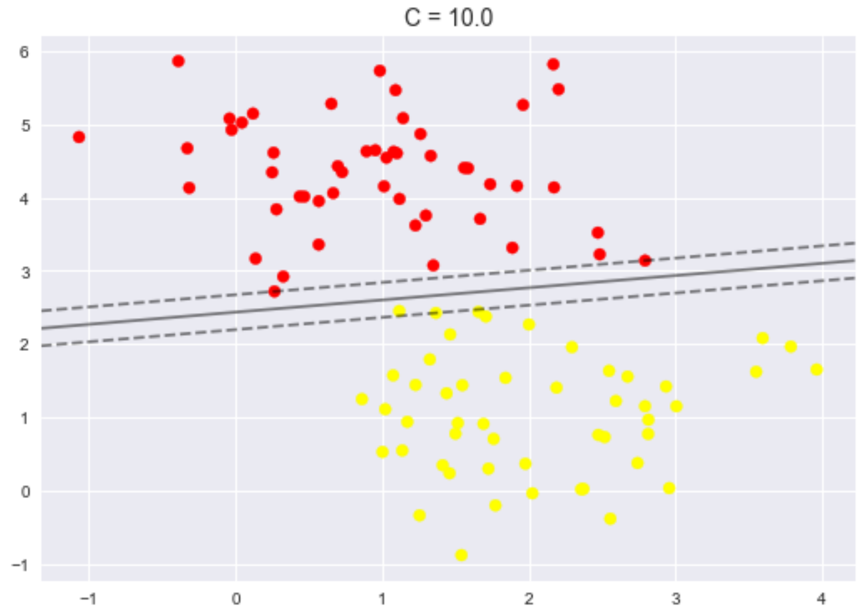
https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

We create a soft- margin classifier by adding a cost to points that are not correctly classified and outside the margin

The primal objective function: $$\min_{w_0, \mathbf{w}, \{\xi^{(i)}\}_{i=1}^{N}} \|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} \xi^{(i)}$$

$$\text{subject to } y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) \geq 1 - \xi^{(i)} \quad \text{for all } i = 1, \dots, N \qquad \xi^{(i)} \geq 0$$

The tuning parameter C softens (C smaller) or hardens (C larger)

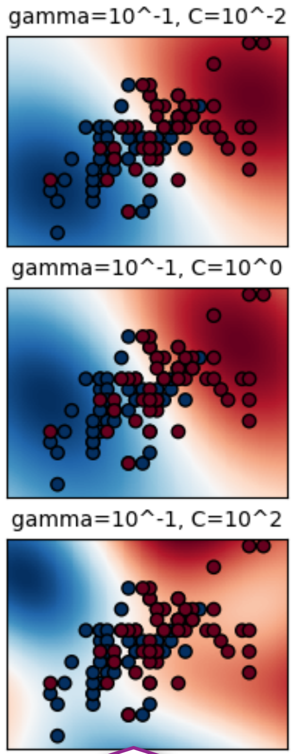


model = SVC(kernel='linear', C=10).fit(X, y)

model = SVC(kernel='linear', C=0.1).fit(X, y)

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

SVC



Underfitting for small C

Under-fitting for Small gamma

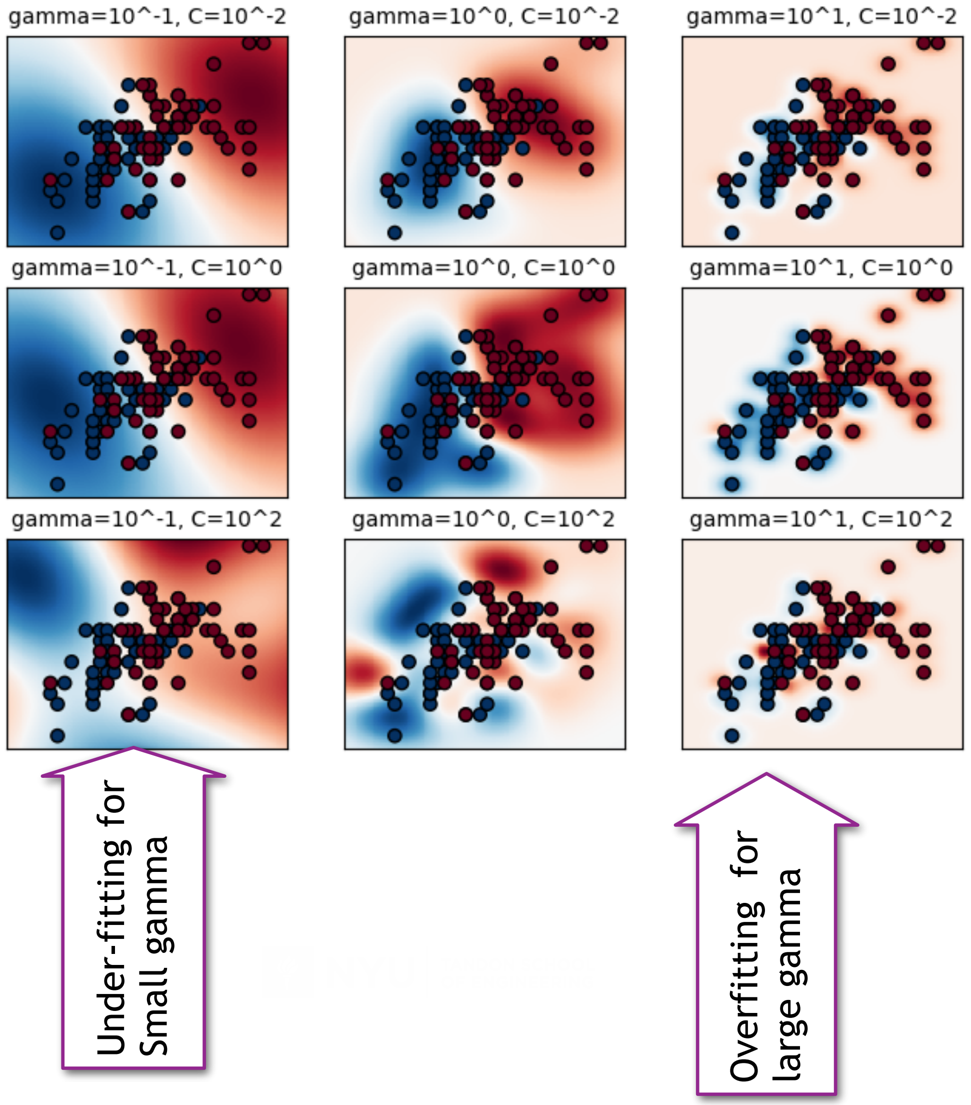
```
from sklearn.svm import SVC


C_2d_range = [1e-2, 1, 1e2]
gamma_2d_range = [1e-1, 1, 1e1]
classifiers = []
for C in C_2d_range:
    for gamma in gamma_2d_range:
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X_2d, y_2d)
        classifiers.append((C, gamma, clf))
```

“Intuitively, the `gamma` parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.”
“The `C` parameter trades off correct classification of training examples against maximization of the decision function’s margin”

The radius of the RBF kernel alone acts as a good structural regularizer.

Example from http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

SVC



```python
from sklearn.svm import SVC
```

```python
C_2d_range = [1e-2, 1, 1e2]
gamma_2d_range = [1e-1, 1, 1e1]
classifiers = []
for C in C_2d_range:
    for gamma in gamma_2d_range:
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X_2d, y_2d)
        classifiers.append((C, gamma, clf))
```

“Intuitively, the `gamma` parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.”
“The `C` parameter trades off correct classification of training examples against maximization of the decision function’s margin”

The radius of the RBF kernel alone acts as a good structural regularizer.

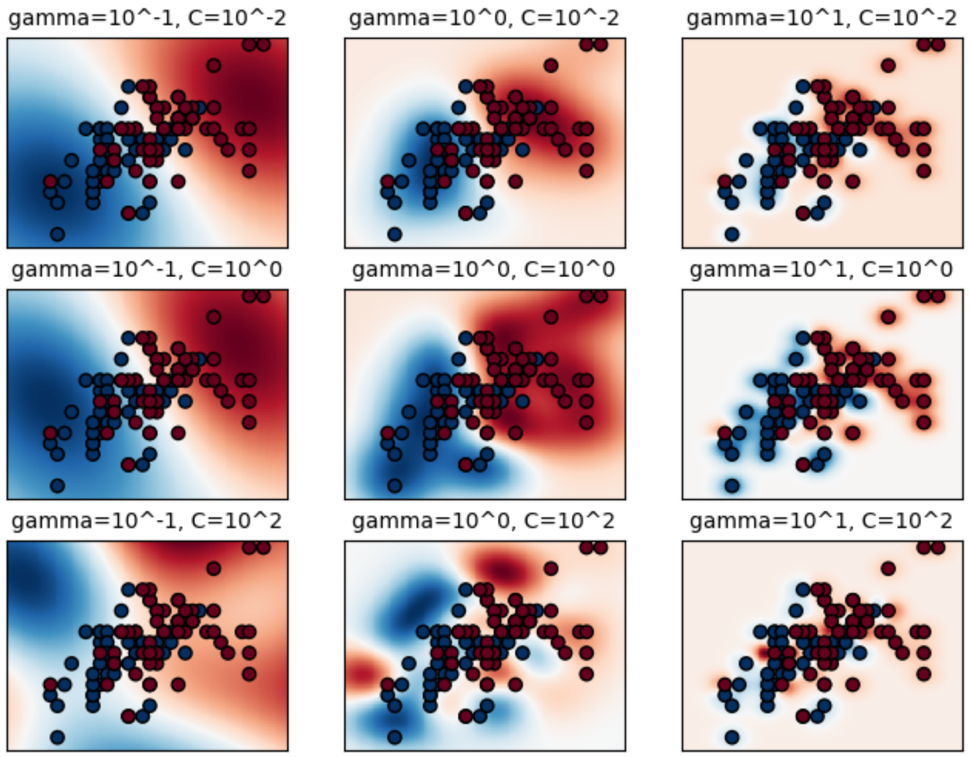
Example from http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Ack!!! There are too many parameters to tune

sklearn’s RandomizedSearchCV and GridSearchCV are classes for parameter tuning that methodically builds and evaluates different combinations of parameters as a grid

GridSearchCV

These also need to be imported



```
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV

# Train classifiers
#
# For an initial search, a logarithmic grid with basis
# 10 is often helpful. Using a basis of 2, a finer
# tuning can be achieved but at a much higher cost.

C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
grid.fit(X, y)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

Example from http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html