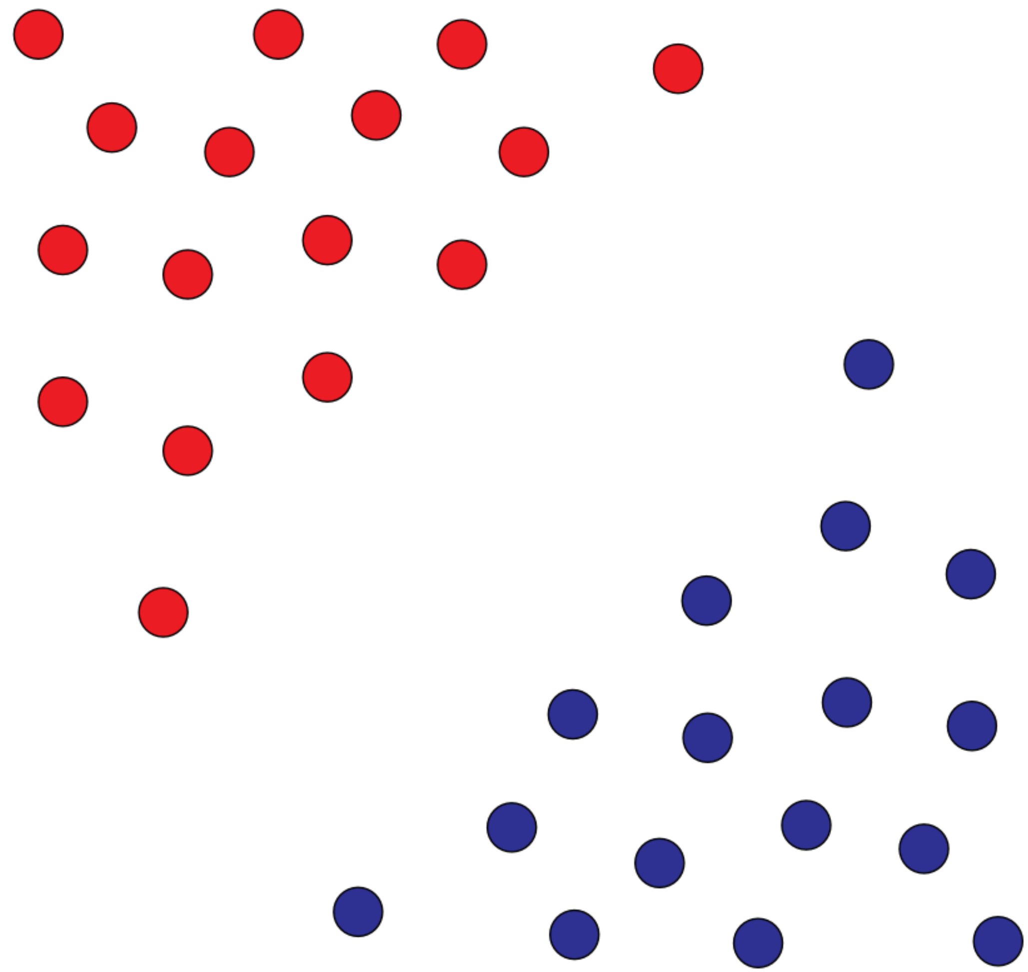
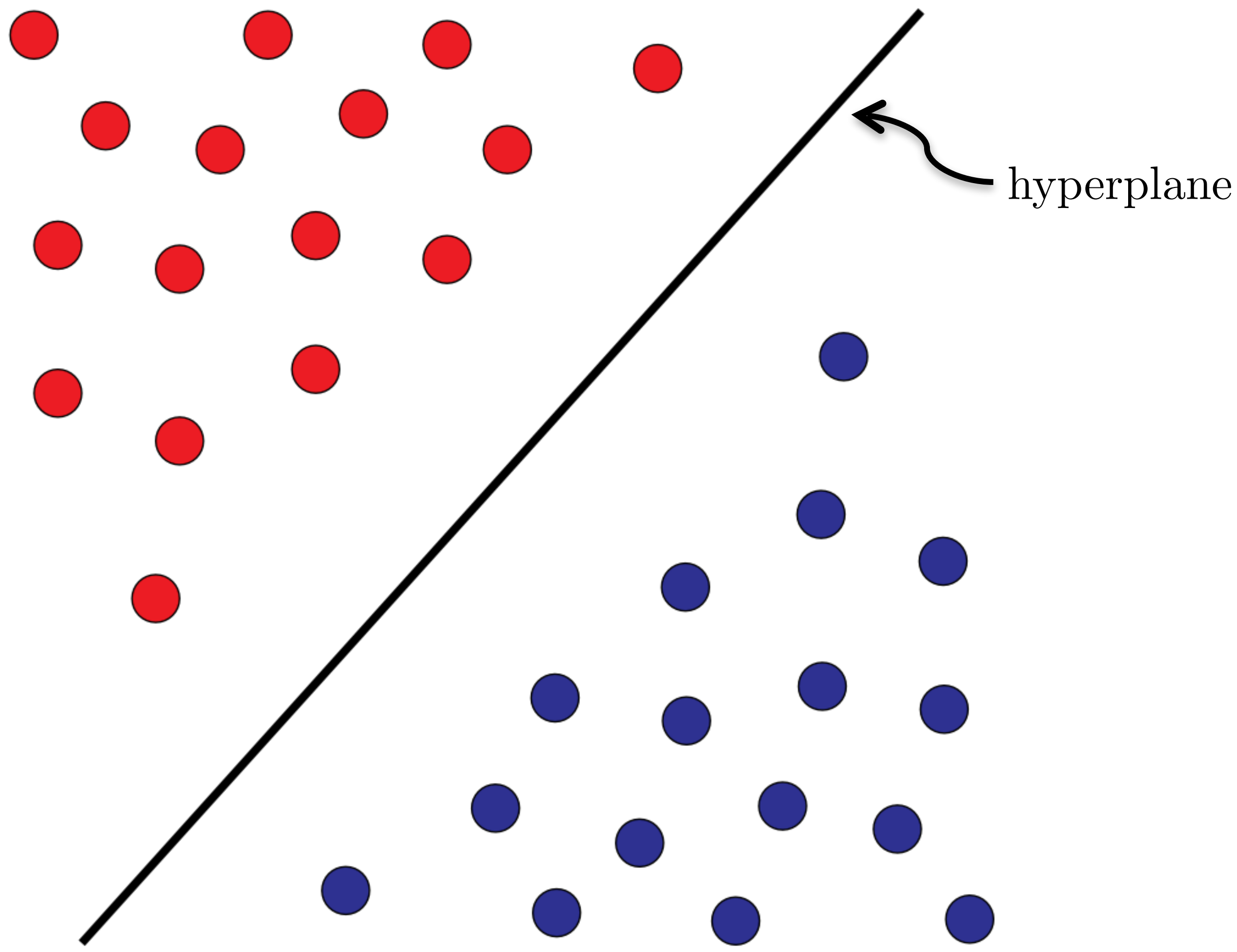
Support Vector Machines

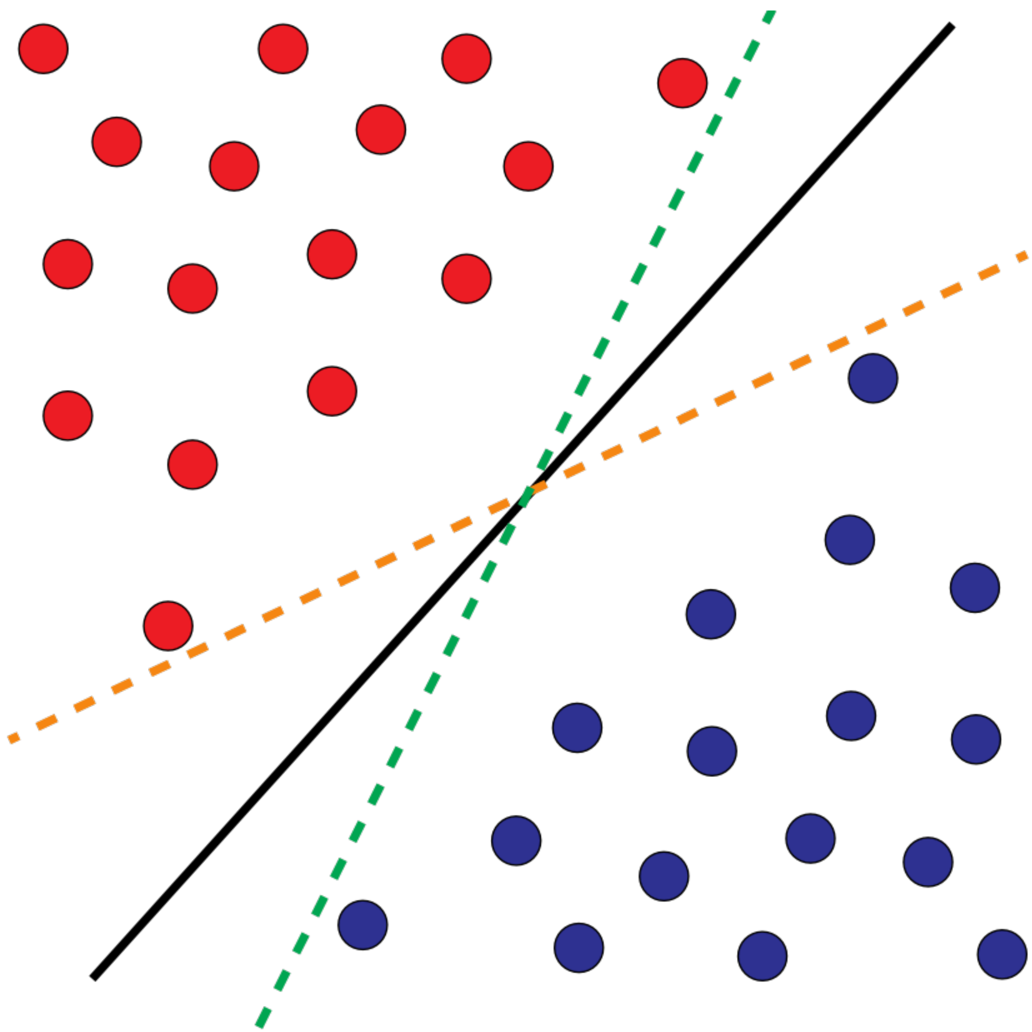
Linearly Separable Data



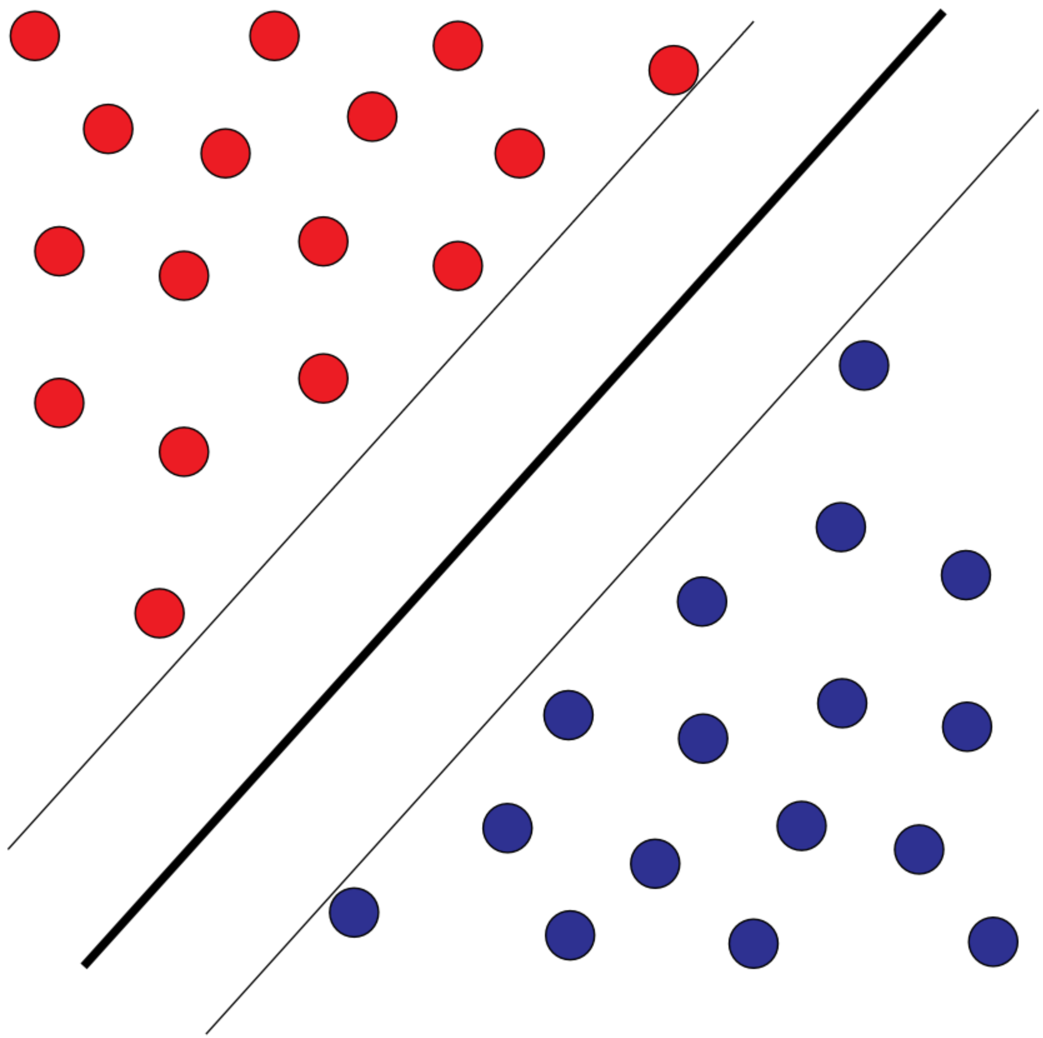
SVM: Simple Linear Separator



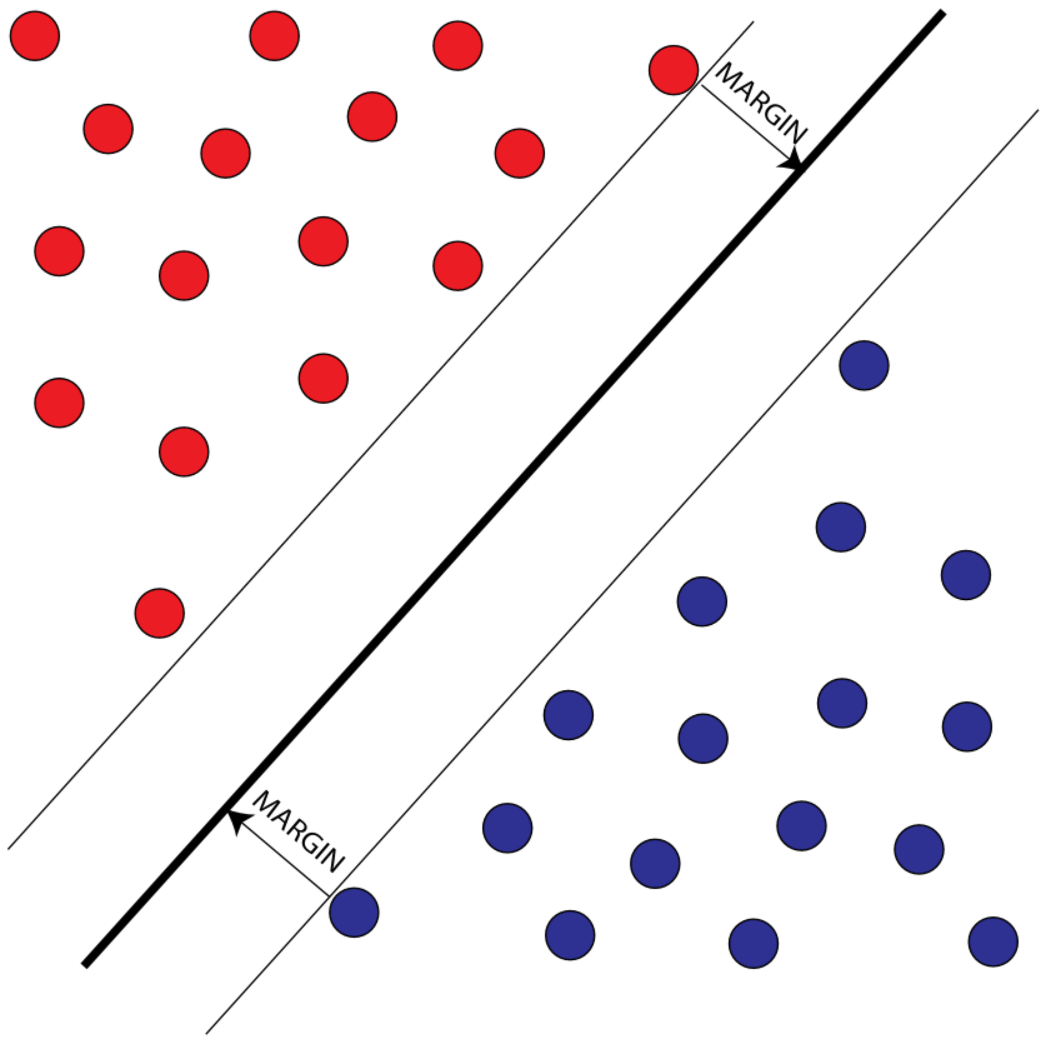
Which Simple Linear Separator?



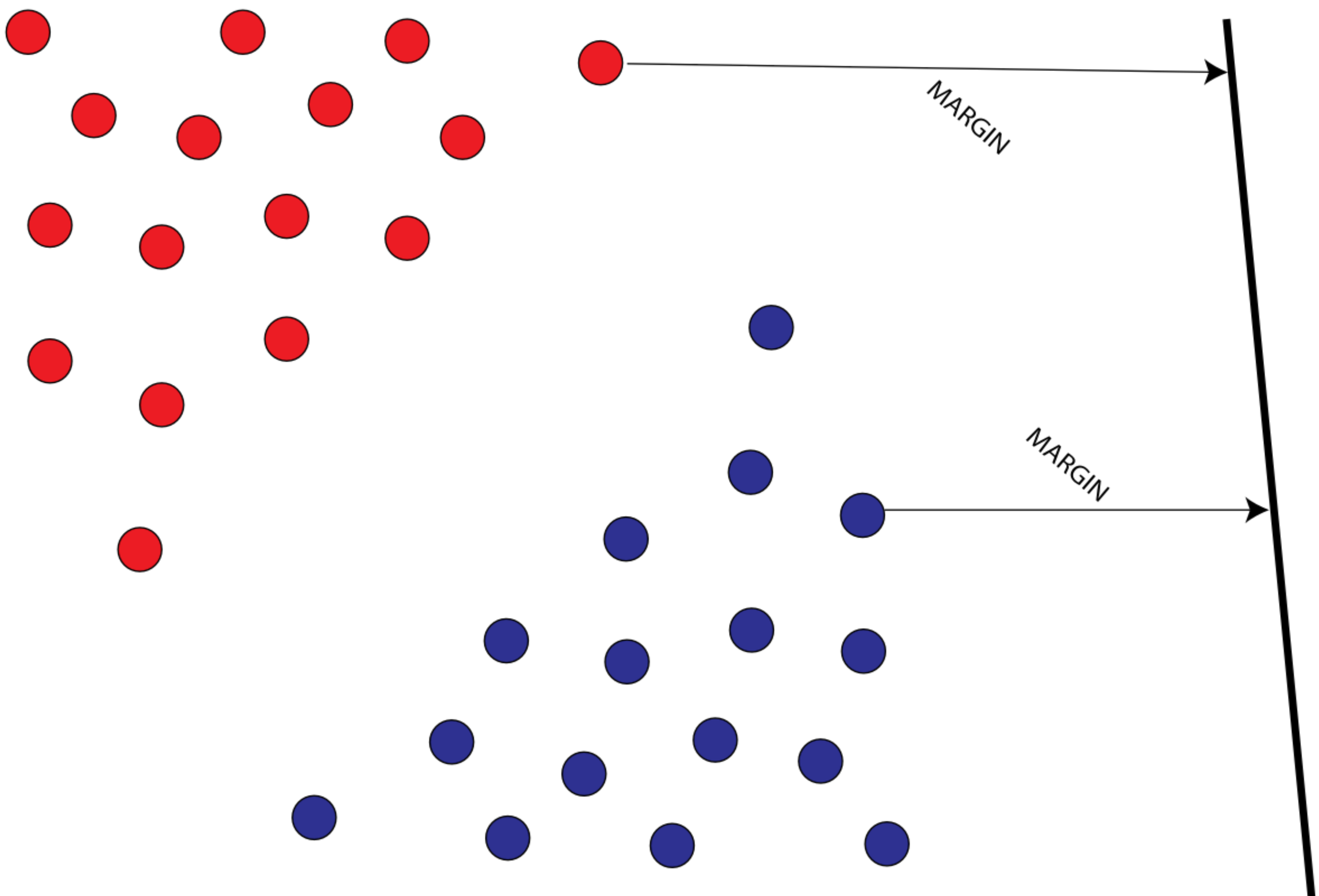
Classifier Margin



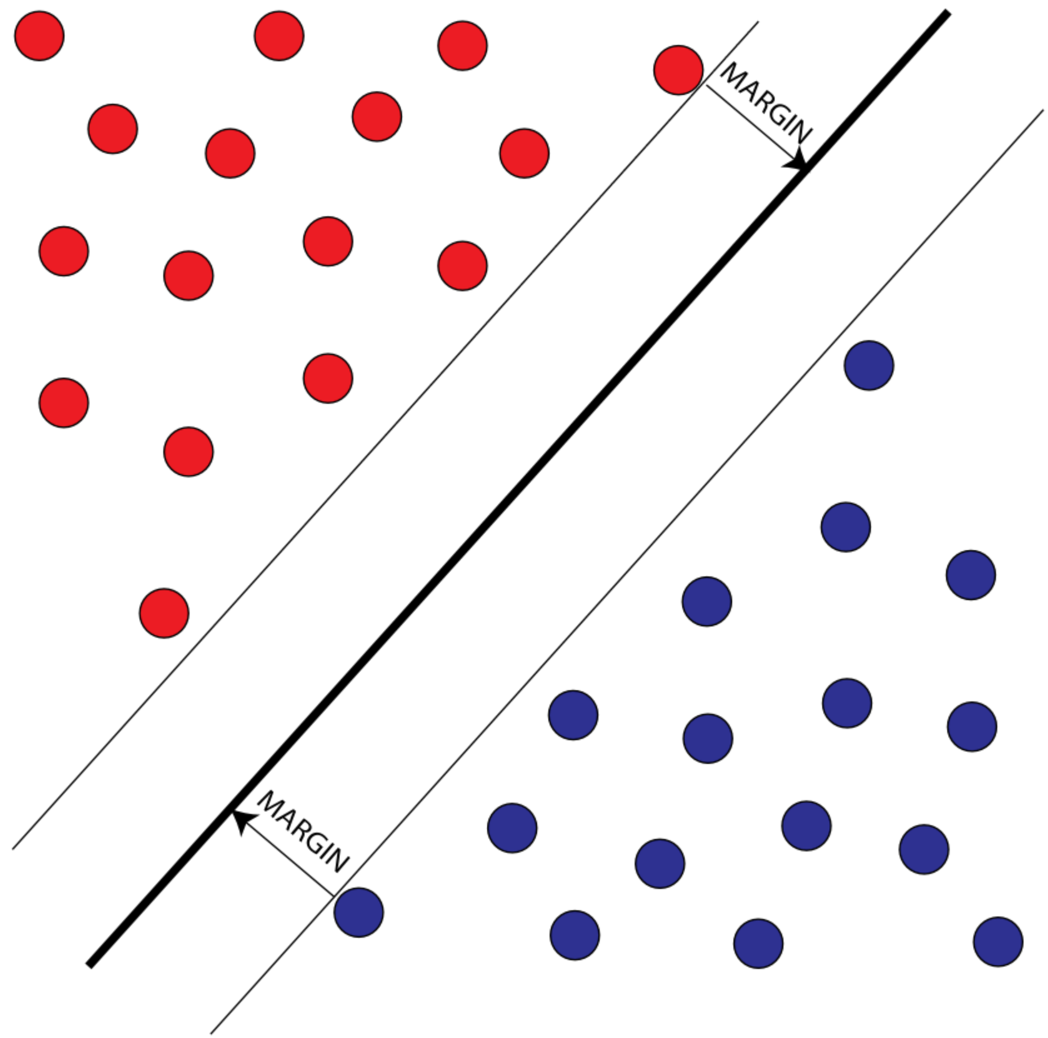
Objective #1: Maximize Margin



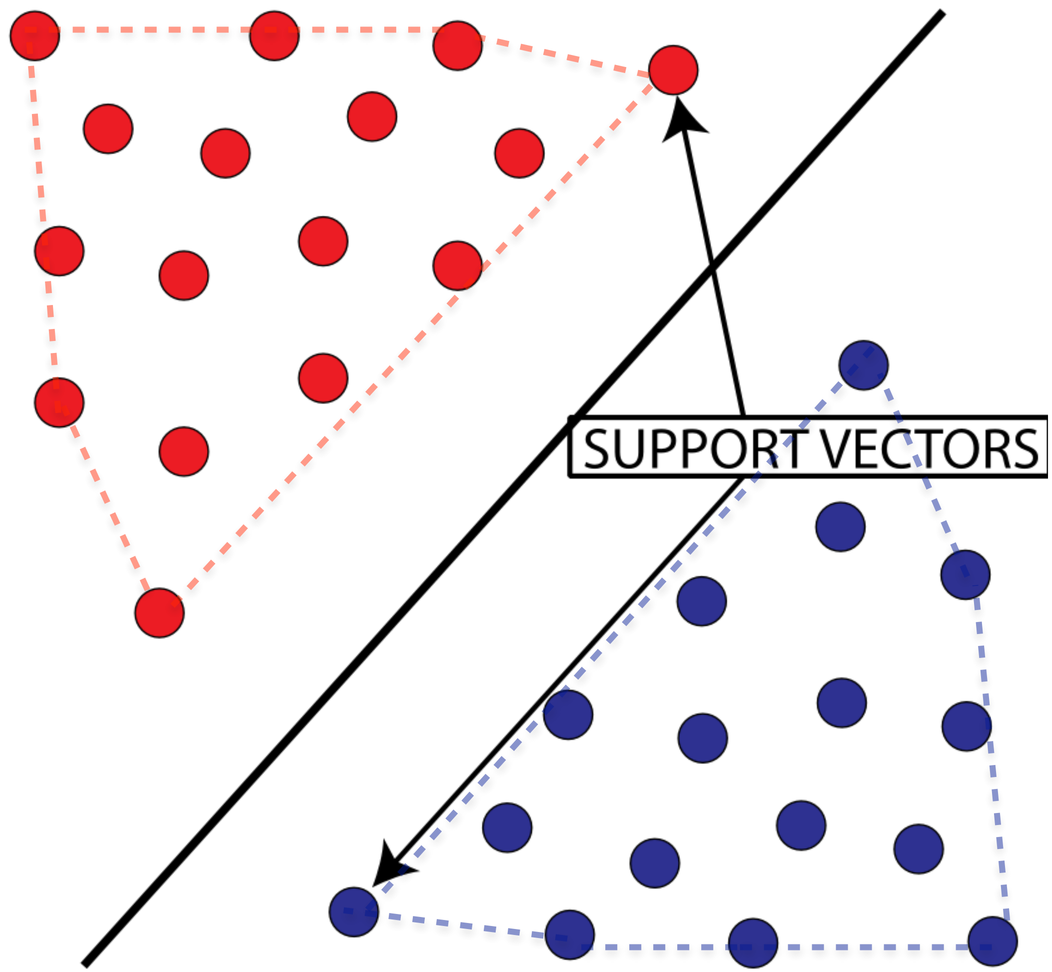
How's this look?



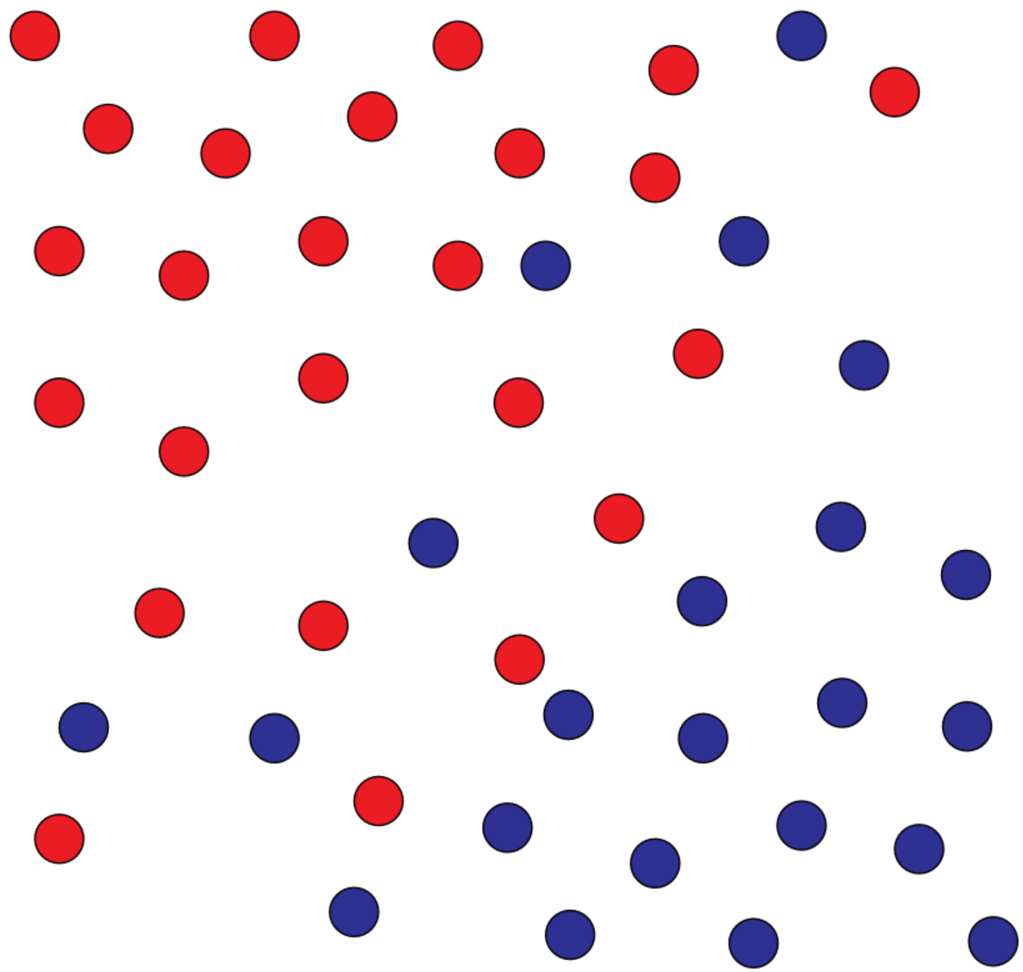
Objective #2: Minimize Misclassifications



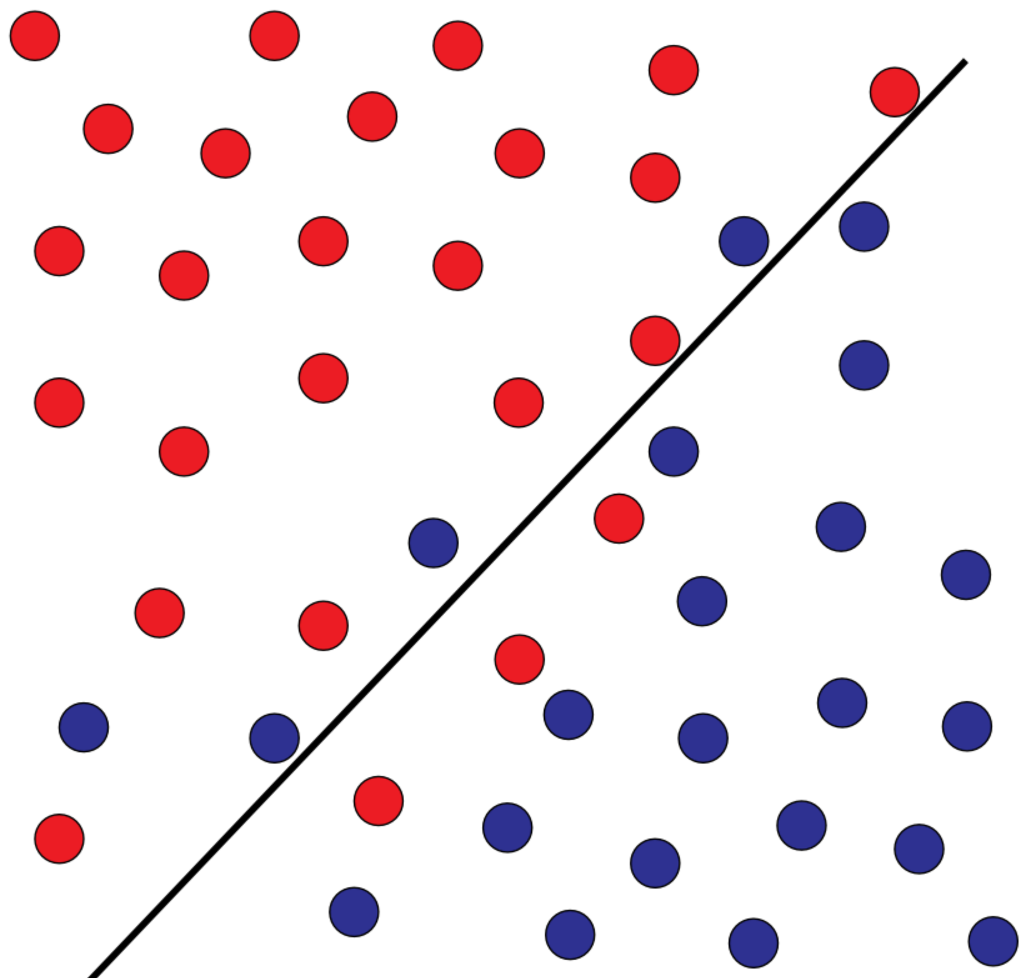
Support Vectors



Not Linearly Separable



SVM w/ "Soft Margin"



The SVM Classifier Model

- A hyperplane in $\mathbb{R}^p$ can be represented by a vector w with p elements (p= #variables), plus a "bias" term, w_0, which lifts it away from the origin.

  $$f(\mathrm{x}) = w_0 + \mathbf{w}^\mathbf{T}\mathbf{x} = 0 \qquad \text{(equation of decision } \textit{boundary}\text{)}$$

- Any observation, x, 'above' the hyperplane has

  $$f(\mathrm{x}) = w_0 + \mathbf{w}^\mathbf{T}\mathbf{x} > 0$$

- Any observation, x, 'below' the hyperplane has

  $$f(\mathrm{x}) = w_0 + \mathbf{w}^\mathbf{T}\mathbf{x} < 0$$

The SVM Classifier Model

- Decision *boundary:* $\mathrm{f(x)} = w_0 + \mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$
- 'Above' the hyperplane: $\mathrm{f(x)} = w_0 + \mathbf{w}^{\mathbf{T}}\mathbf{x} > 0$
- 'Below' the hyperplane $\mathrm{f(x)} = w_0 + \mathbf{w}^{\mathbf{T}}\mathbf{x} < 0$
- Binary target variable y is coded as {+1, -1} so that

$$y_i \cdot f(\mathbf{x_i}) > 0$$

means obs. *i* was correctly classified.

The input...

- Input data and a class target.

- For best results, input data should be centered and standardized/normalized

- Hyperparameters for regularization and kernels.
  - (more on this in a minute...)

The output...

The output 'model' will be a set of parameters (i.e. a vector, **w**, plus an intercept w_0)

For a new example, **x**:

- If $w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x} < 0$ then predict target $= -1$
- If $w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x} > 0$ then predict target $= +1$

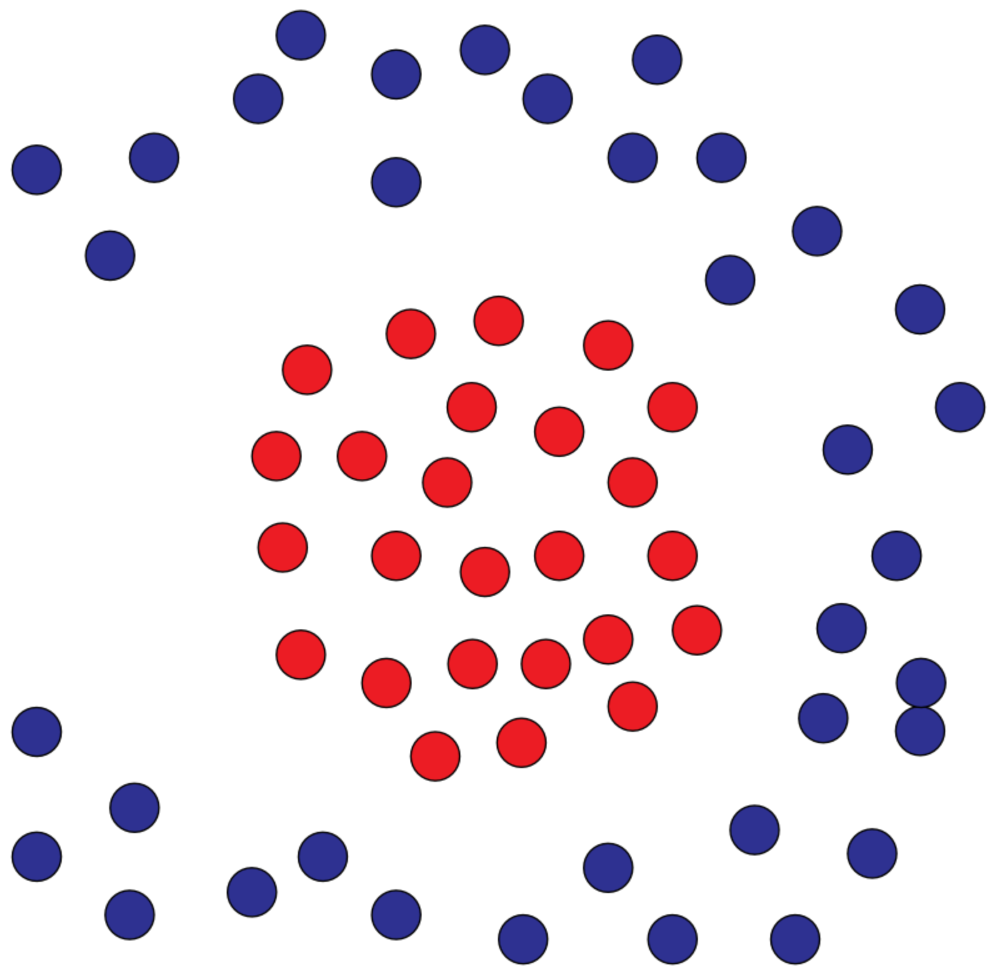
The above output changes when kernels are used, and it is best to use the model as an output object in that case.

Nonlinear SVMs

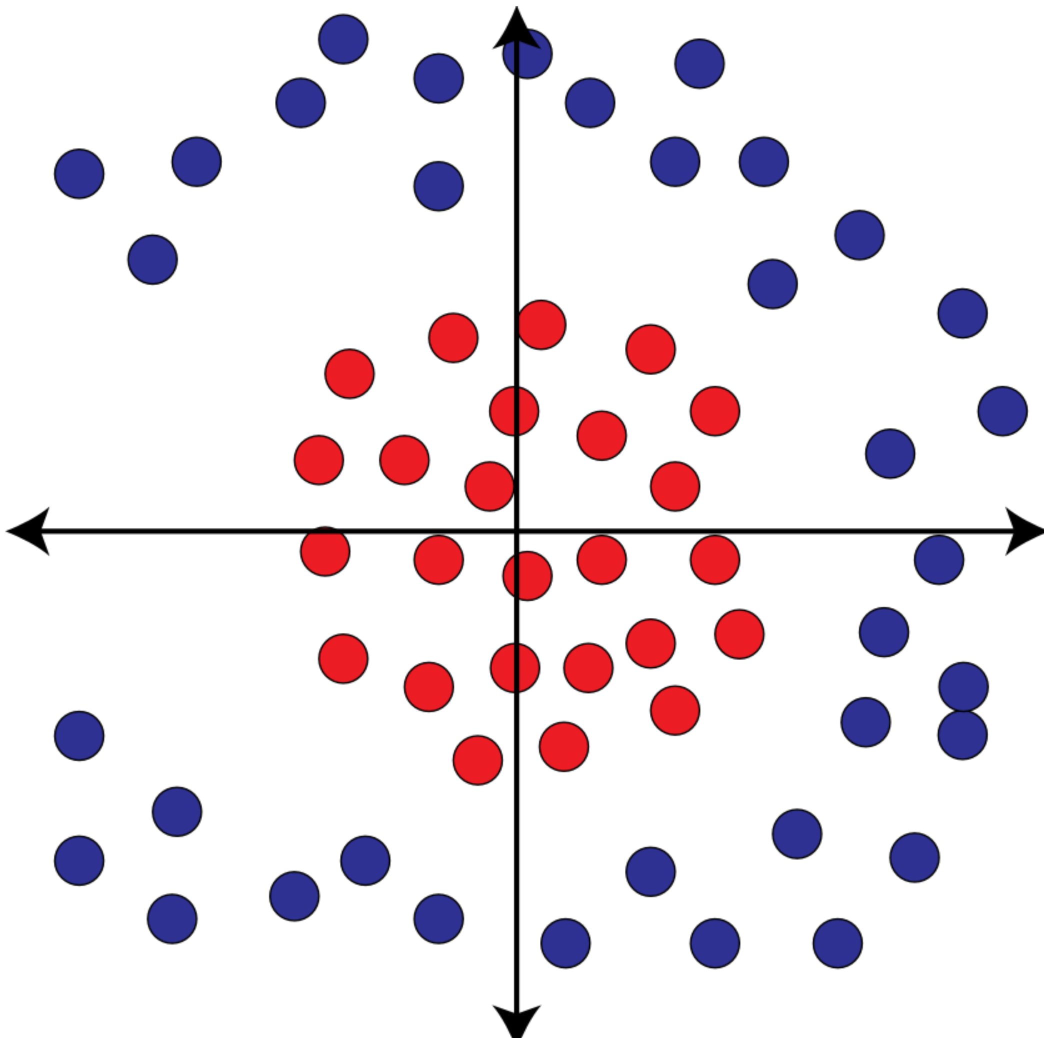
• • •

"The Kernel Trick"

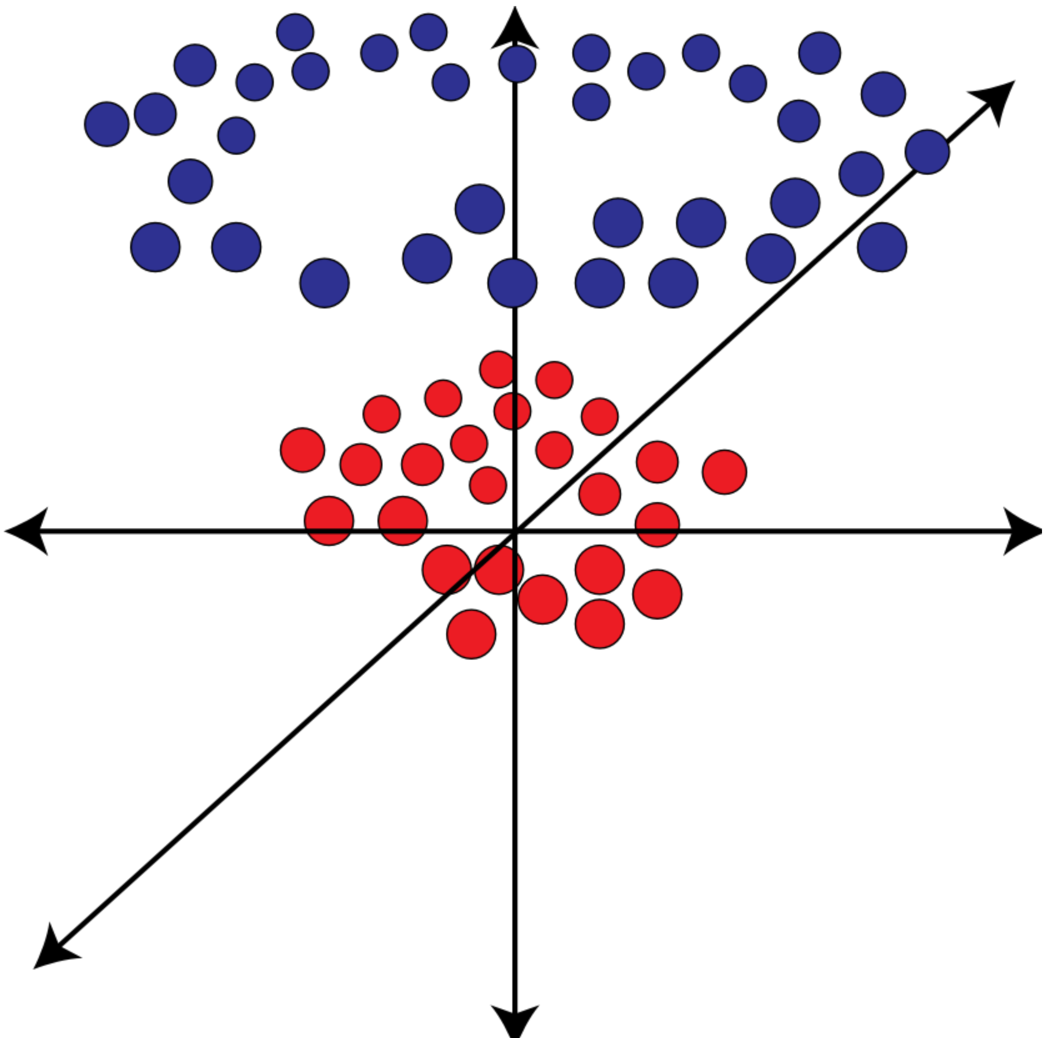
Not Linearly Separable



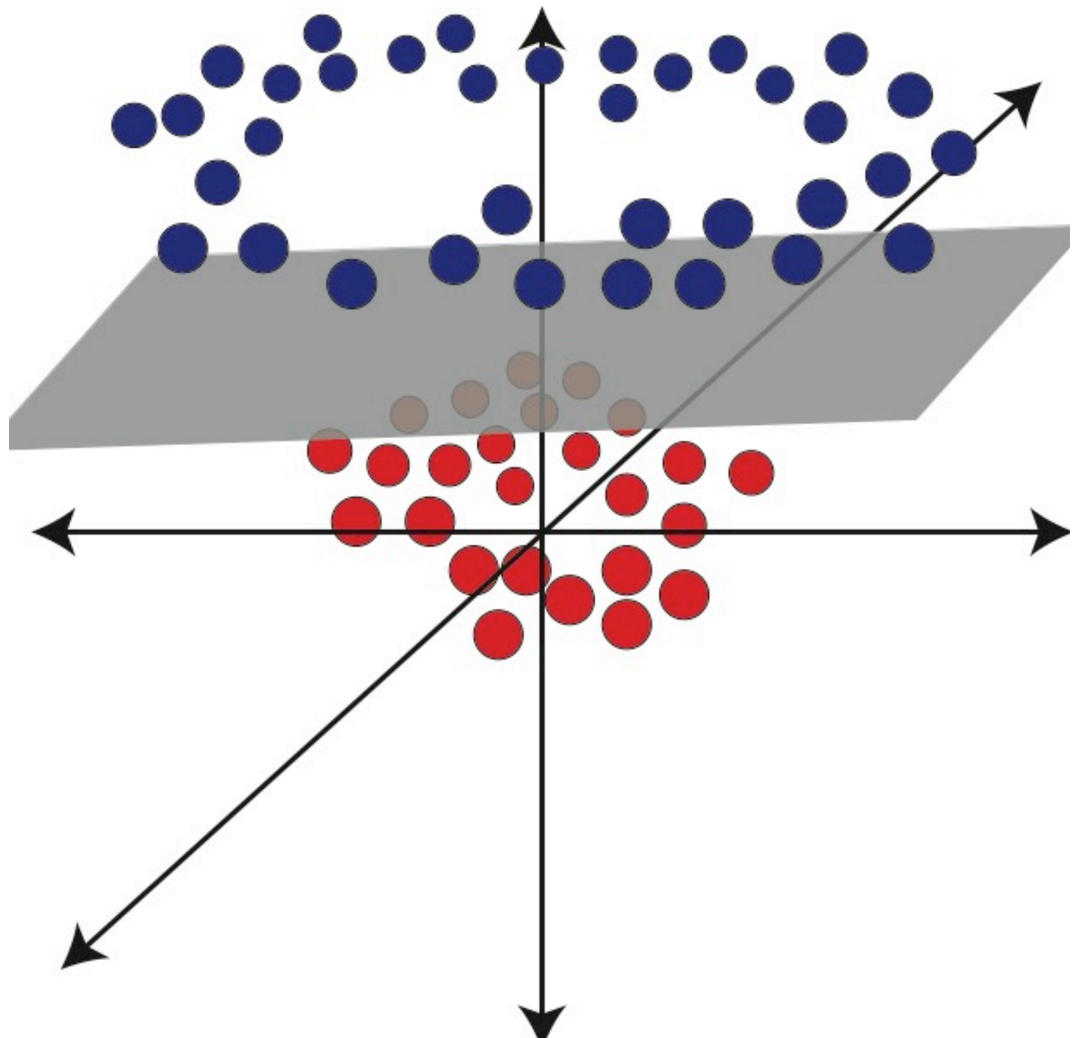
Create Additional Variables?



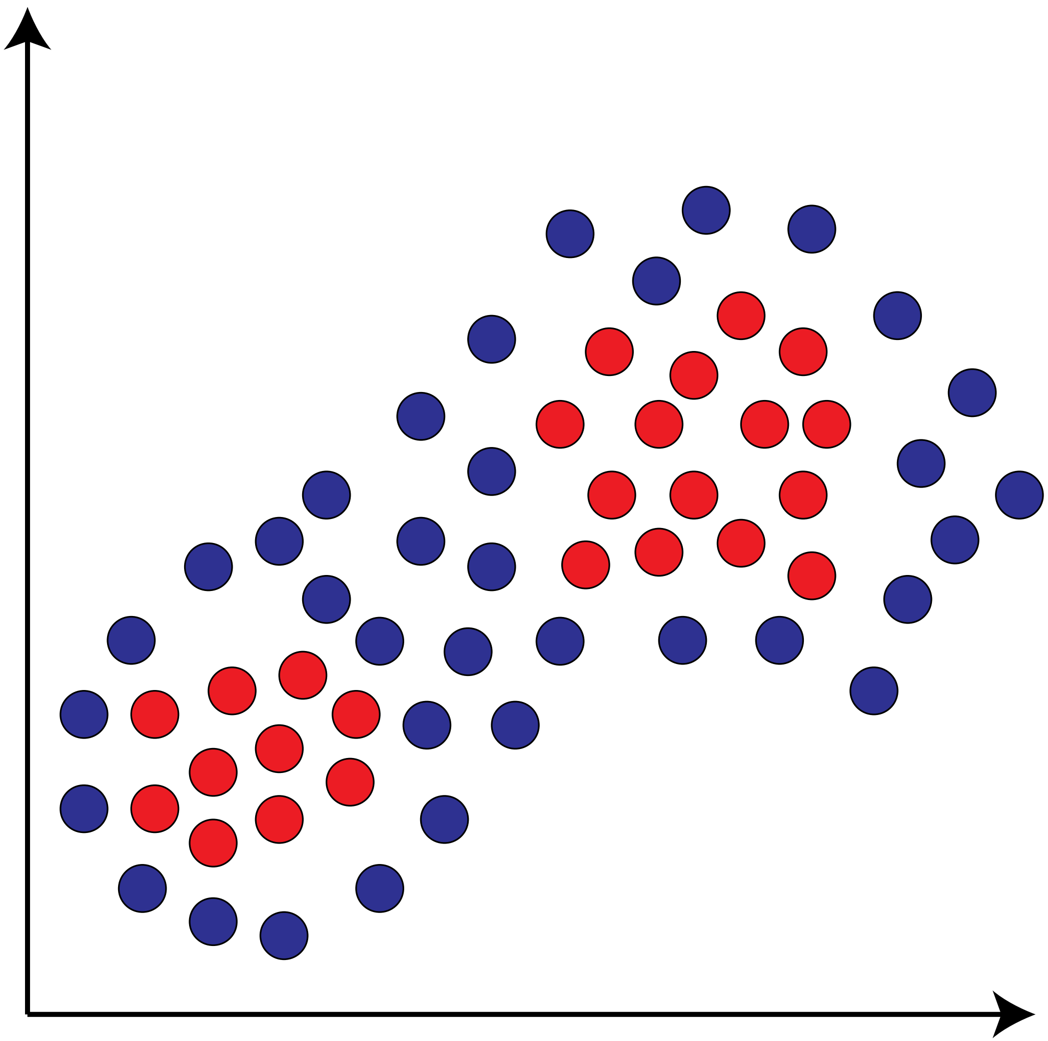
$$z = x^2 + y^2$$



New Data is Linearly Separable!



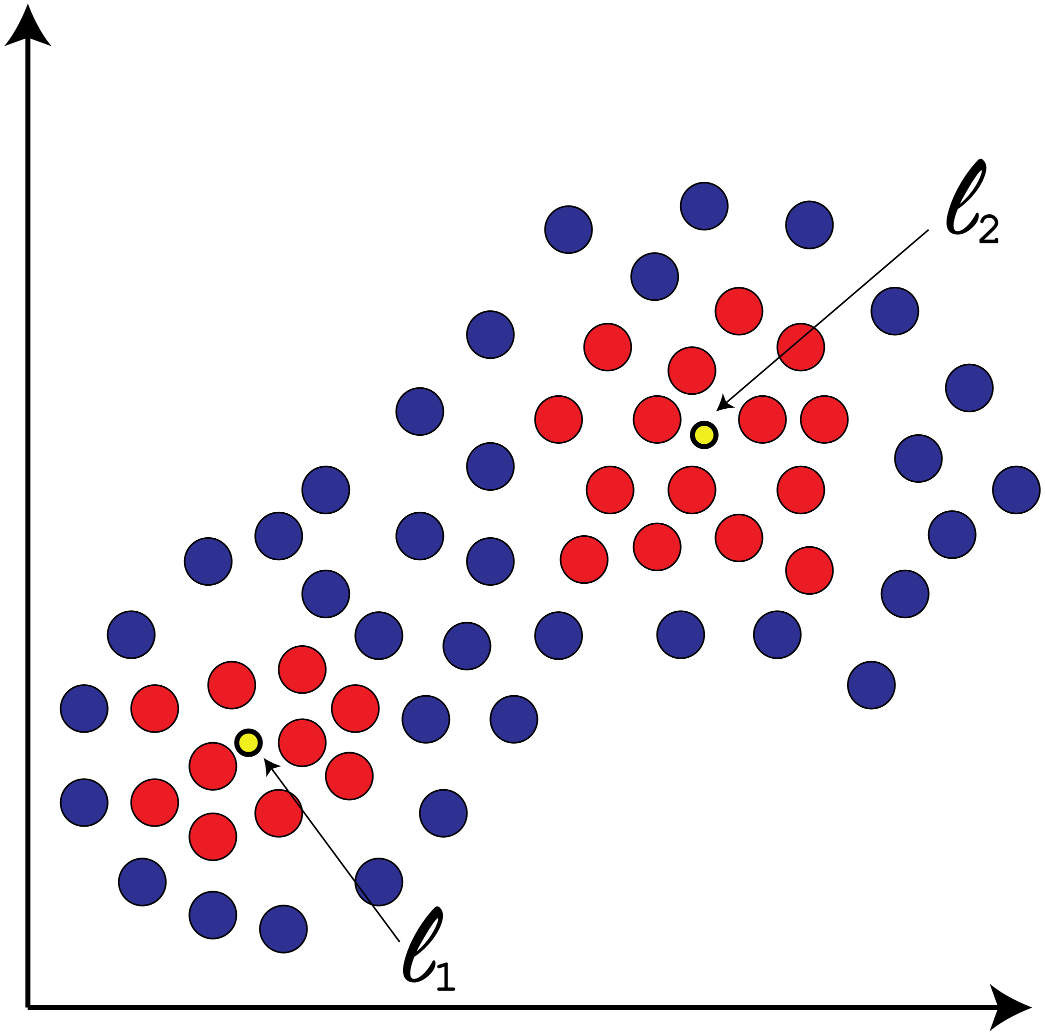
Another view...



The last ‘trick’ seems difficult in this case!

Not immediately clear what transformation will make this data linearly separable.

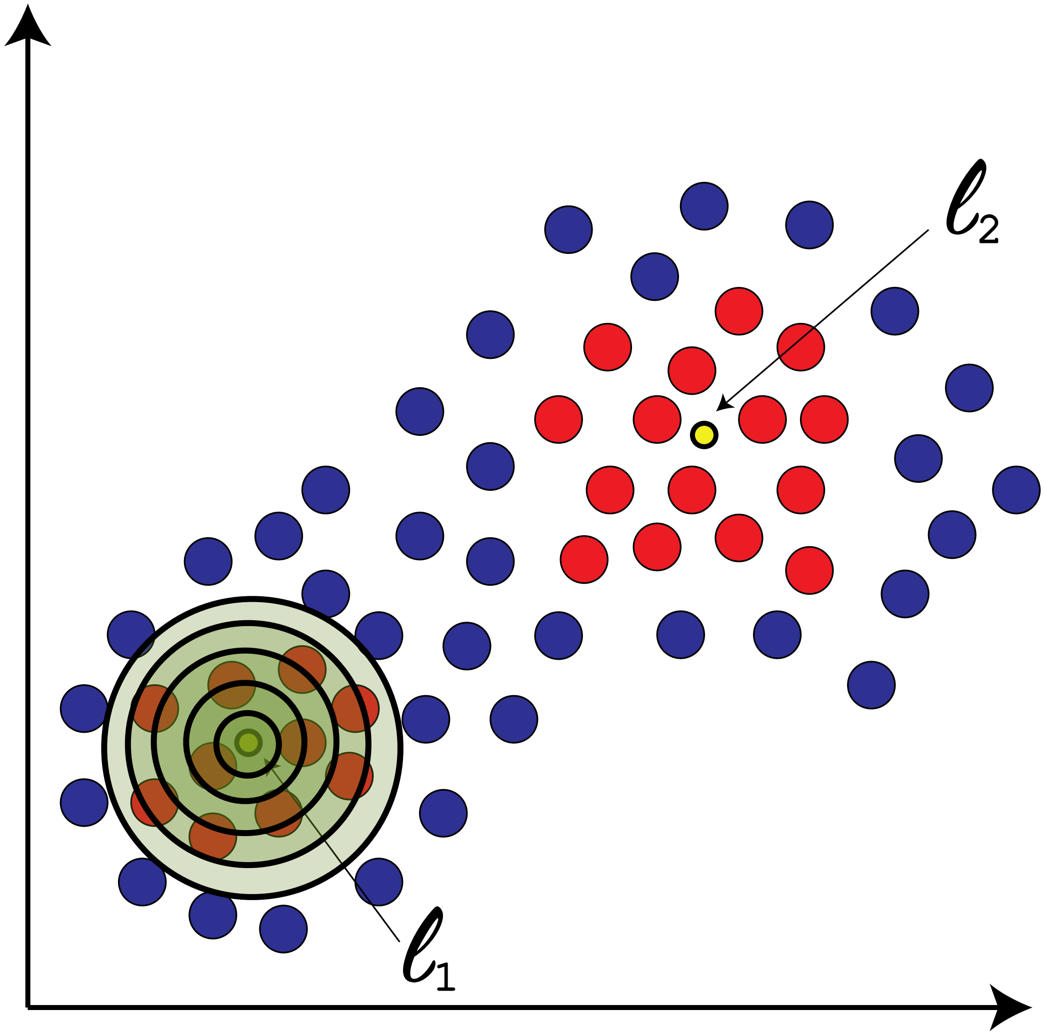
Kernels



Suppose we add two points, which we'll call 'landmarks'.

Then, we create two new variables, f_1 and f_2, which measure the ***similarity*** of each point to those landmarks.

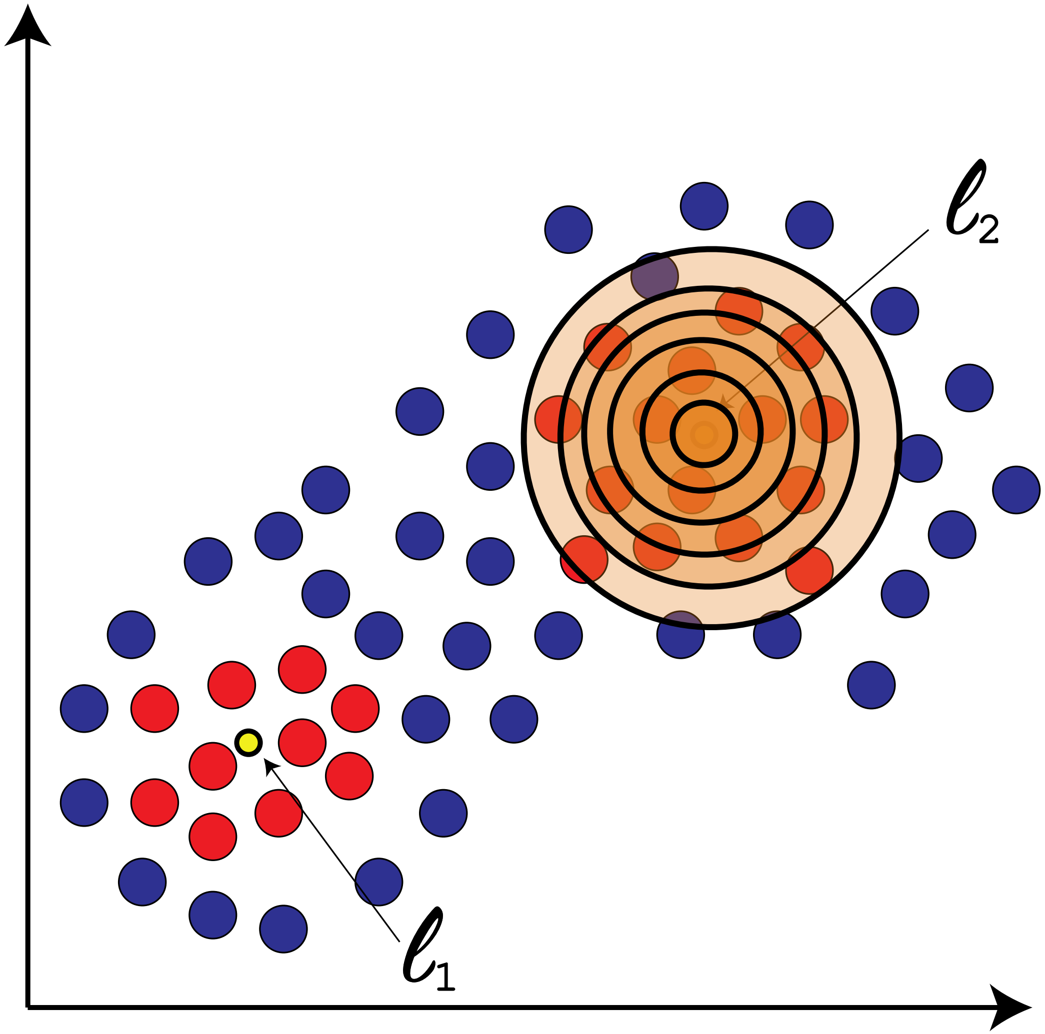
Kernels



f_1 is some measure of similarity (proximity) to l_1

It takes large values near l_1 and small values far from l_1.

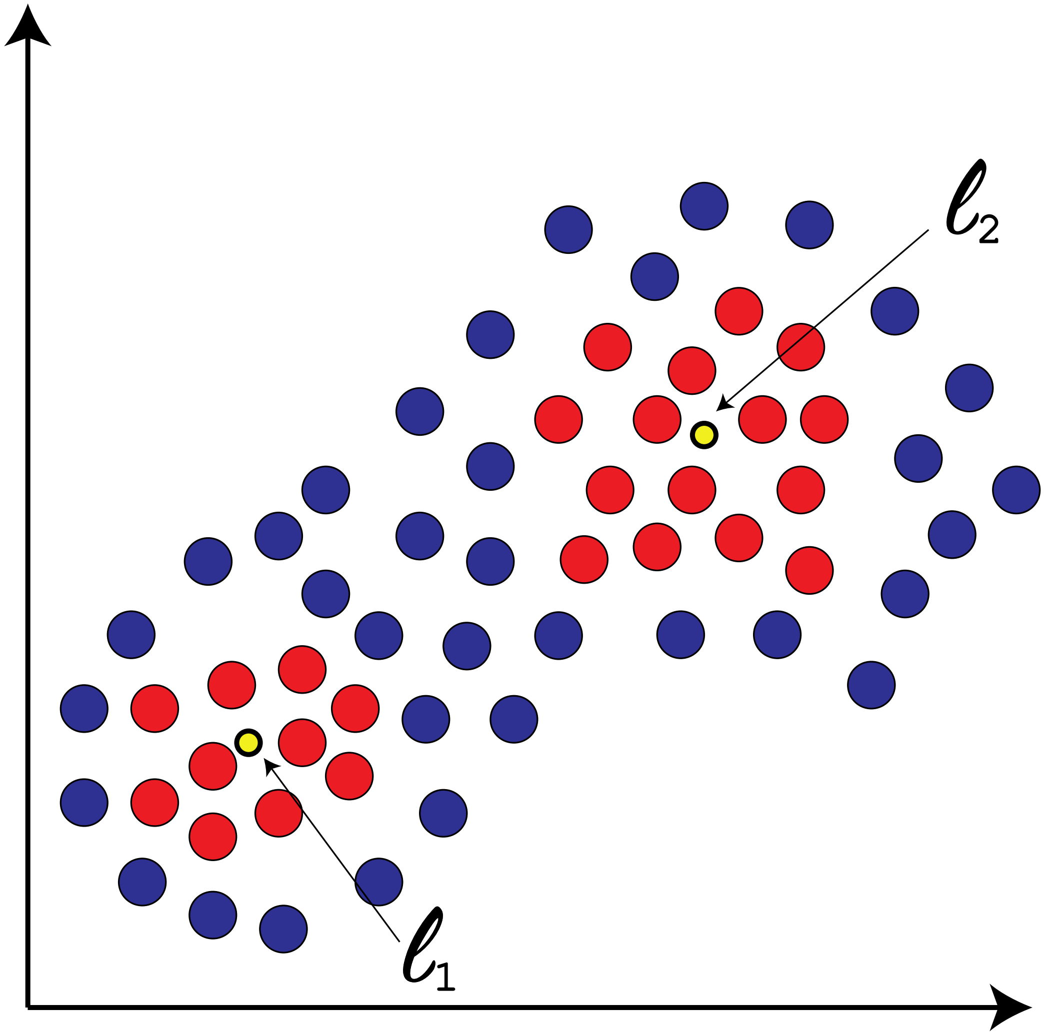
Kernels



f_2 is some measure of similarity (proximity) to l_2

It takes large values near l_2 and small values far from l_2.

Kernels



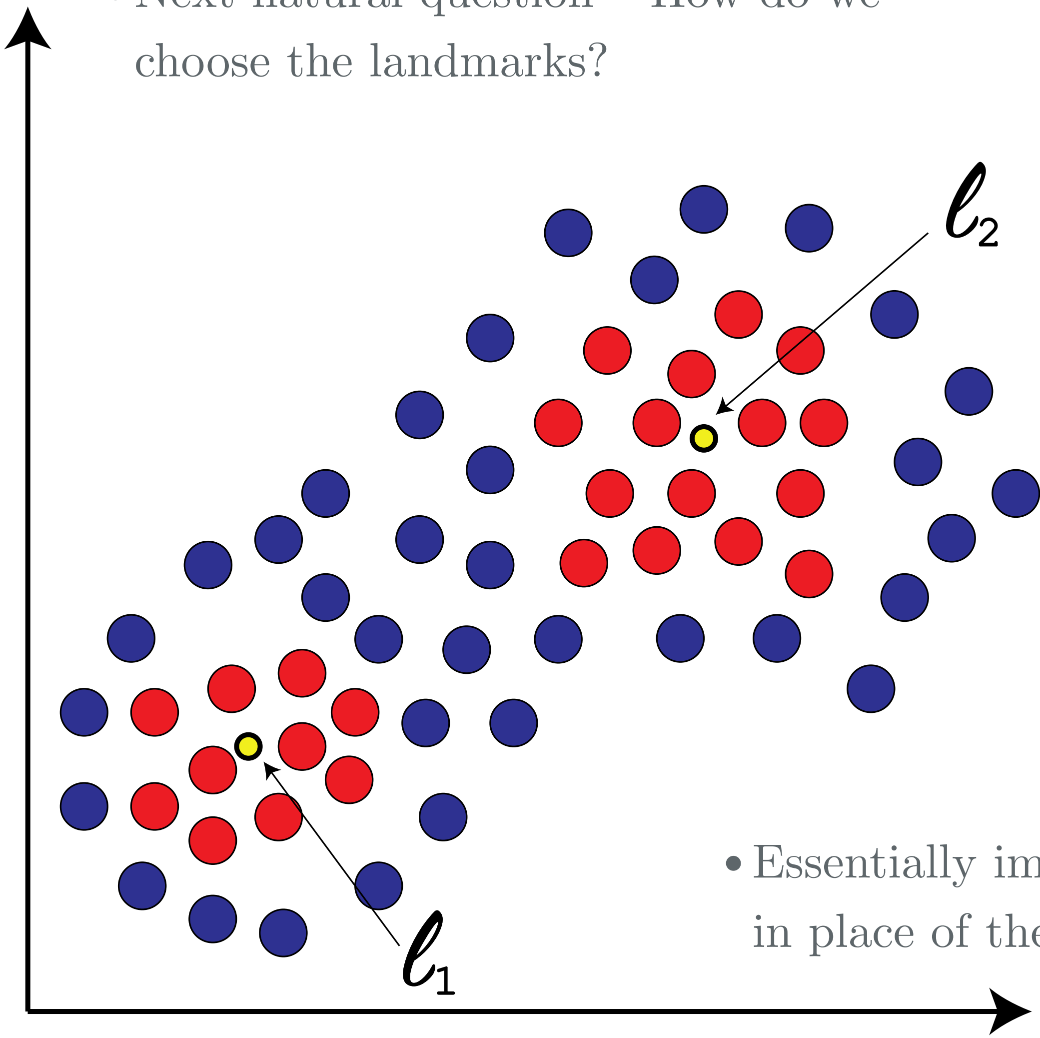
Let's ignore our previous variables (the axis presently shown) and instead use f_1 and f_2.

Where would the red and blue points be located if the axes were f_1 and f_2?

Draw this picture

Kernels

- Next natural question – How do we choose the landmarks?



- You *could* choose a modest number of landmarks (using clustering or other methodology).

- In practice, a **kernel** uses *every* data point as a landmark.

- Essentially implies a similarity matrix to use in place of the data.

Summary of Kernels

- Kernels are similarity functions that measure some kind of proximity between data points.
- Number of data points becomes number of variables
  - This is not great for large datasets!
- SVMs can use kernels without explicitly computing/storing a similarity matrix, but still computationally slow
- Kernels can improve the performance of SVMs in most situations.

Choosing Kernels

- Kernels embed data in a higher dimensional space (implicitly)
- Cannot typically know ahead of time which kernel function will work best (although for text data, linear kernel is highly recommended)
- Can try several, take best performer on validation data

Popular Kernels

- Linear (i.e. no kernel)
- Radial Basis Functions (RBFs)
  - **Gaussian is most common and usually default**

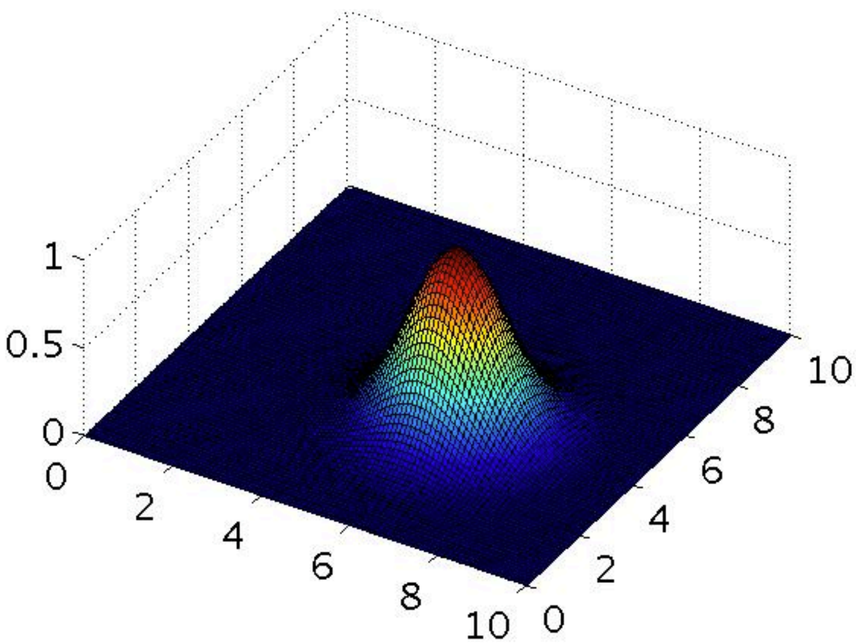
$$e^{\frac{-\|x_i - x_j\|_2}{2\sigma^2}} = e^{-\gamma\|x_i - x_j\|_2}$$

  - $\gamma = \frac{1}{2\sigma^2}$ is hyper parameter controlling shape of function.
  - Some packages want you to specify gamma (γ).
    Some ask you to specify sigma (σ).
  - *NOT good for text classification. Typically linear is best for text*

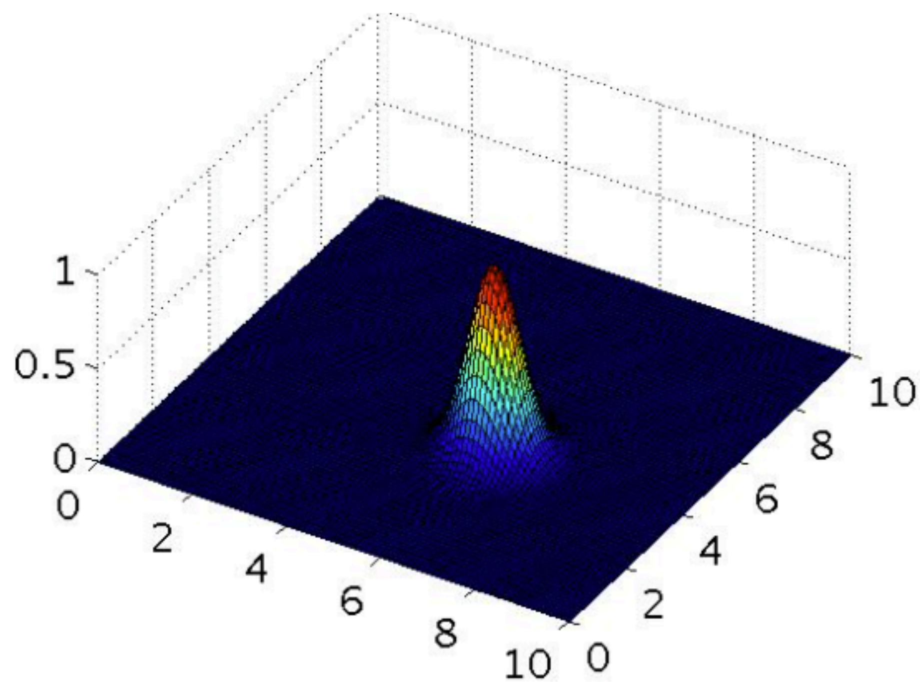
RBF/Gaussian Kernel

$$e^{\frac{-\|x_i - x\|_2}{2\sigma^2}}$$

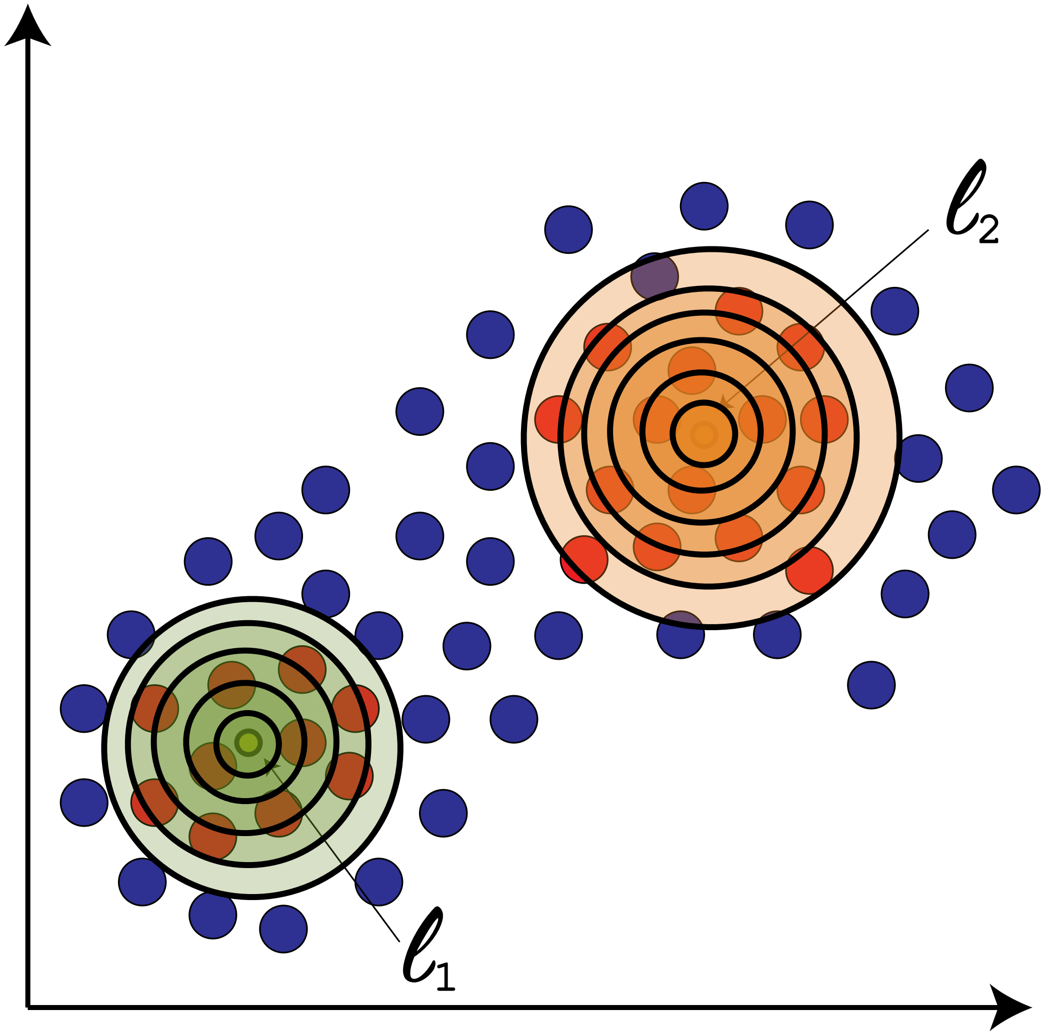
$\sigma = 1$



$\sigma = 0.5$



Kernels

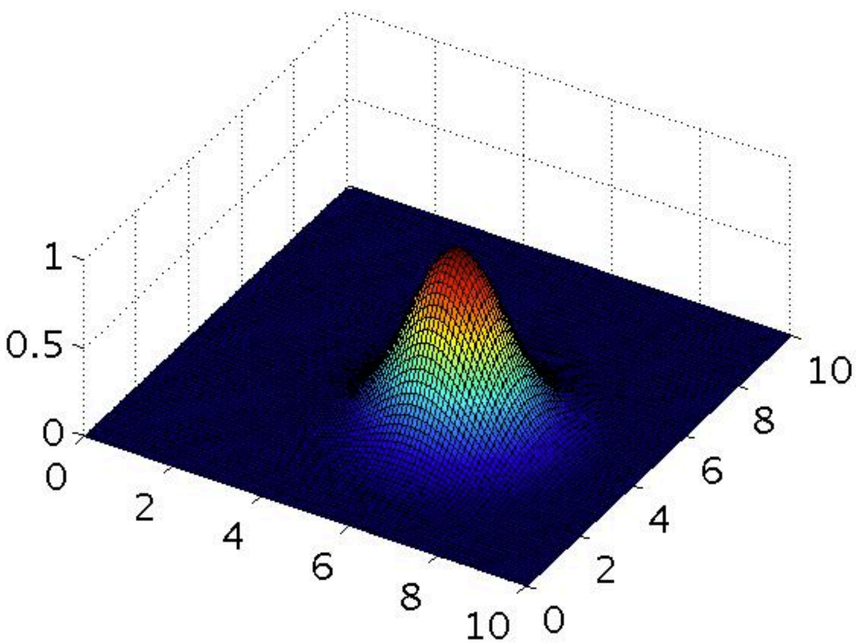


- The circles shown are meant to represent contours of those Gaussian functions.
- For which kernel fuction is $\boldsymbol{\sigma}$ larger, f_1 or f_2?
- (In the actual method, σ is the same for each point)

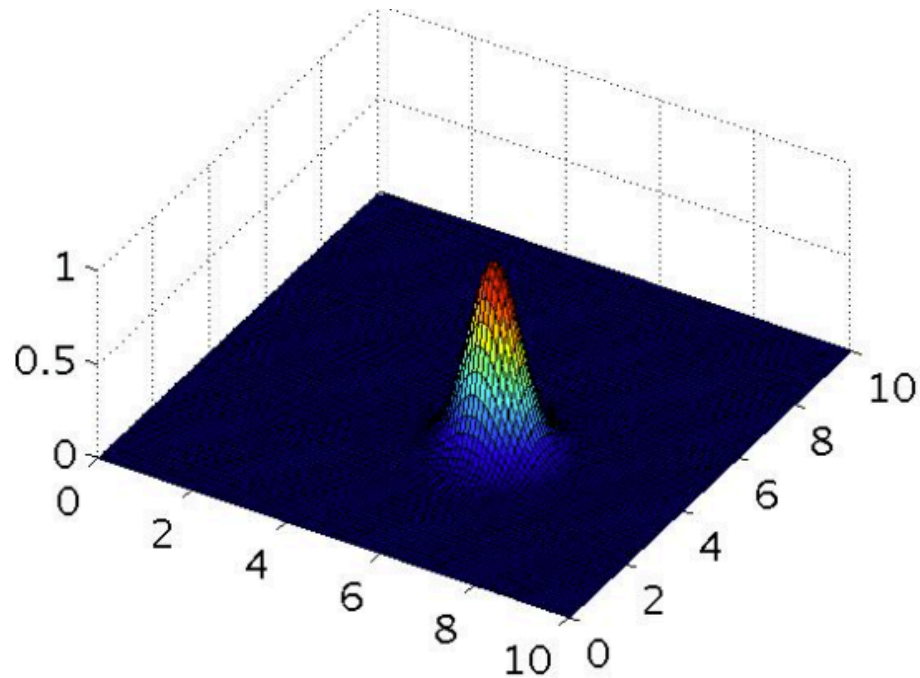
RBF/Gaussian Kernel

$$e^{\frac{-\|x_i - x\|_2}{2\sigma^2}}$$

$\sigma = 1$



$\sigma = 0.5$



Tuning σ

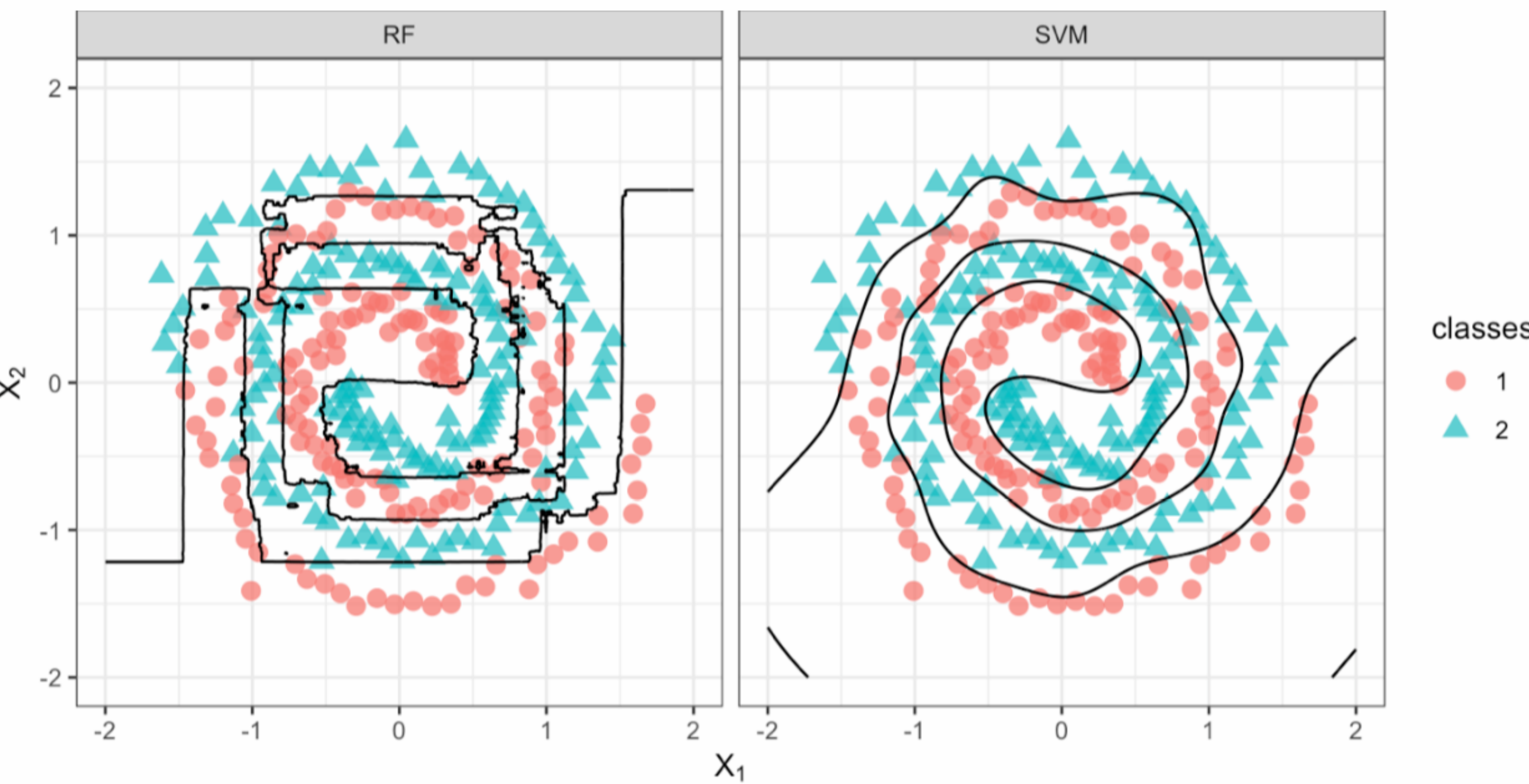
(or equivalently, γ)

- This hyperparameter controls the 'influence' of each training observation.
- A **larger value of σ** (equivalently, a smaller value of γ) means that basis functions are wider – **the influence of a single point is expanded.**
  - Smoother decision boundary => Reduce potential for overfitting.
- A **smaller value of σ** (equivalently, a larger value of γ) means that basis functions are slimmer – **the influence of a single point is diminished.**
  - More localized/jagged decision boundary => Overfitting more likely
  - Consider: if σ were small enough, every point might be identified individually!

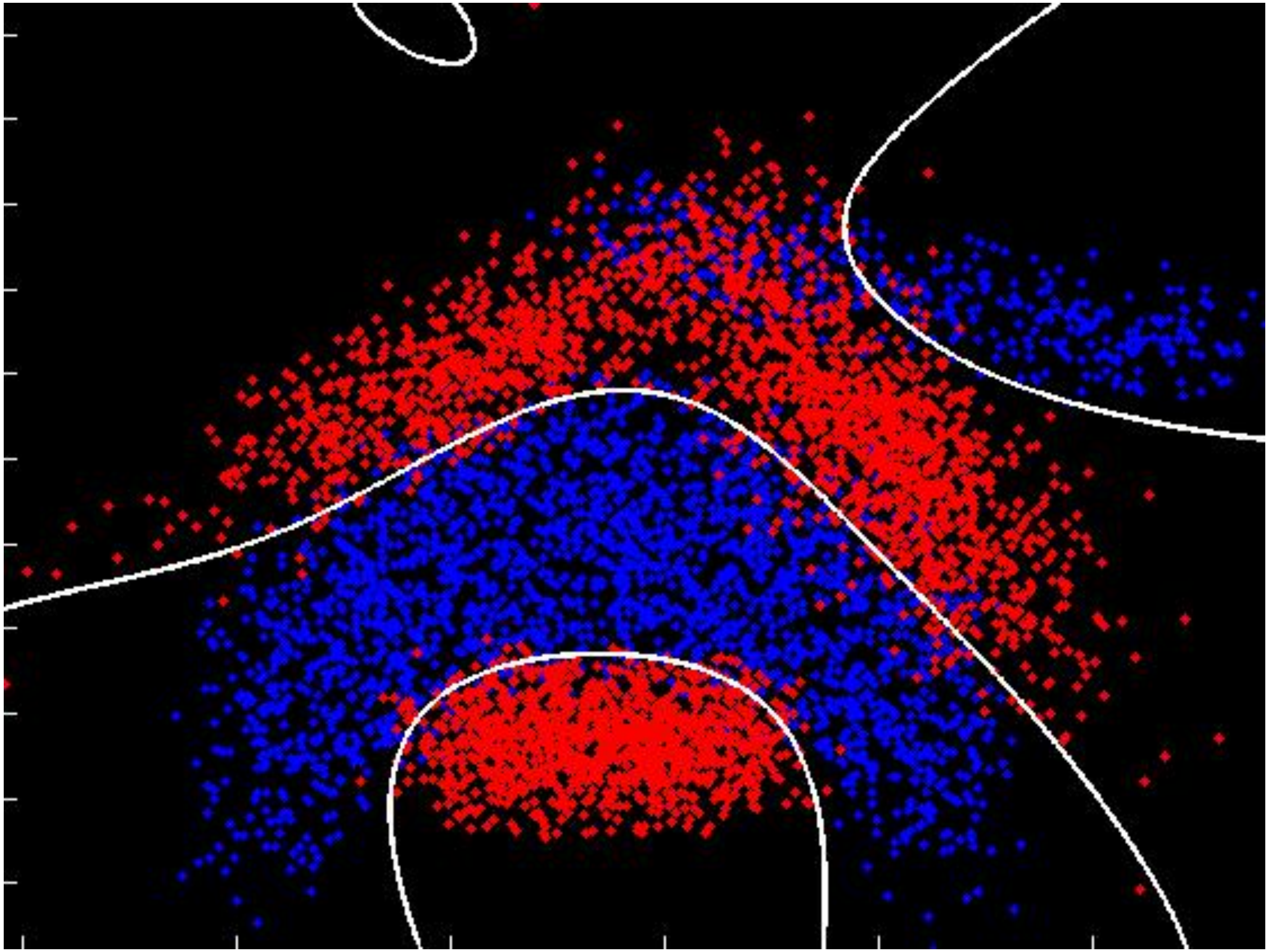
Other Kernels

- Polynomial
  - $\left(ax_i^T x_j + c\right)^d$ where a and c are constants and d is degree of polynomial
- Sigmoid
  - $tanh\left(ax_i^T x_j + c\right)$ where a and c are constants
- Both much less popular than linear/RBF

What kernels can do



What kernels can do



Regularization

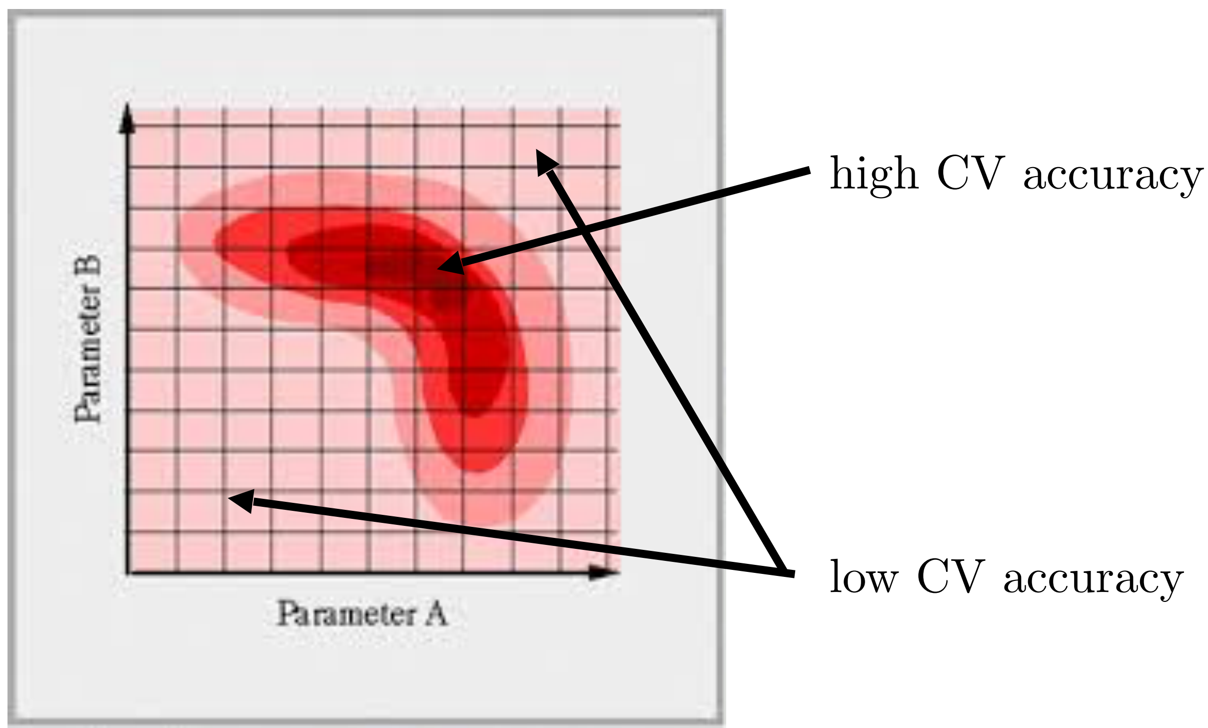
- As with most machine learning algorithms, a *regularization penalty* on $\mathbf{w}$ can be added, $\lambda||\mathbf{w}||$
- Rather than specifying λ, SVMs are coded to expect

$$\mathrm{C} = \frac{1}{\lambda}$$

  - C controls the tradeoff between a smooth decision boundary (bias/underfitting) and classifying training points correctly (variance/overfitting).
  - Larger C aims to classify all training points correctly.
  - Smaller C aims to make decision surface more smooth.

Tuning Hyperparameters

- How do we choose the *specific* values of the hyperparameters σ (or γ) and C?
- One option is a grid search. See how the algorithm performs for all combinations of σ and C within a certain range:



Summary of SVM

Advantages

- Good classifier in large margin situations
- Kernels often work *really* well
- Only requires support vector data points, **memory efficient**
- **Works well with more variables than observations, and with high dimensional data in general**

Summary of SVM

Disadvantages

- **Computationally complex** - large datasets require long training time
- **No variable selection**
- **No variable importance**/interpretability
- **No predicted probabilities** (only "decisions"/classes)
    - Achieved post hoc analysis via logistic regression on the SVM's scores
- Two **hyperparameters to tune**
    - (C or λ) equivalent regularization parameters ($C = \frac{1}{\lambda}$)
    - (γ or σ) equivalent kernel parameters ($\gamma = \frac{1}{2\sigma^2}$)

Extensions of SVMs

• • •

Multiclass classification

Regression

Multiclass Classification with SVM

- Most straightforward approach: **One vs. All (OVA) method**
  - Starting with k classes
  - Train one SVM for each class, separating the points in that class (code as +1) from all other points (code as -1).
  - For SVM on class i, result is a set of parameters $\boldsymbol{w}_i$
  - To classify a new data point $\boldsymbol{d}$, compute $\boldsymbol{w}_i^T\boldsymbol{d}$ and place $\boldsymbol{d}$ in the class for which $\boldsymbol{w}_i^T\boldsymbol{d}$ is largest.

Multiclass Classification with SVM

- Another approach: **One vs. One (OVO) method**
  - Starting with k classes
  - Train one SVM for each pair of classes, separating the points from the two classes.
  - To classify a new data point ***d***, place ***d*** in the class for which it won the most number of pairwise comparisons.

- This is still an ongoing research issue: how to define a larger objective function efficiently to avoid several binary classifiers.
- New methods/packages constantly being developed. Most existing packages *can* handle multiclass targets.

Support Vector Regression

- The methodology behind SVMs has been extended to the regression problem.
- Essentially, the data is imbedded in a very high dimensional space via kernels and then a regression hyperplane is determined via optimization.
- ε-insensitive loss regression - one popular implementation

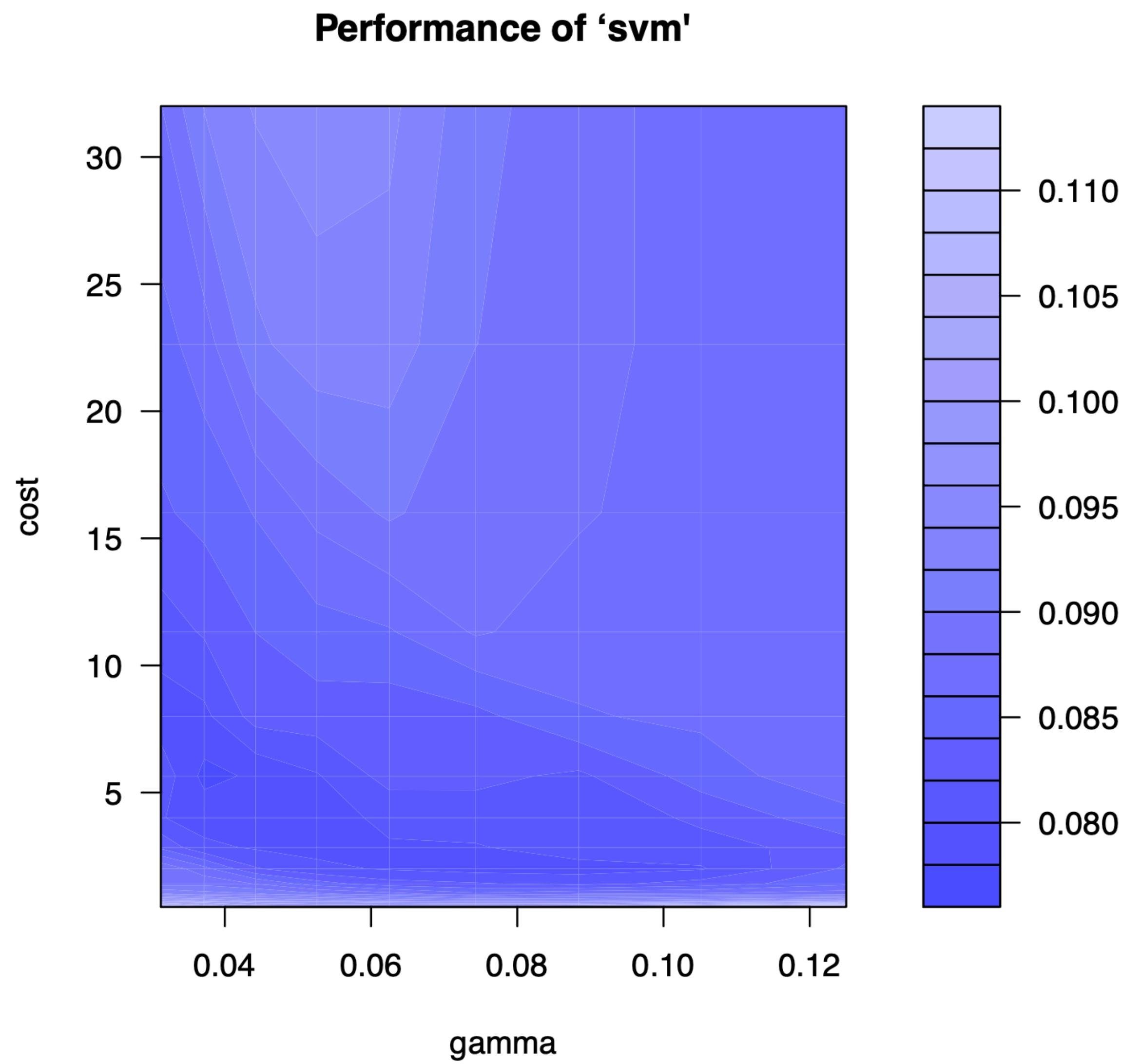
Creating and Tuning an SVM in R

• • •

e1071 library

note: SAS no longer seems to support Radial Basis Functions!

```
library(e1071)
TuneSVM = tune.svm(churn~., data=churnTrainScale,kernel='radial',
                   gamma=2^seq(-5,-3,0.25), cost=2^seq(-1,5,0.5))
summary(TuneSVM)
plot(TuneSVM)
```



Exact Specification of SVM Optimization w/o Kernels

• • •

For those who are interested. When Kernels are introduced, we need more sophisticated math, namely *reproducing kernel Hilbert spaces* = 😩

Optimization Setup - Hard Margin Classifier (HMC)

$$\underset{\mathbf{w}}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \begin{cases} \|\mathbf{w}\| = 1, \\ y_i \left(w_0 + w_1 x_{i1} + \ldots + w_p x_{ip} \right) \geq M, \quad i = 1,2,\ldots,n \end{cases}$$

If $\|\mathbf{w}\| = 1$, this is distance btwn points and hyperplane

Optimization Setup - Soft Margin Classifier (SMC)

$$\underset{\mathbf{w}}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \begin{cases} \|\mathbf{w}\| = 1, \\ y_i\left(w_0 + w_1 x_{i1} + \ldots + w_p x_{ip}\right) \geq M\left(1 - \xi_i\right), \quad i = 1,2,\ldots,n \\ \xi_i \geq 0, \\ \sum_{i=1}^{n} \xi_i \leq C \end{cases}$$