Support Vector Machine

Classification

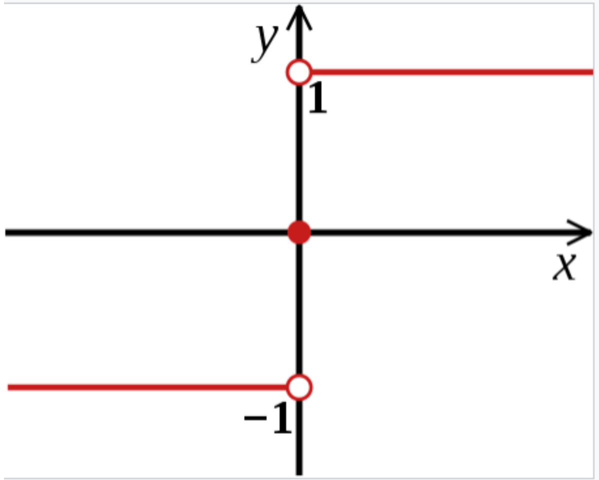
Linear classifiers: how would you separate these two groups of training examples with a straight line?

Feature vector

Class value

$$f(x, w, b) = sign(w \circ x + b)$$

$$= sign\left(\sum w[i]x[i] + \mathrm{b}\right)$$



Signum function $y = \operatorname{sgn} x$

Linear classifiers: how would you separate these two groups of training examples with a line?

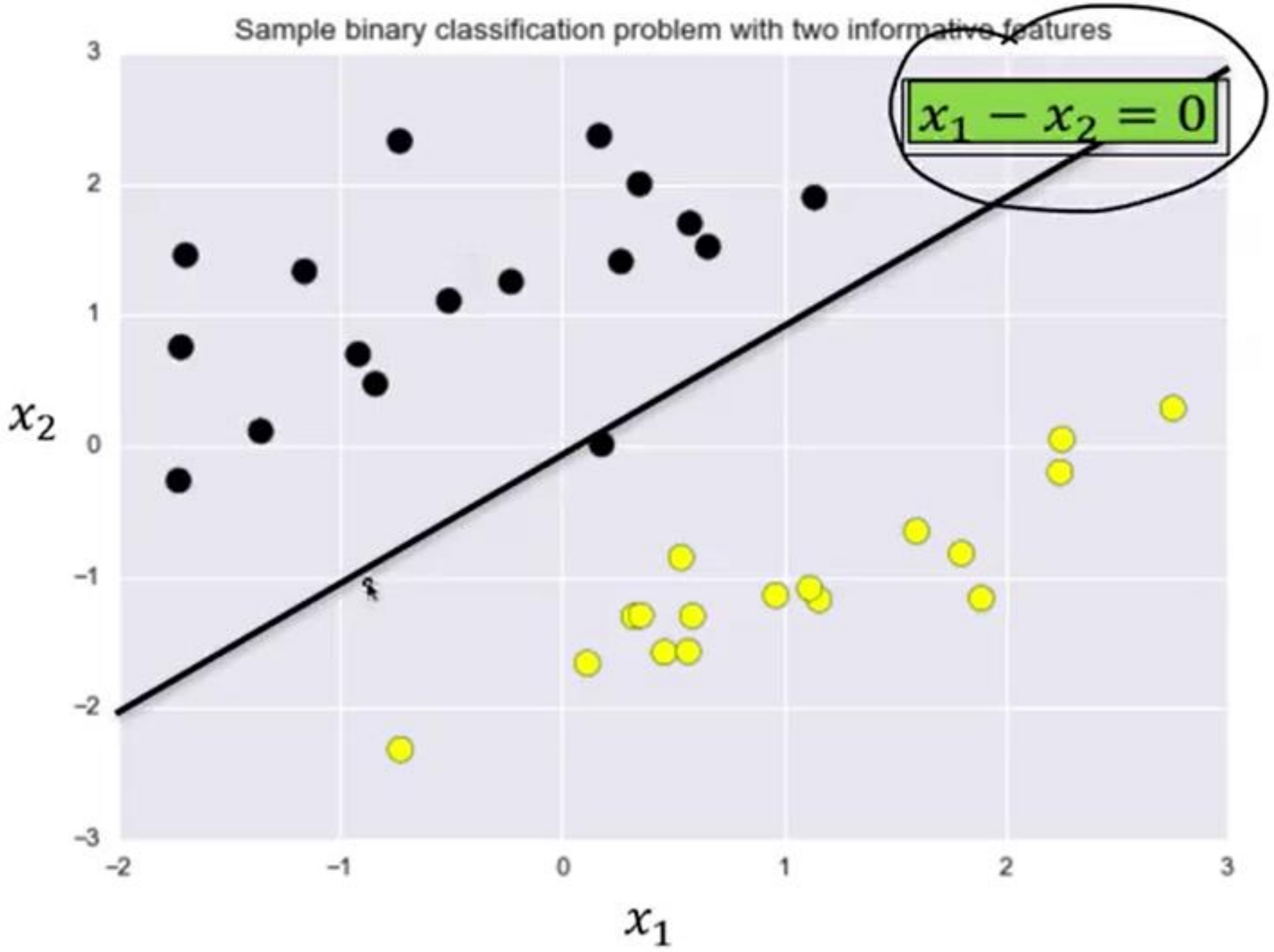
Feature vector

Class value

$$x \rightarrow \boxed{f} \rightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

$$x_1 - x_2 = 0$$
$$w = [1, -1]$$
$$b = 0$$



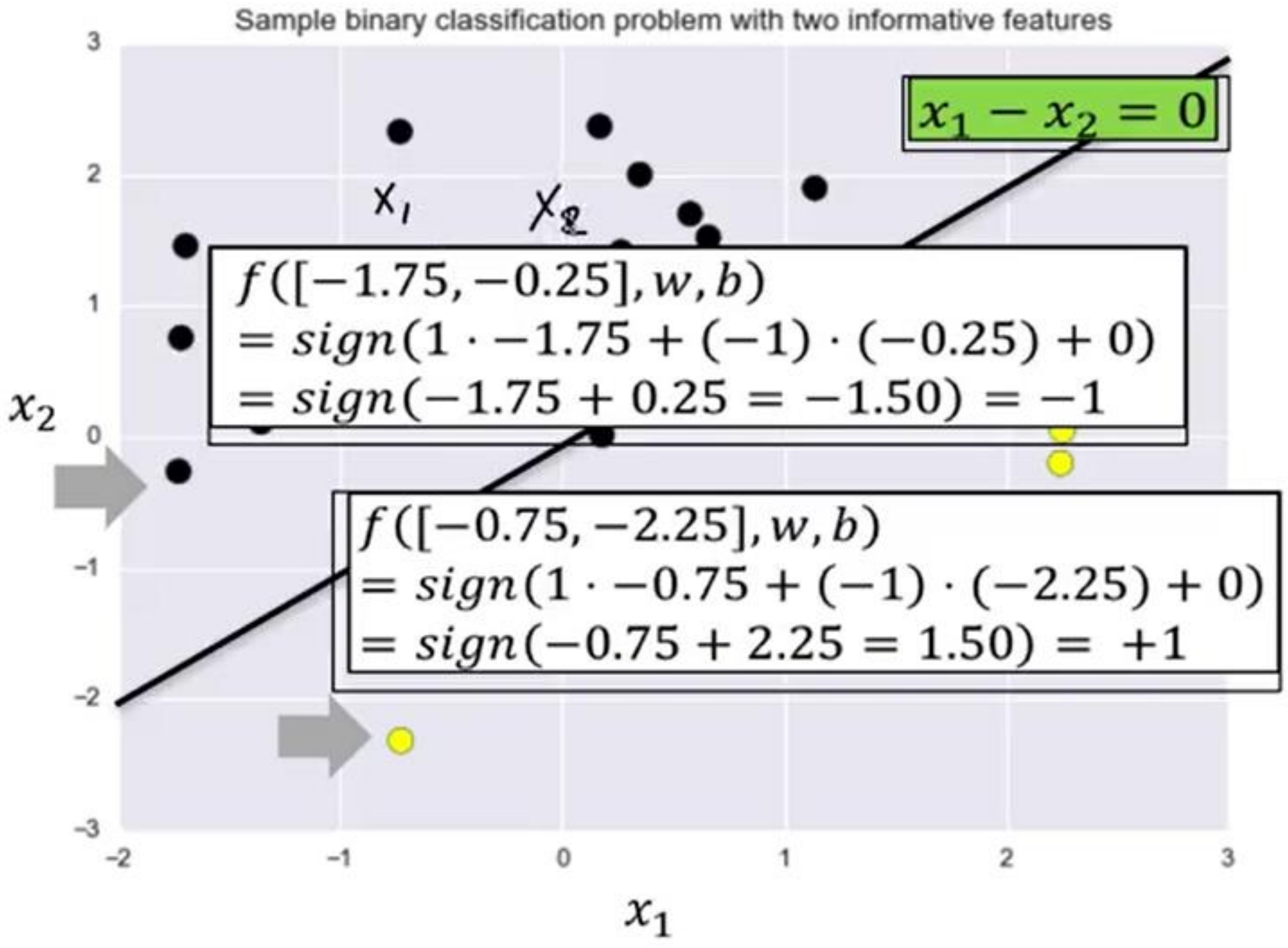
Linear classifiers: how would you separate these two groups of training examples with a line?

Feature vector

Class value

$$f(x, w, b) = sign(w \circ x + b)$$

$$x_1 - x_2 = 0$$
$$w = [1, -1]$$
$$b = 0$$



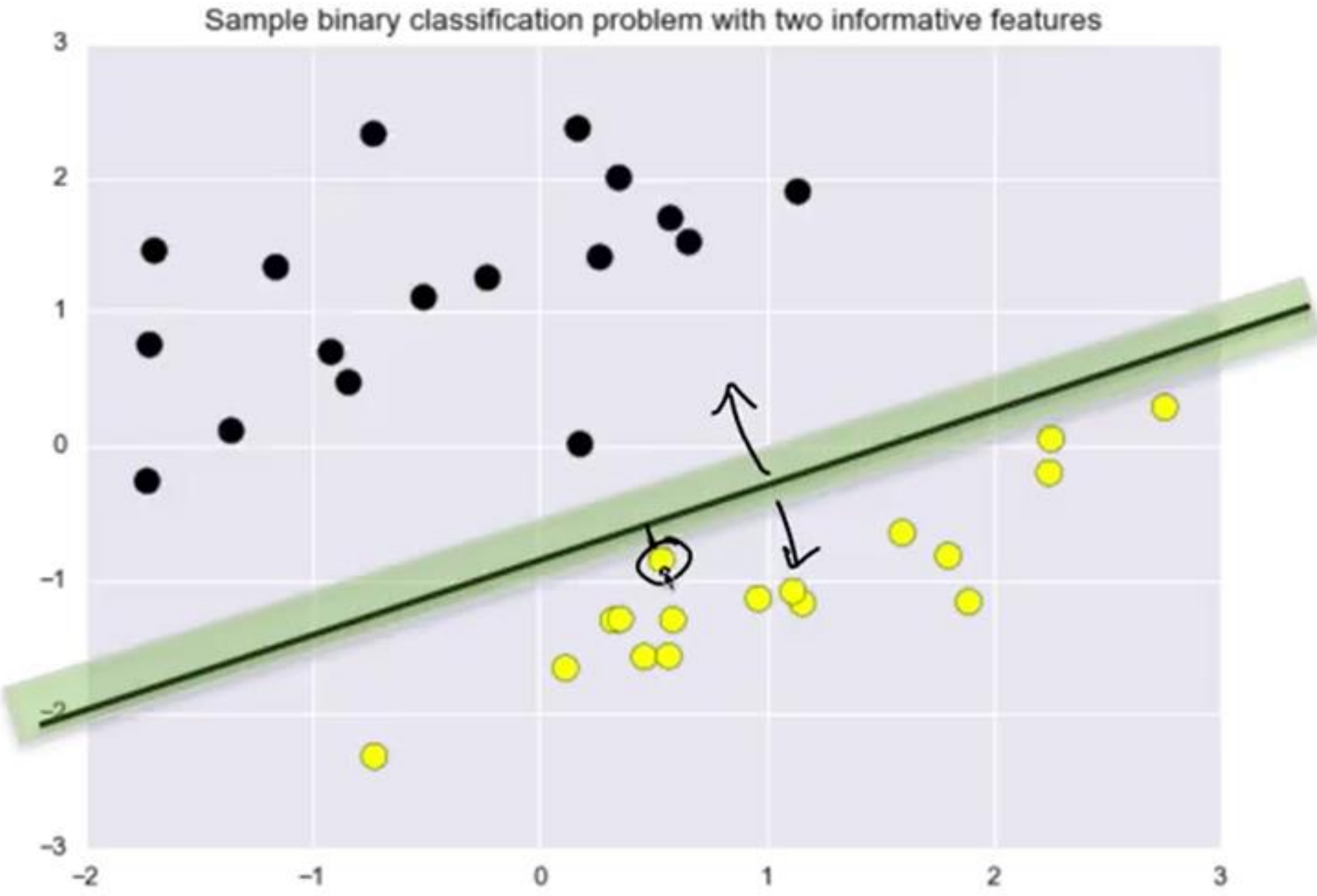
Classifier Margin

$$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

<u>Classifier margin</u>

Defined as the maximum width the decision boundary area can be increased before hitting a data point.



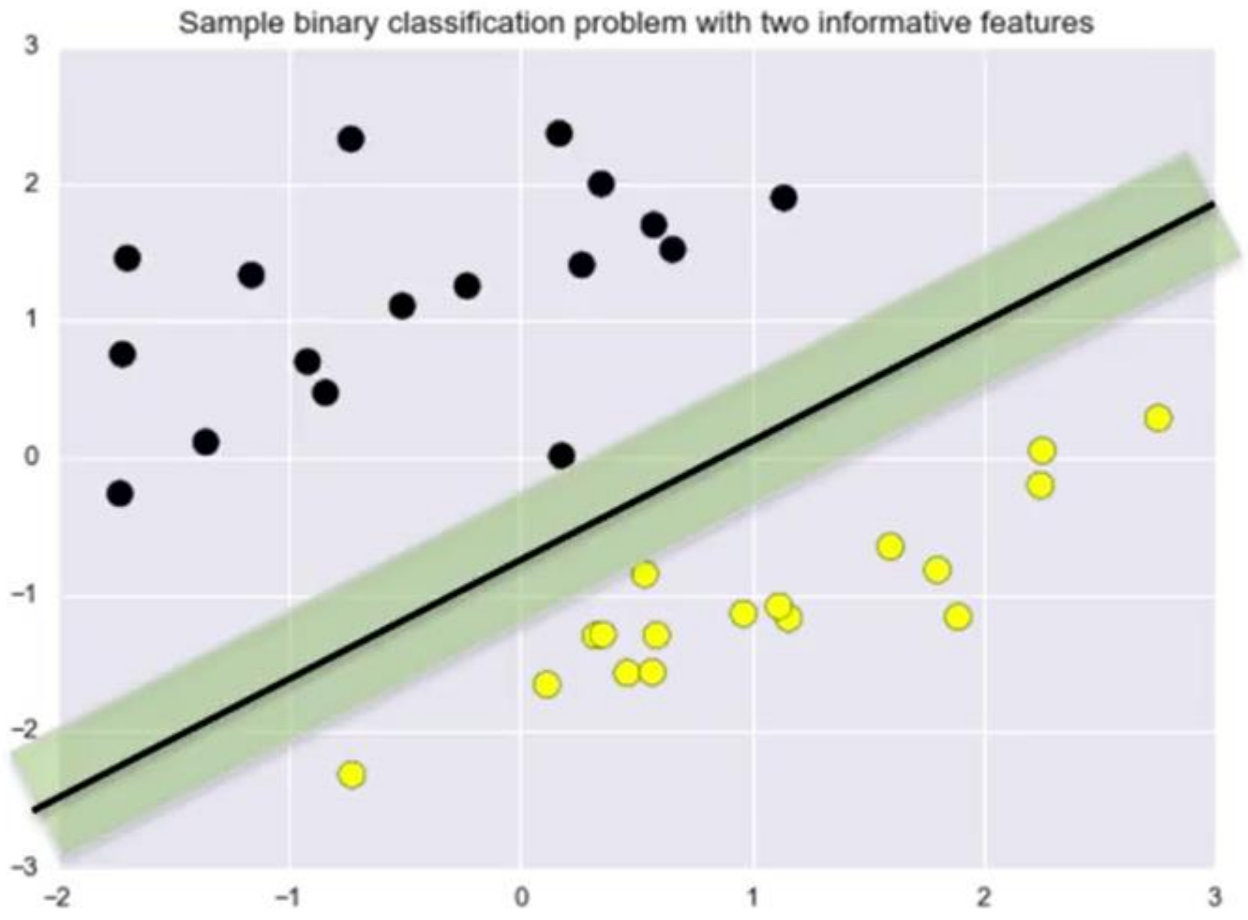
Maximum Margin Linear Classifier: Linear Support Vector Machines

$$f(x, w, b) = sign(w \circ x + b)$$

Maximum margin classifier

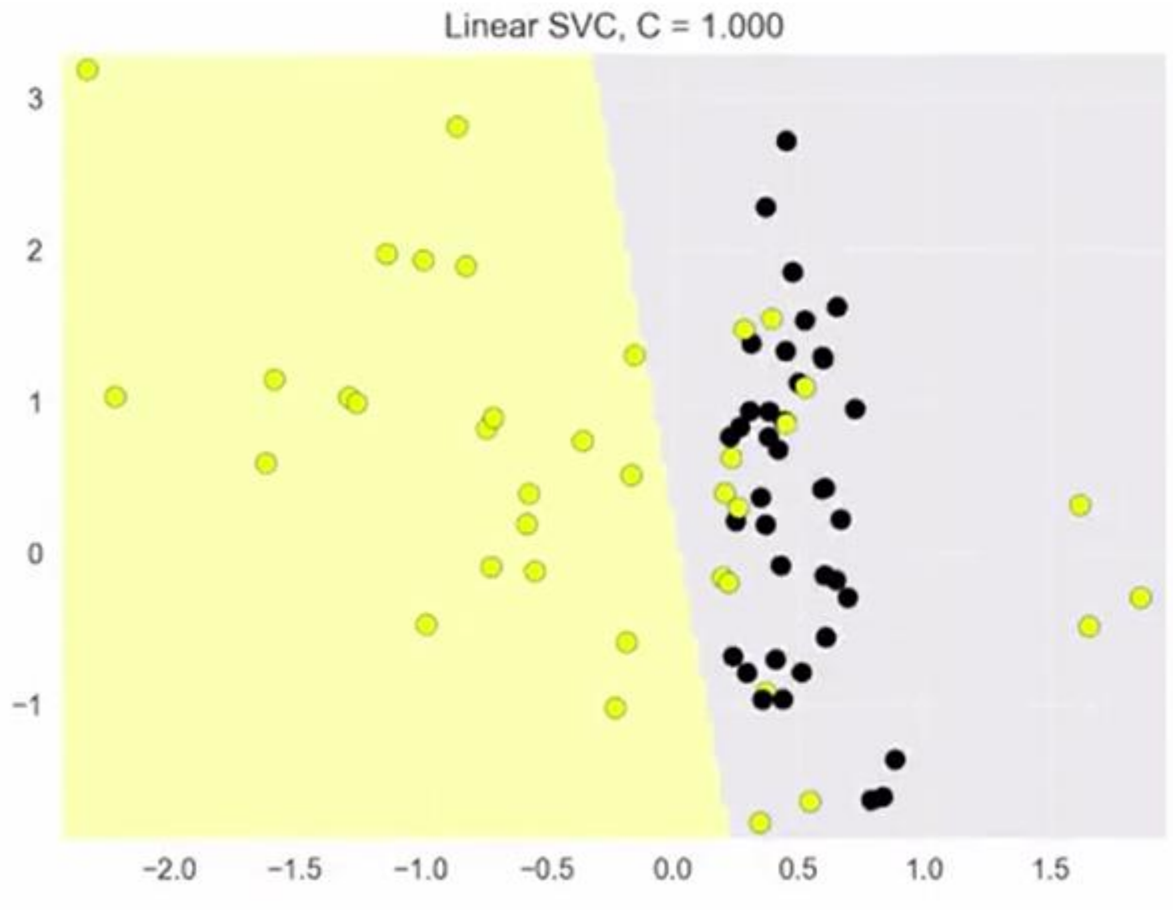
The linear classifier with maximum margin is a linear Support Vector Machine (LSVM).

Or a SVM with linear kernel



Linear Support Vector Machine

```
from sklearn.svm import SVC
from adspy_shared_utilities import (
plot_class_regions_for_classifier_subplot)


X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                    random_state = 0)

fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))
this_C = 1.0
clf = SVC(kernel = 'linear', C=this_C).fit(X_train, y_train)
title = 'Linear SVC, C = {:.3f}'.format(this_C)
plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                         None, None, title, subaxes)
```


Linear SVC, C = 1.000
3
2
1
0
−1
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

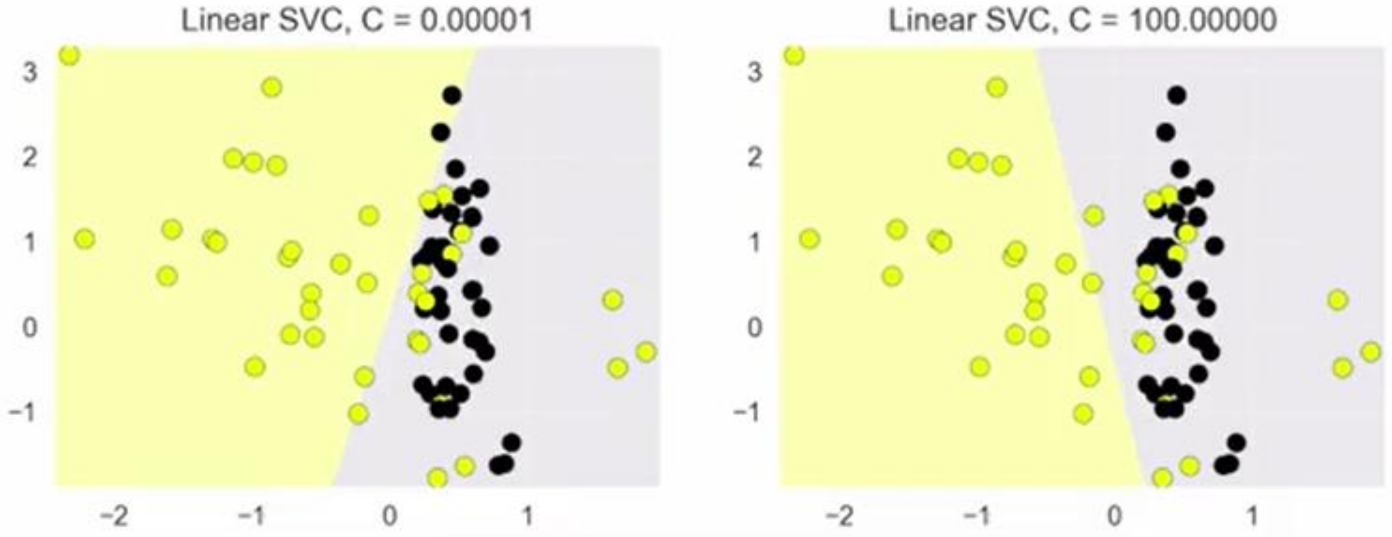
Regularization for SVMs: the C parameter

- The strength of regularization is determined by C
- Larger values of C: less regularization
  - *Fit the training data as well as possible*
  - *Each individual data point is important to classify correctly*
- Smaller values of C: more regularization
  - *More tolerant of errors on individual data points*

In [21]:

```
from sklearn.svm import LinearSVC
from adspy_shared_utilities import plot_class_regions_for_classifier

X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                    random_state = 0)
fig, subaxes = plt.subplots(1, 2, figsize=(9, 3))

for this_C, subplot in zip([0.00001, 100], subaxes):
    clf = LinearSVC(C=this_C).fit(X_train, y_train)
    title = 'Linear SVC, C = {:.5f}'.format(this_C)
    plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                              None, None, title, subplot)
```

Figure 7



Linear Models: Pros and Cons

Pros:

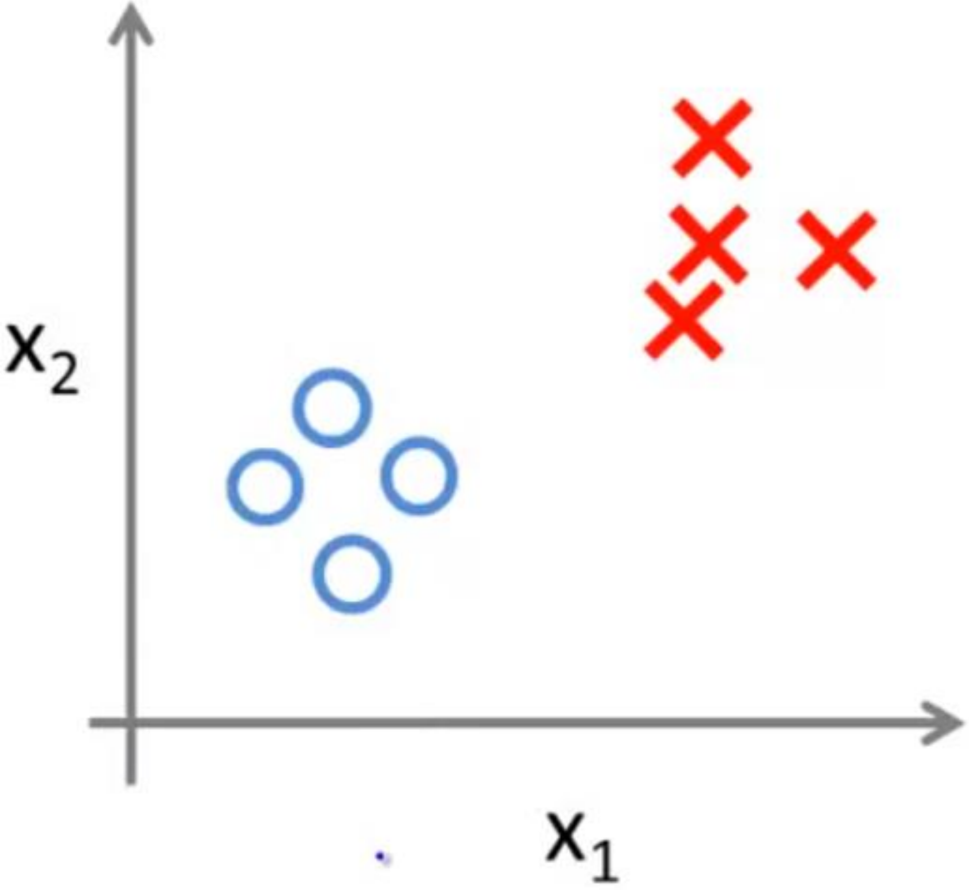
- Simple and easy to train.
- Fast prediction.
- Scales well to very large datasets.
- Works well with sparse data.
- Reasons for prediction are relatively easy to interpret.

Cons:

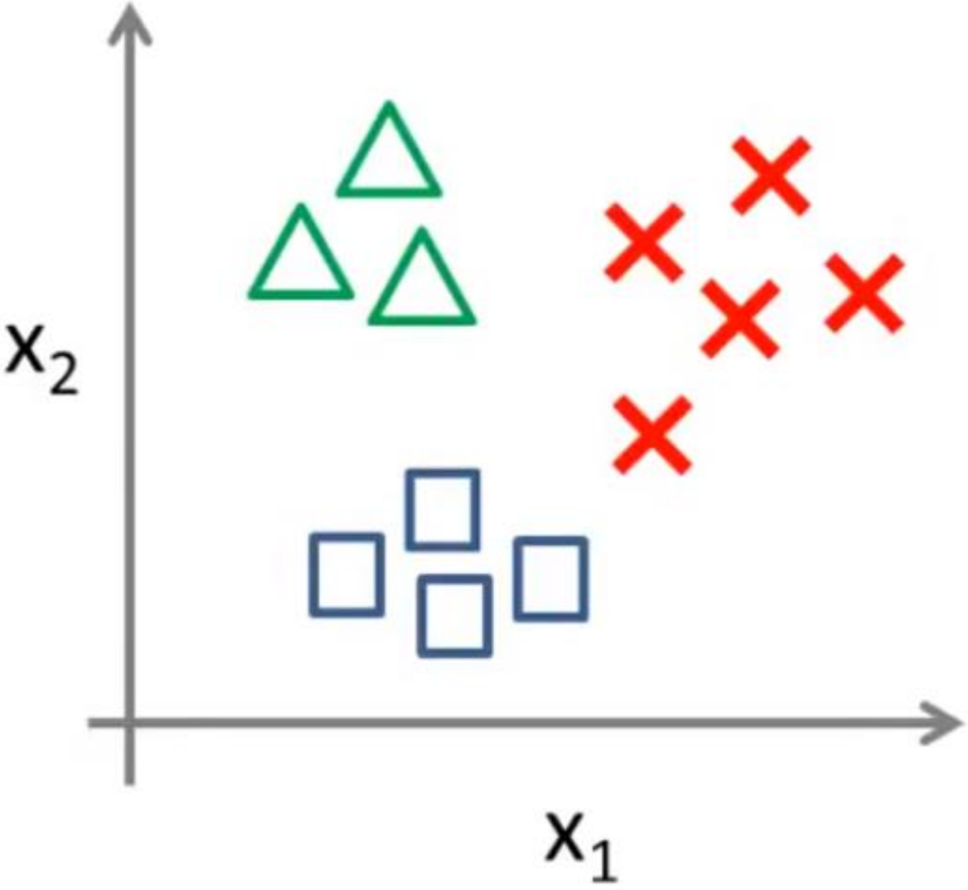
- For lower-dimensional data, other models may have superior generalization performance.
- For classification, data may not be linearly separable (more on this in SVMs with non-linear kernels)

SVM – Multi-class Classification

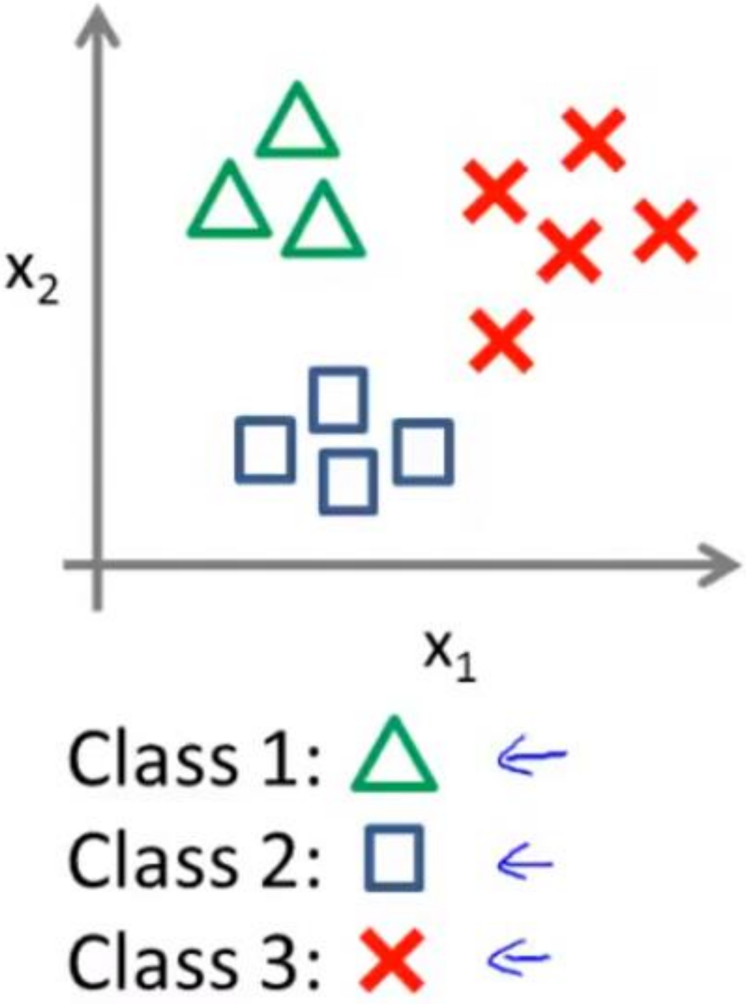
Binary classification:

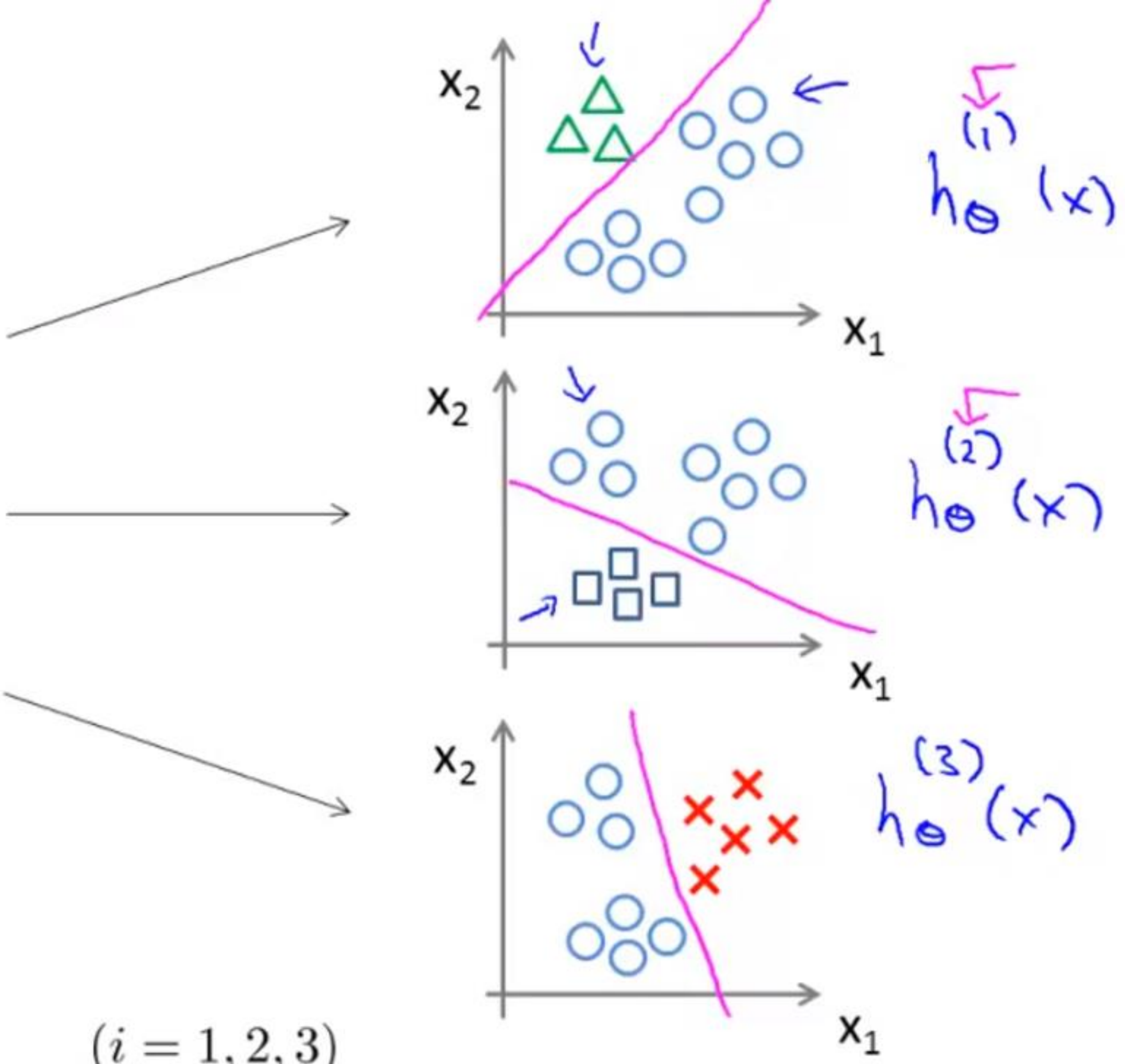


Multi-class classification:



One-vs-all (one-vs-rest):



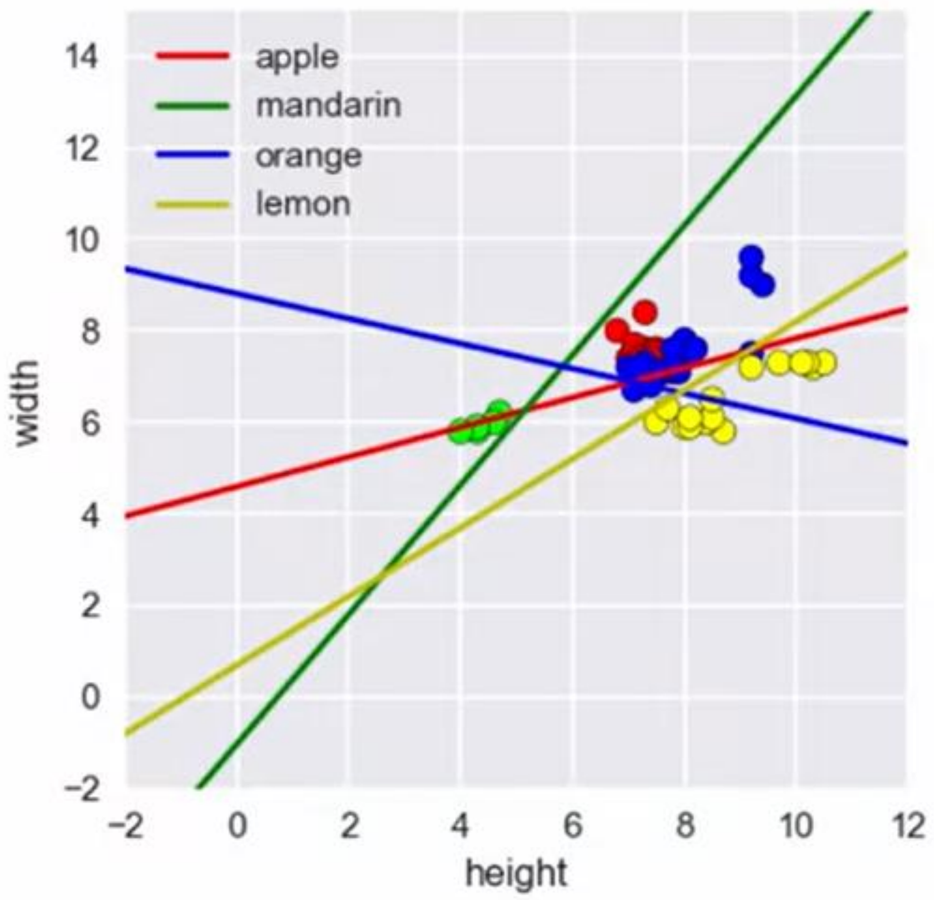


$$h_\theta^{(i)}(x) = P(y = i | x; \theta) \qquad (i = 1, 2, 3)$$

Andrew Ng

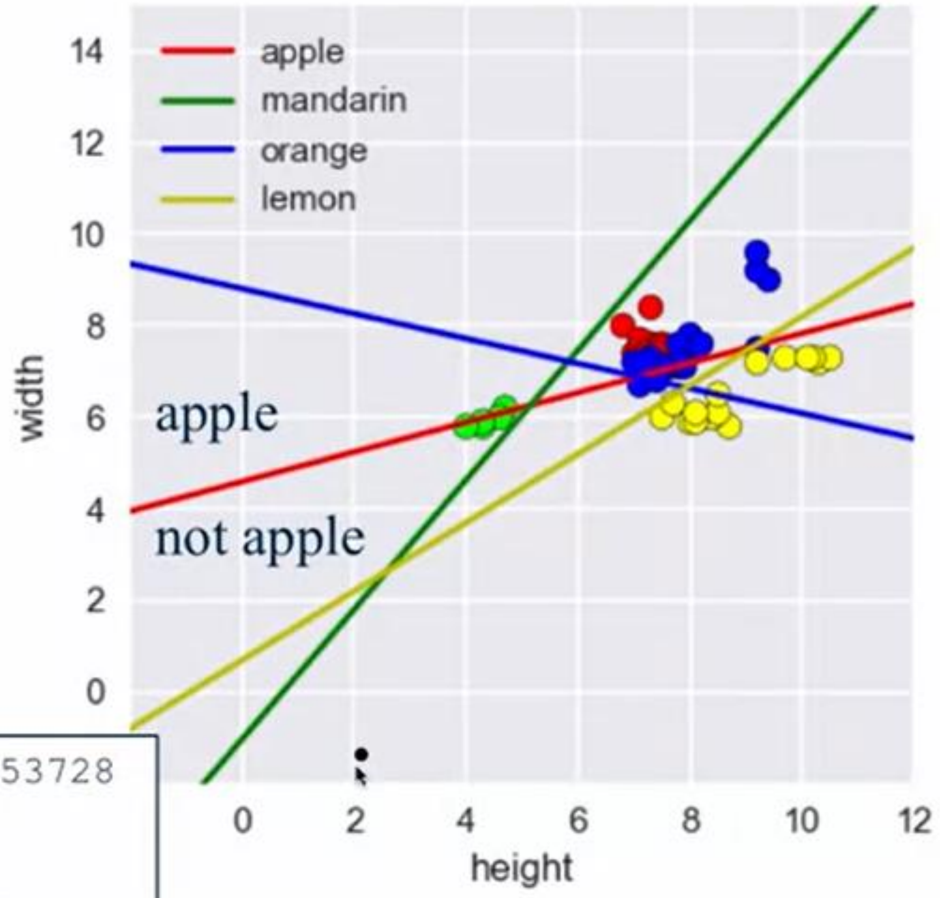
Multi-class Classification with Linear Models

```
clf = LinearSVC(C=5, random_state = 67)
clf.fit(X_train, y_train)

print(clf.coef_)

[[-0.23401135  0.72246132]
 [-1.63231901  1.15222281]
 [ 0.0849835   0.31186707]
 [ 1.26189663 -1.68097   ]]

print(clf.intercept_)
[-3.31753728  1.19645936 -2.7468353  1.16107418]
```



Multi-class Classification with Linear Models

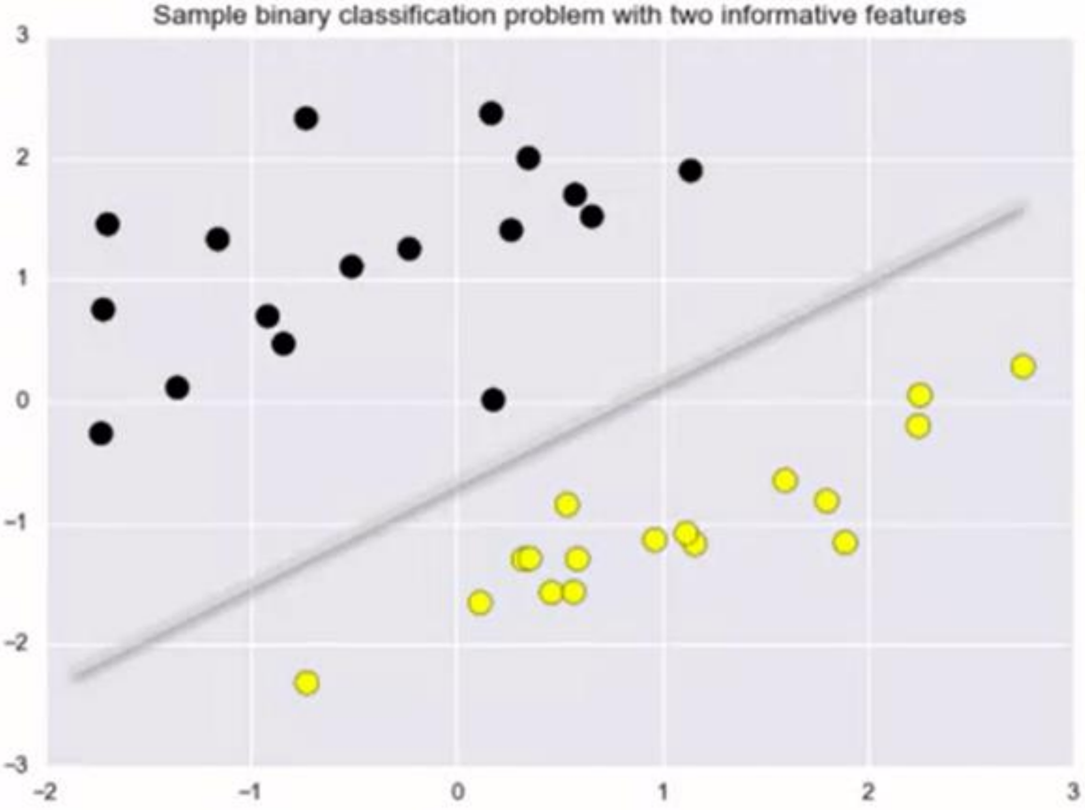
```
clf = LinearSVC(C=5, random_state = 67)
clf.fit(X_train, y_train)

print(clf.coef_)

[[-0.23401135  0.72246132]
 [-1.63231901  1.15222281]
 [ 0.0849835   0.31186707]
 [ 1.26189663 -1.68097   ]]

print(clf.intercept_)
[-3.31753728  1.19645936 -2.7468353  1.16107418]
```




```
y_apple = -0.23401135 * height + 0.72246132 * width - 3.31753728

height=2, width=6:  y_apple = + 0.549 (>= 0: predict apple)
height=2, width=2:  y_apple = - 2.340 (< 0:  predict other)
```

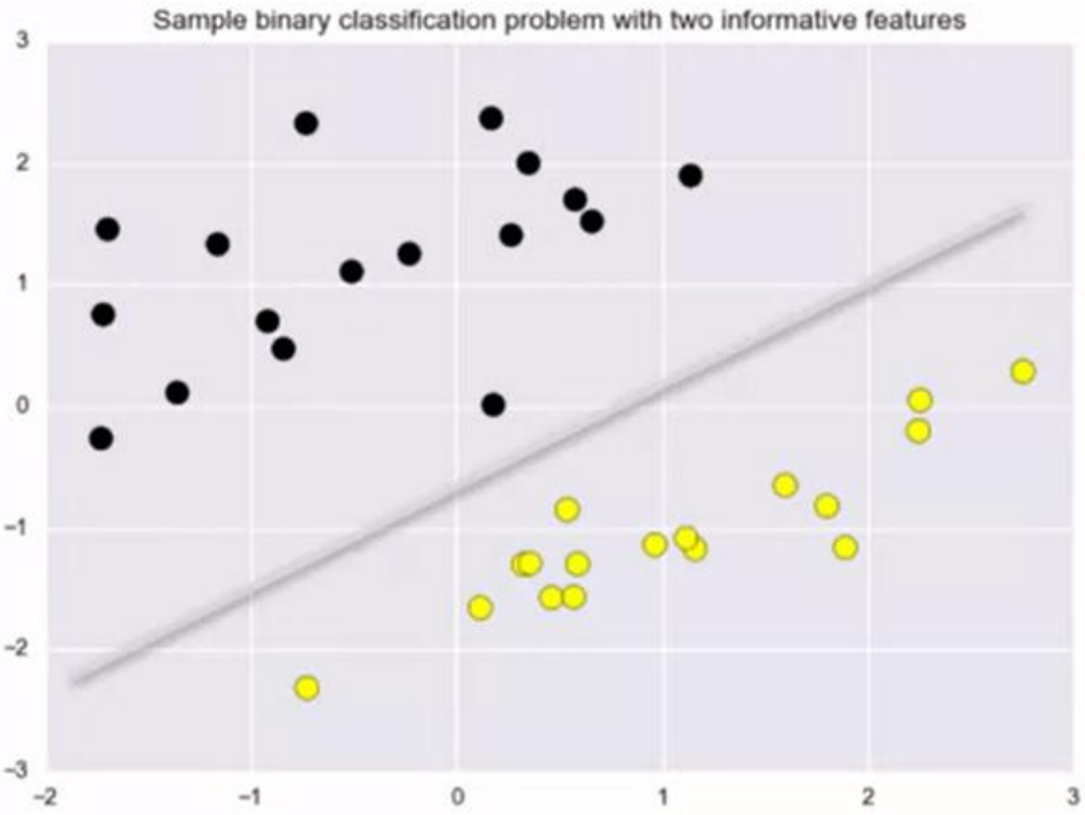
Kernelized SVM (non-linear SVM)

We saw how linear support vector classifiers could effectively find a decision boundary with maximum margin

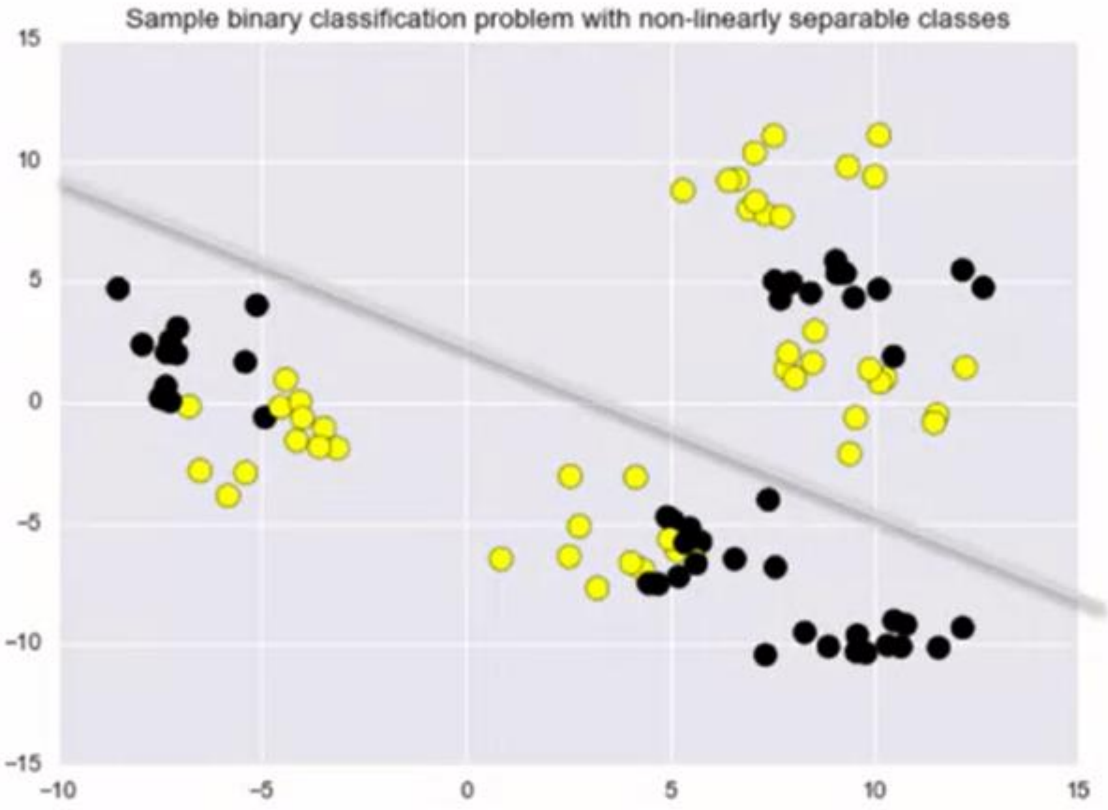


Easy for a linear classifier

But what about more complex binary classification problems?

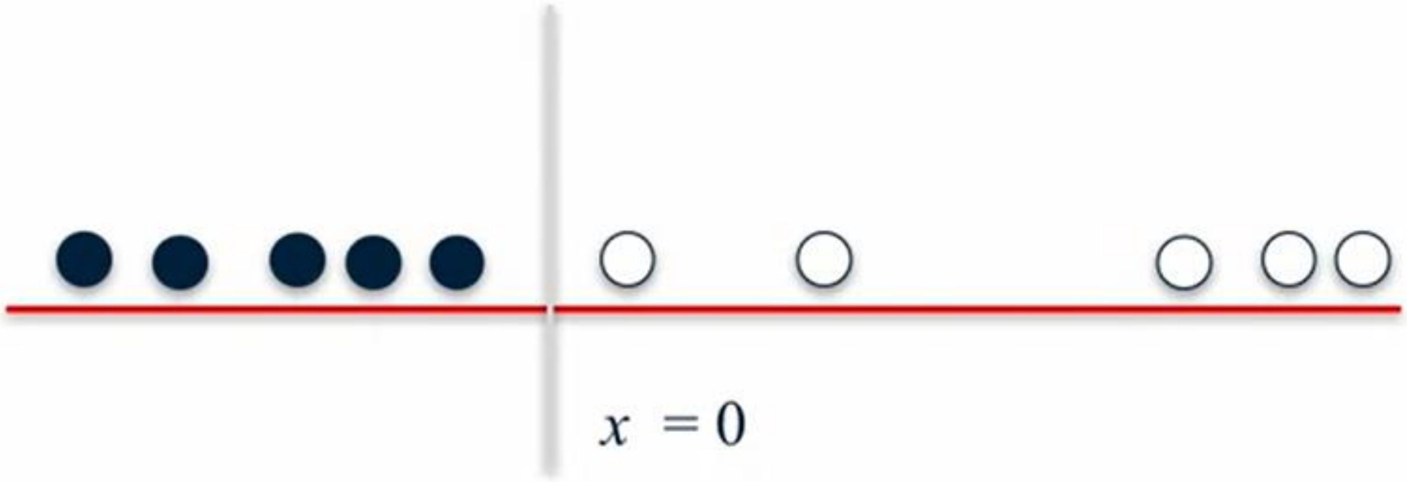


Easy for a linear classifier

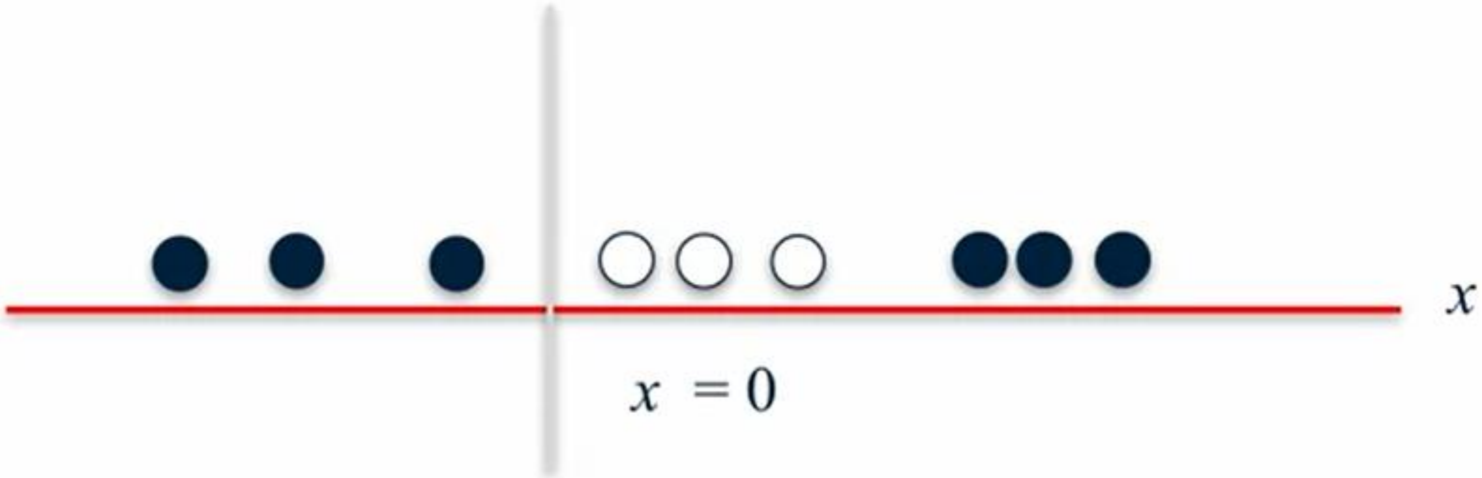


Difficult/impossible for a linear classifier

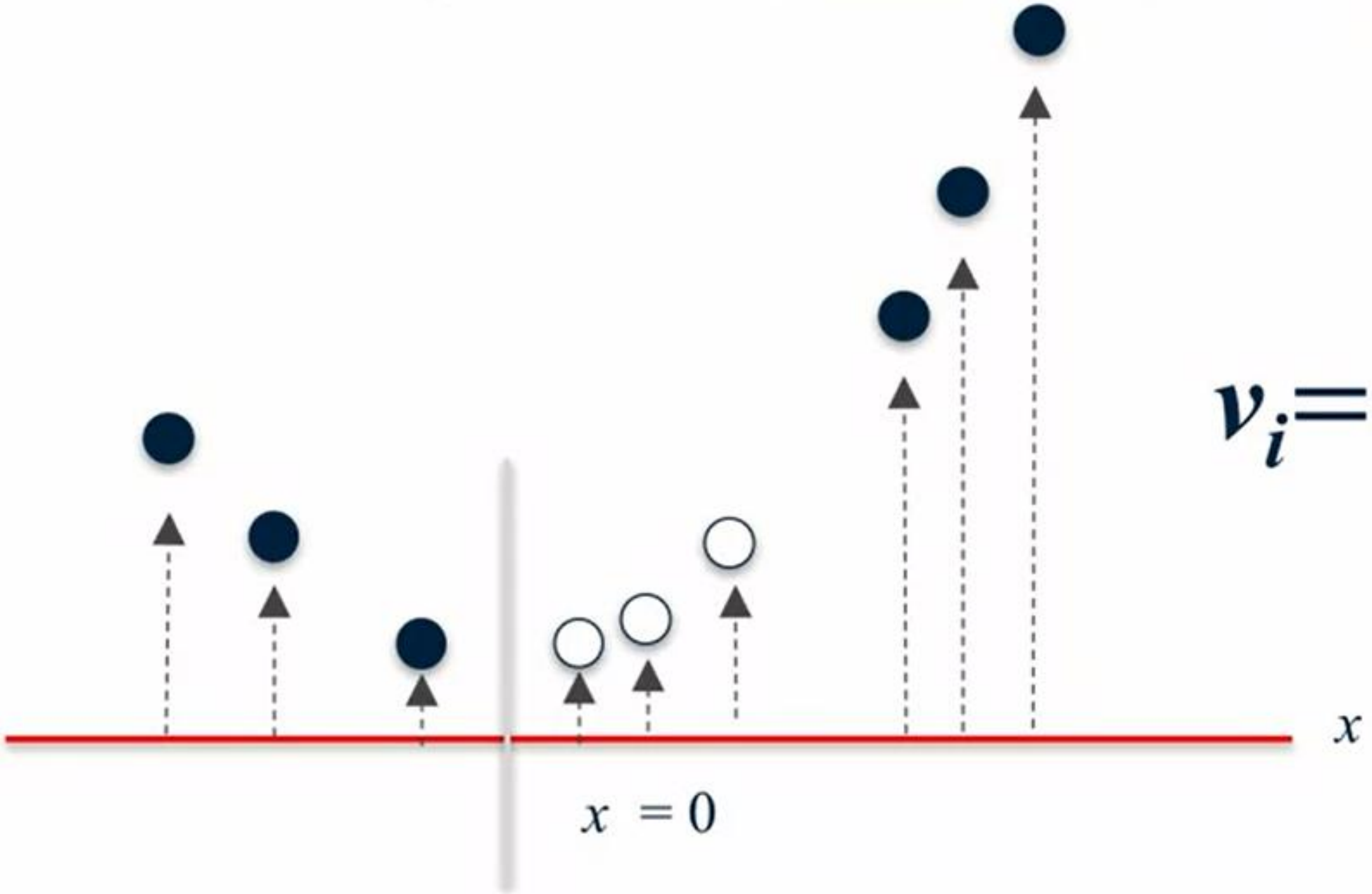
A simple 1-dimensional classification problem for a linear classifier



A more perplexing 1-d classification problem for a linear classifier

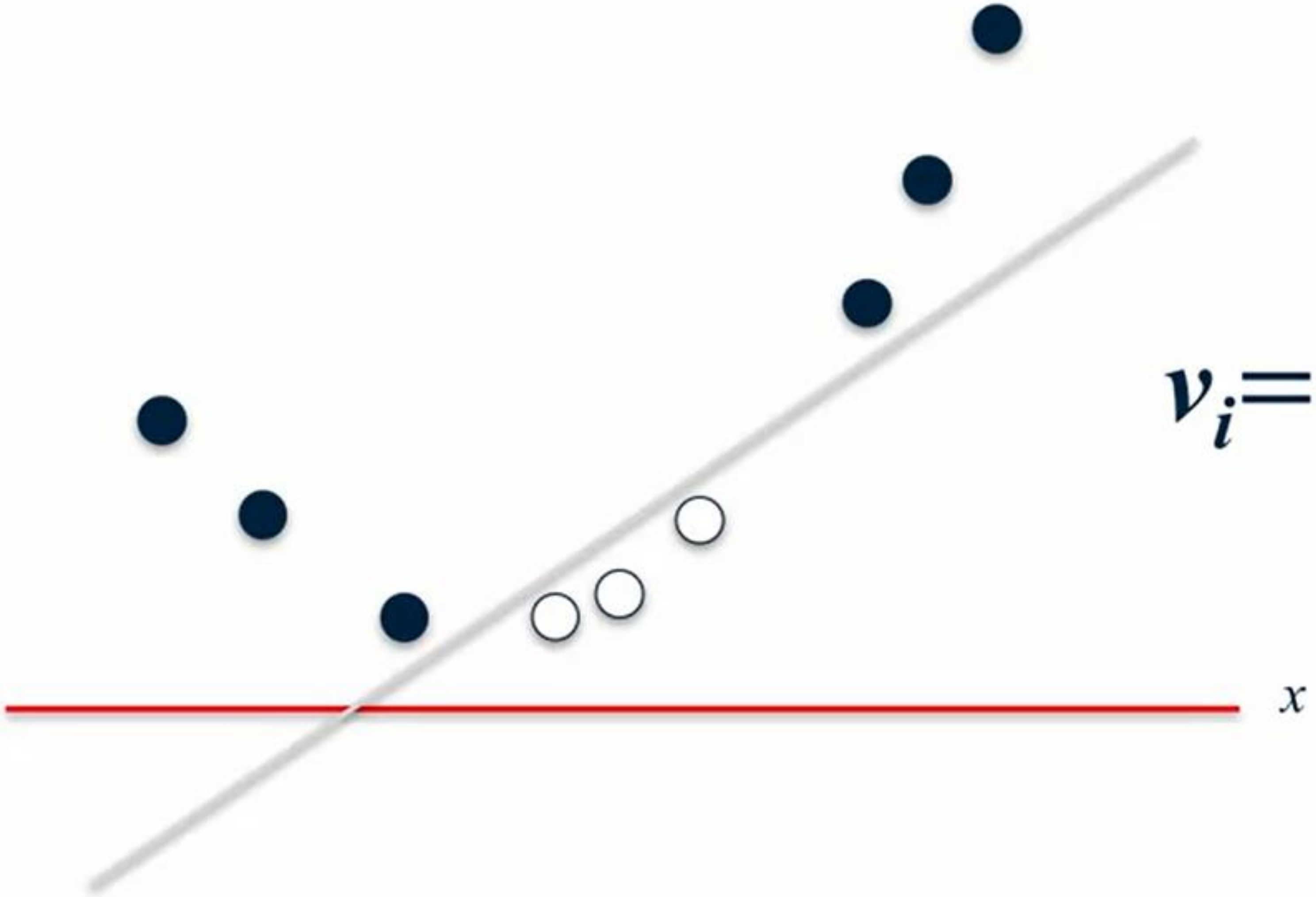


Let's transform the data by adding a second dimension/feature (set to the squared value of the first feature)



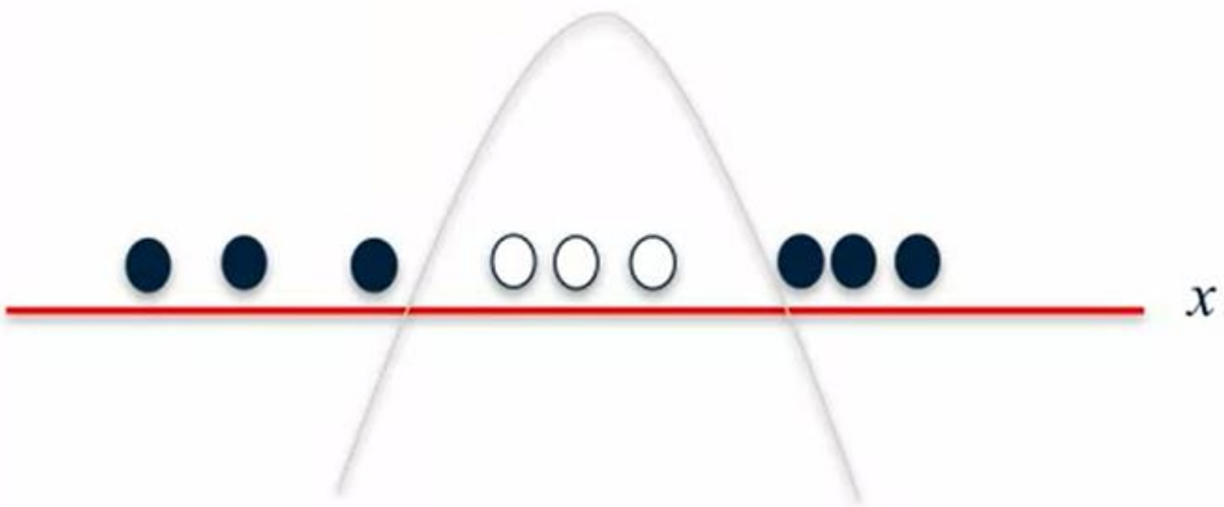
$$v_i = (x_i, x_i^2)$$

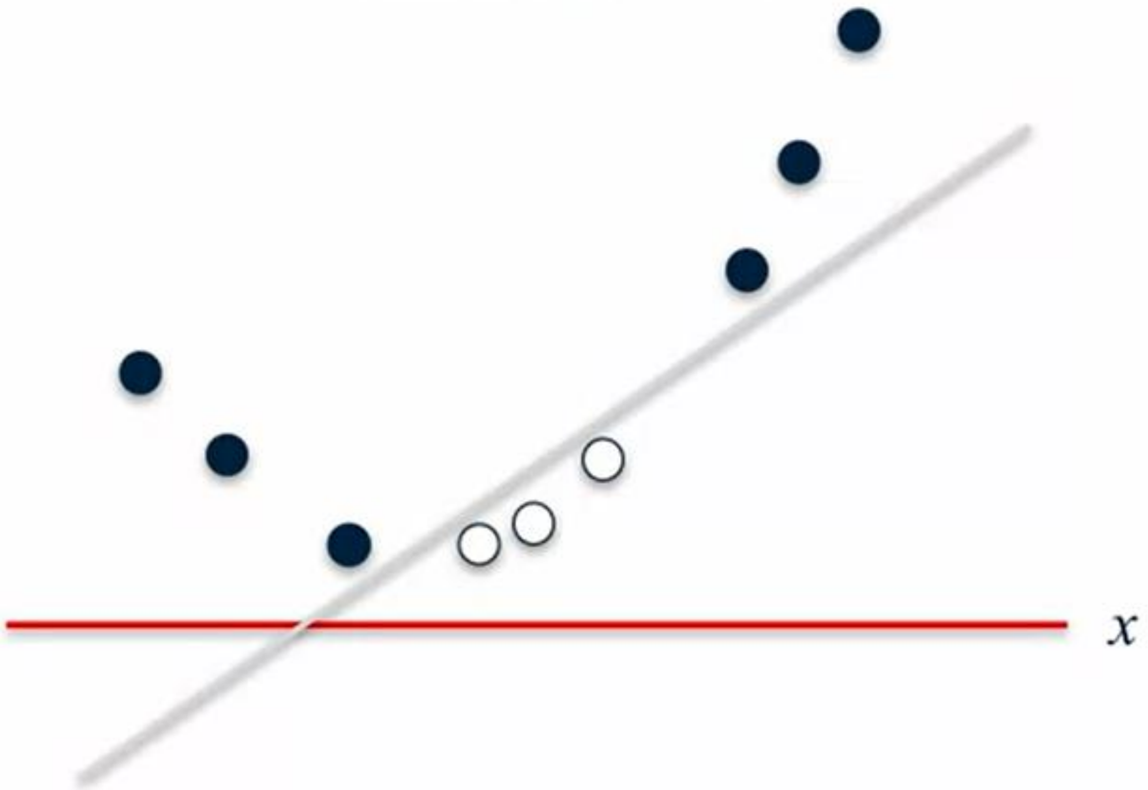
The data transformation makes it possible to solve this with a linear classifier



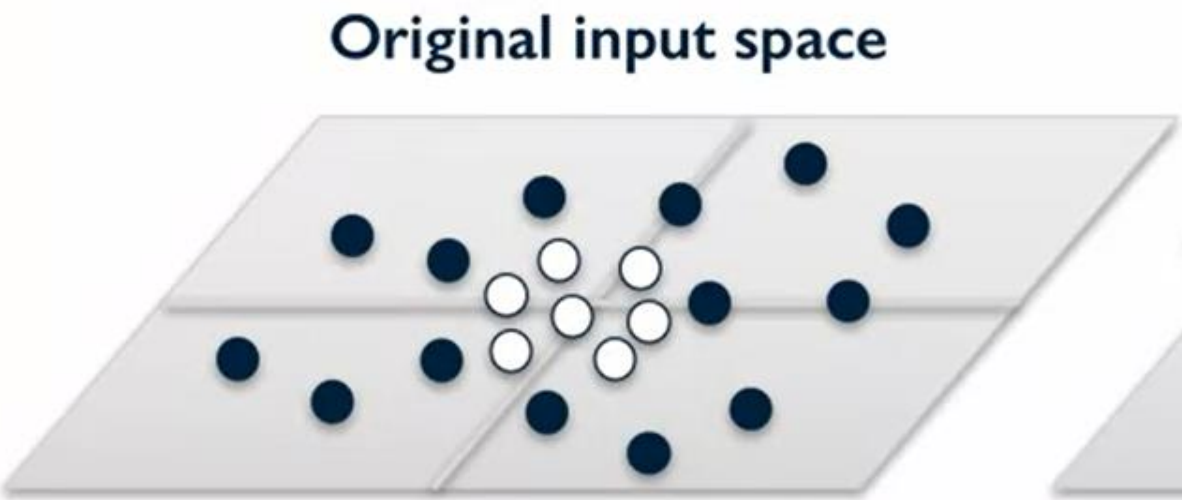
$$v_i = (x_i, x_i^2)$$

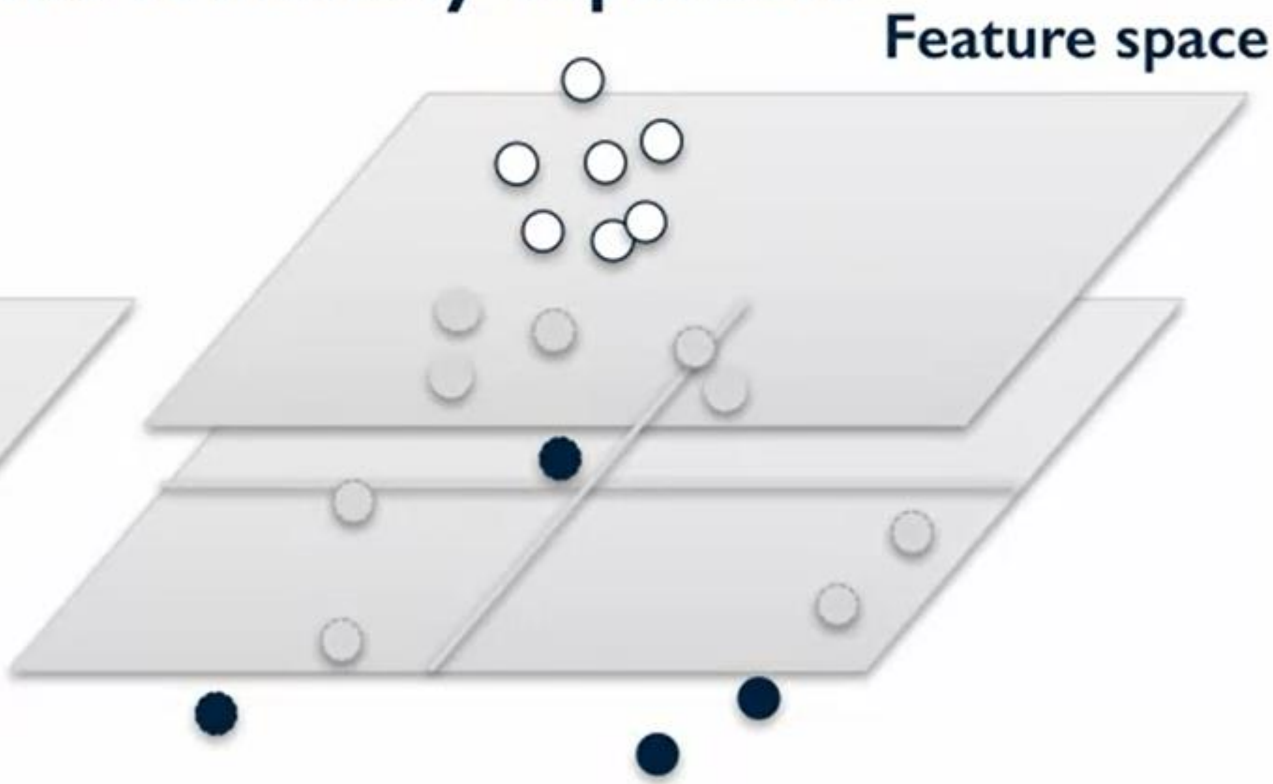
What does the linear decision boundary correspond to in the original input space?





Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

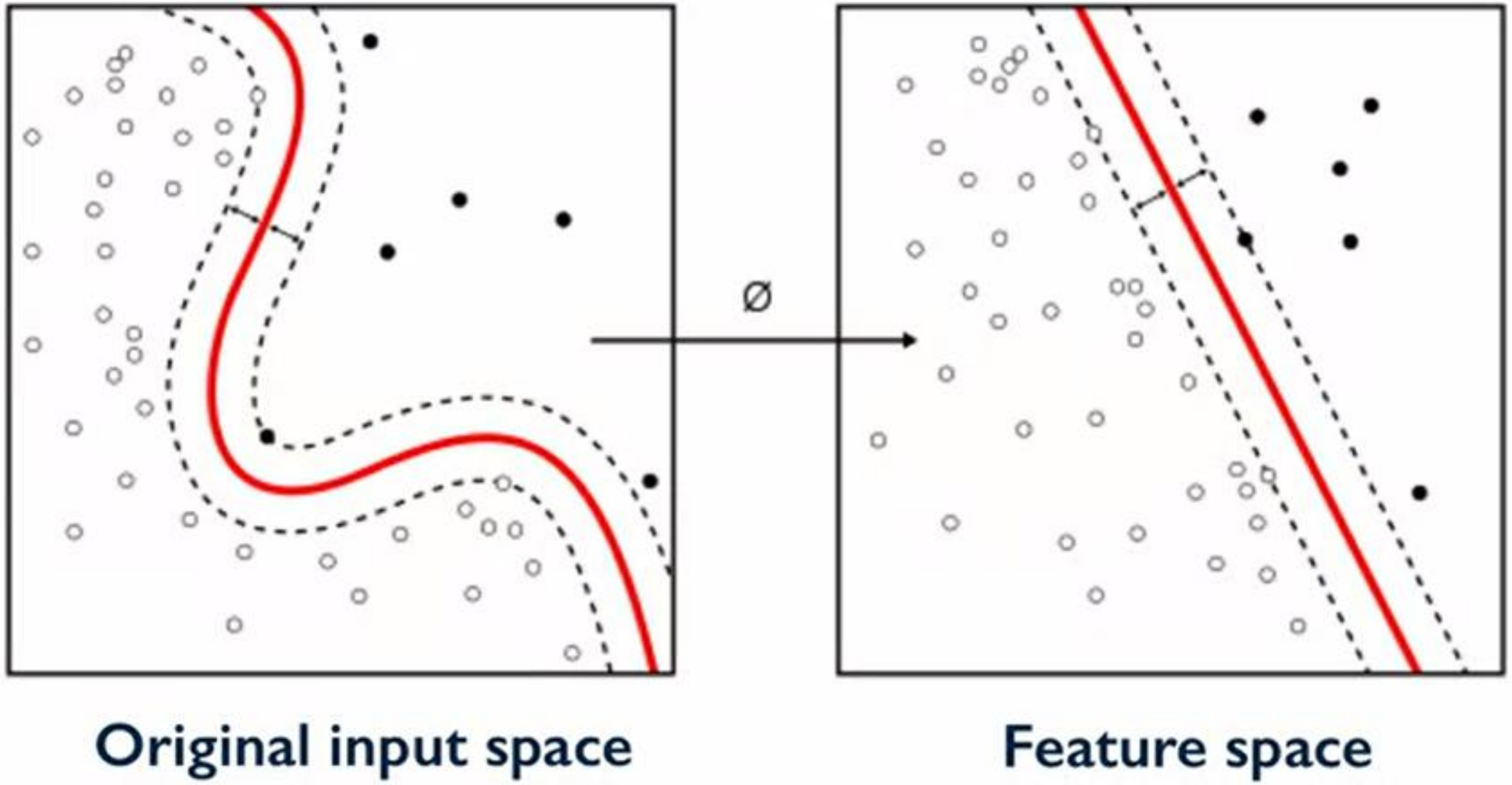




$$x_i = (x_0, x_1)$$

$$v_i = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

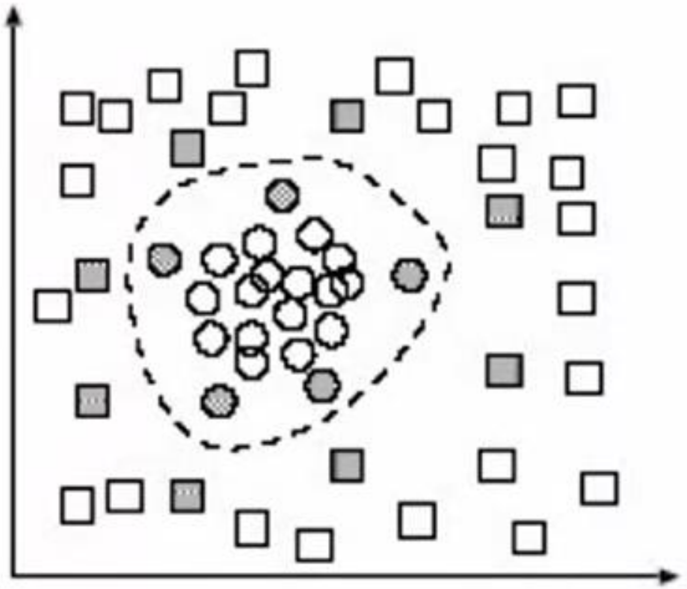
Transforming the data can make it much easier for a linear classifier.



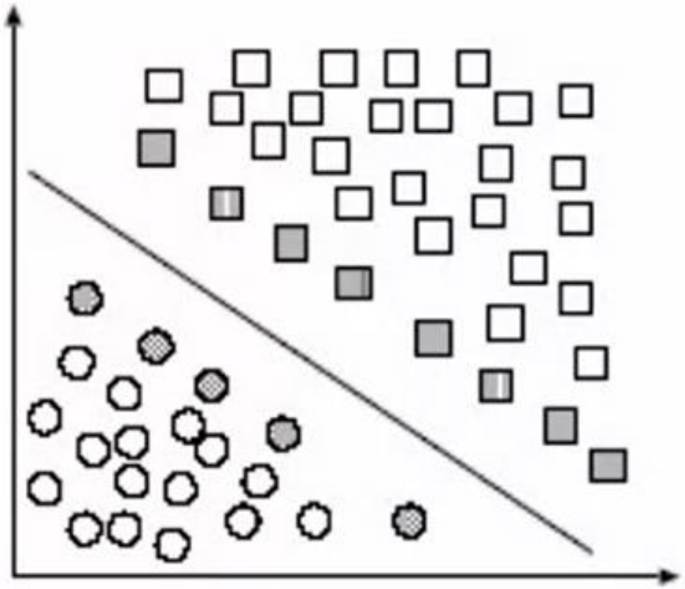
Source: Wikipedia "Kernel Machine" article.
https://commons.wikimedia.org/w/index.php?curid=47868867

Radial Basis Function Kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\gamma \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right]$$



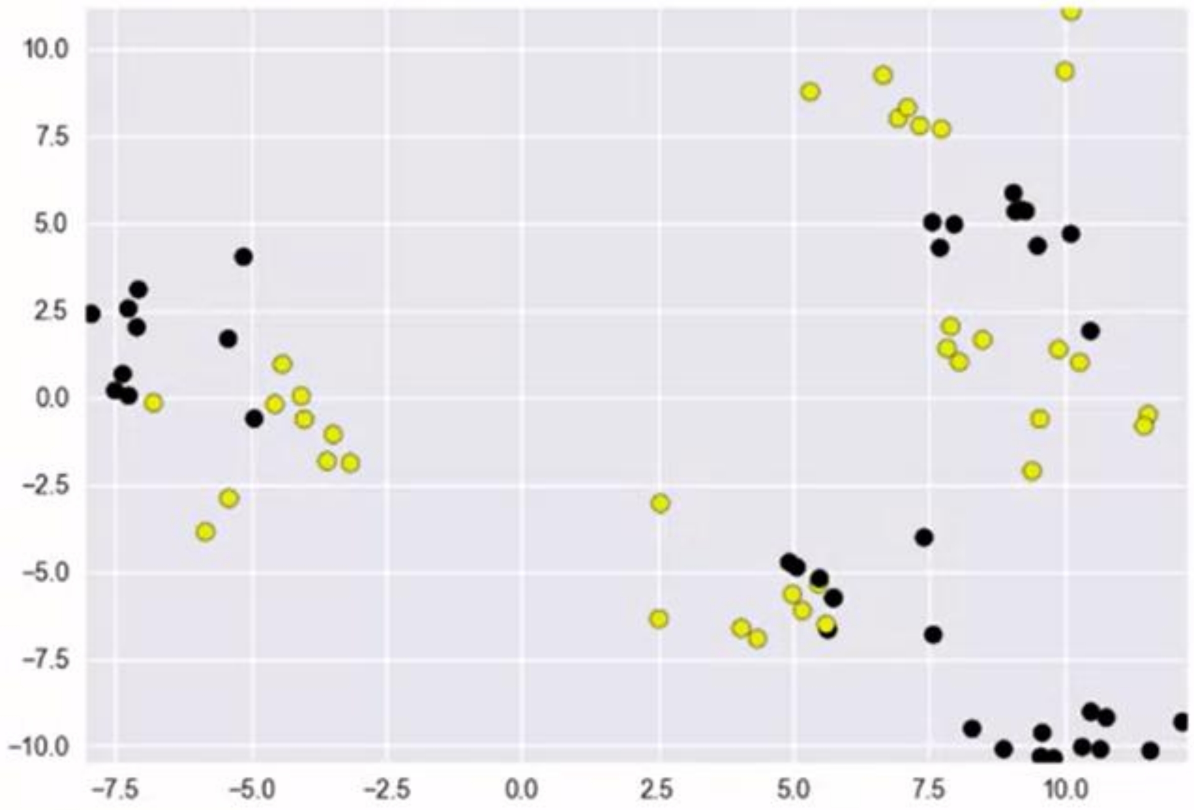
Original input space

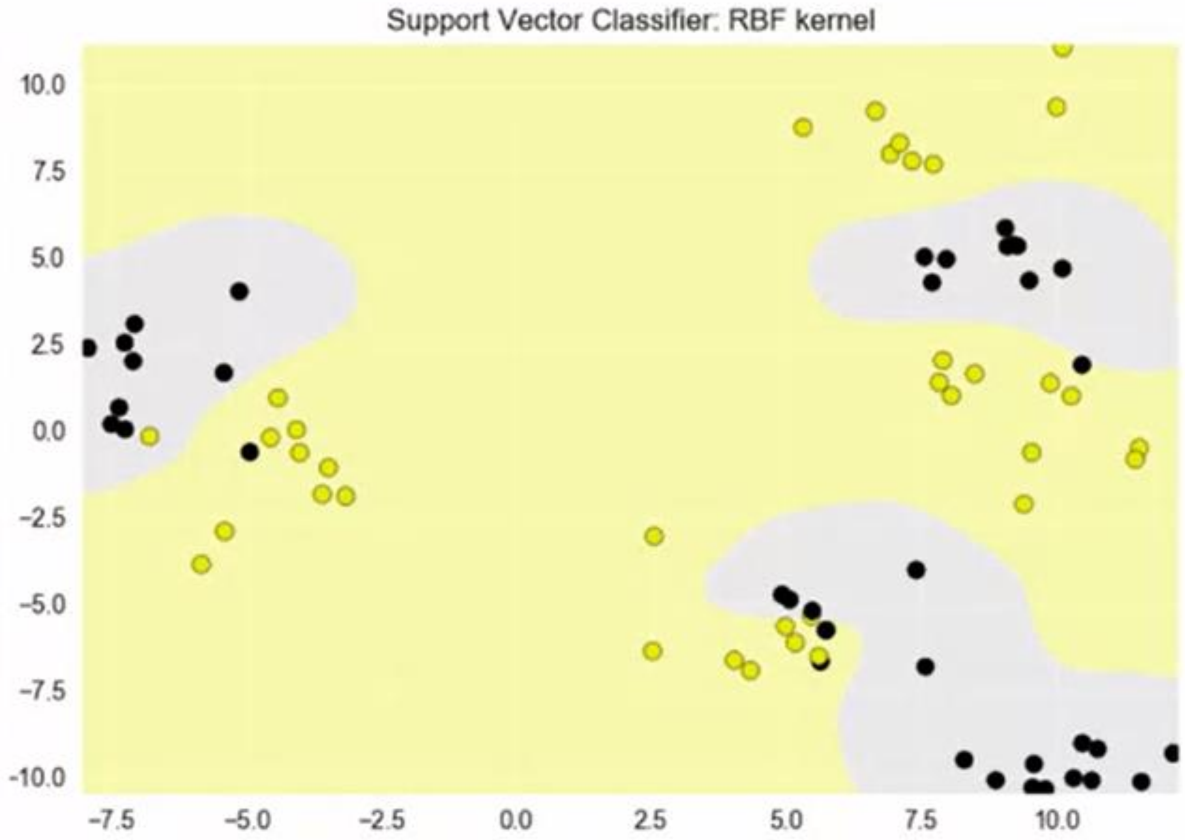


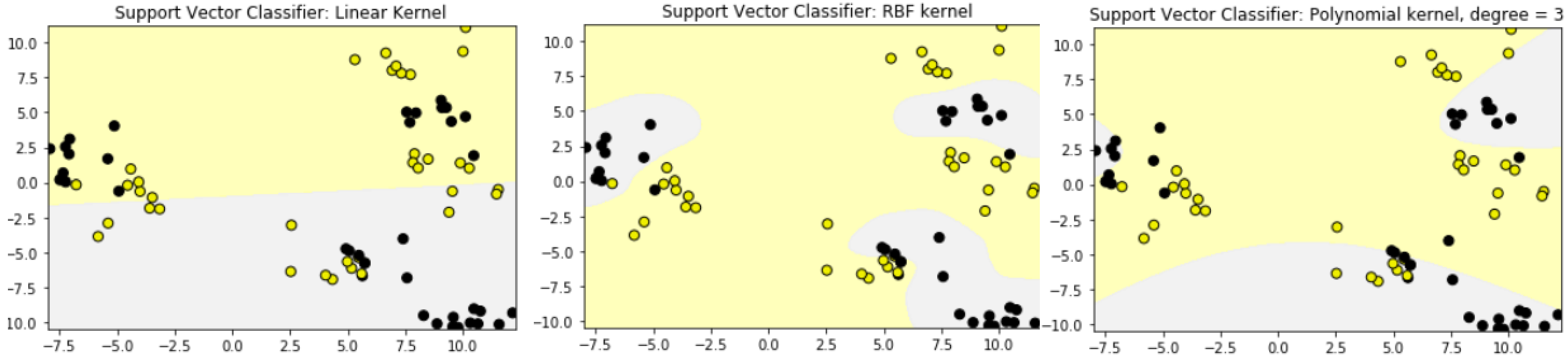
Feature space

A kernel is a similarity measure (modified dot product) between data points

Applying the SVM with RBF kernel






Support Vector Classifier: Linear Kernel
Support Vector Classifier: RBF kernel
Support Vector Classifier: Polynomial kernel, degree = 3
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Radial Basis Function kernel: Gamma Parameter

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\gamma \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right]$$

gamma (γ): kernel width parameter

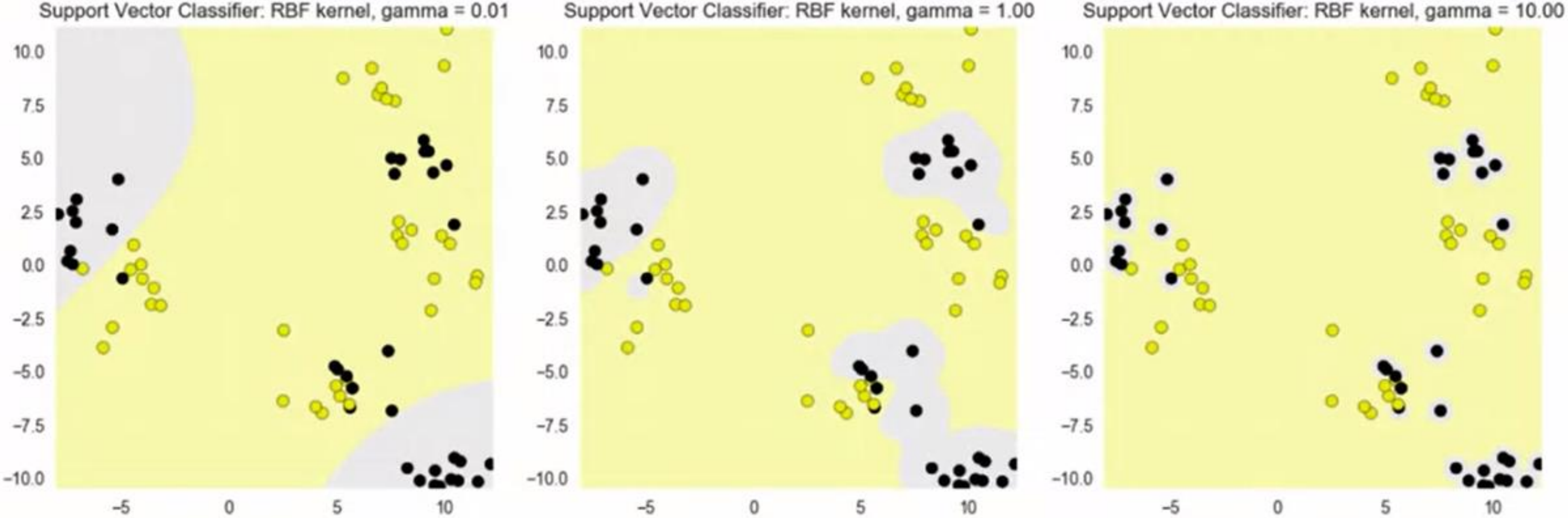
small gamma (0.01) large gamma (10)

1 1

$\|\boldsymbol{x} - \boldsymbol{x}'\|^2$ 0 0

Squared distance between x and x'

The effect of the RBF gamma parameter on decision boundaries

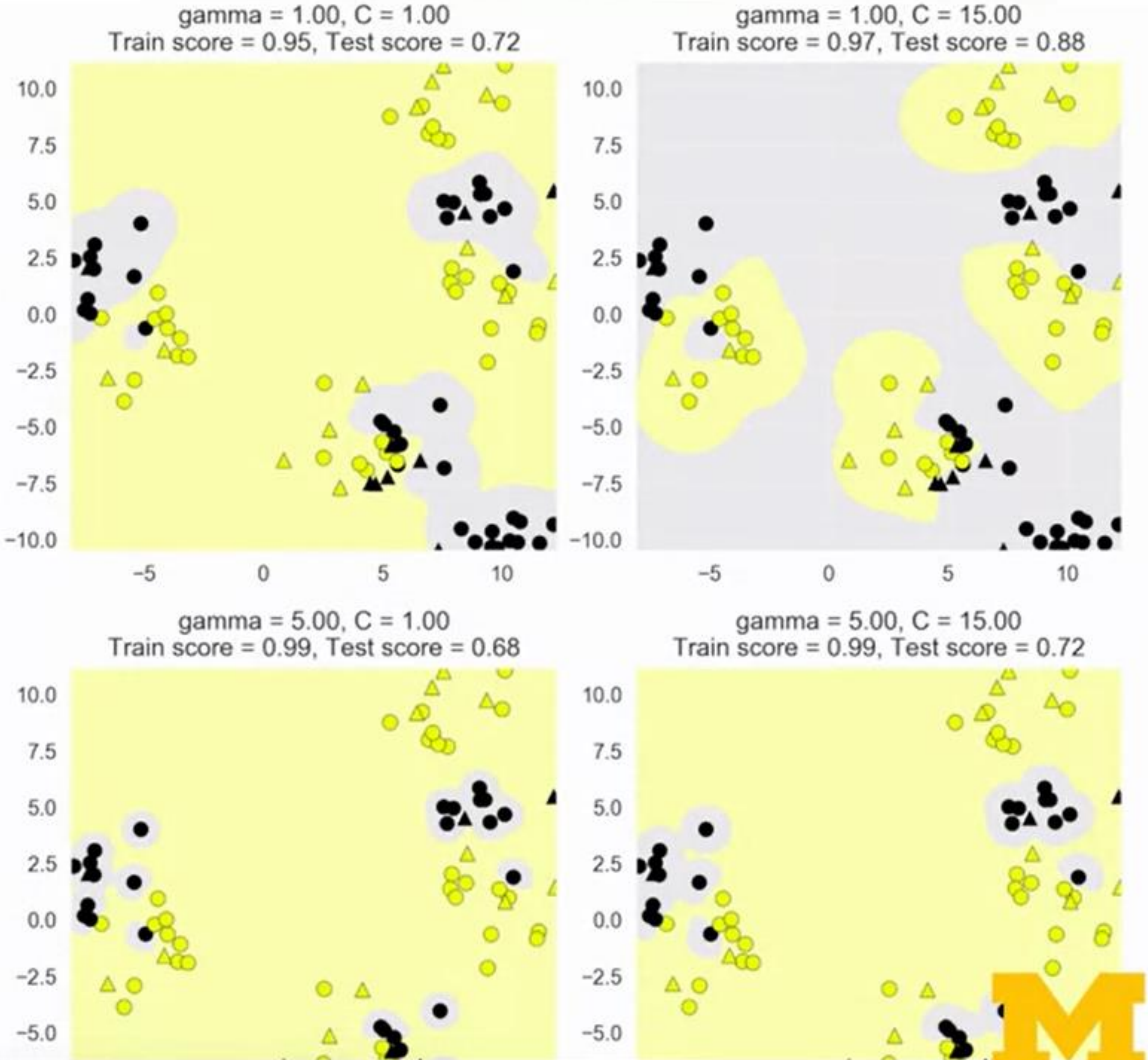


gamma = 0.01 gamma = 1.0 gamma = 10

Support Vector Machine with RBF kernel: using both C and gamma parameter

In [ ]:
```
from sklearn.svm import SVC
from adspy_shared_utilities import (
plot_class_regions_for_classifier_subplot)

from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2,
                                                    random_state = 0)
fig, subaxes = plt.subplots(3, 4, figsize=(15, 12))

for this_gamma, this_axis in zip([0.01, 1, 5], subaxes):

    for this_C, subplot in zip([0.1, 1, 15, 250], this_axis):
        title = 'gamma = {:.2f}, C = {:.2f}'.format(this_gamma, this_C)
        clf = SVC(kernel = 'rbf', gamma = this_gamma,
                 C = this_C).fit(X_train, y_train)
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                  X_test, y_test, title,
                                                  subplot)
        plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```

- Using both
- Gamma and C paramater



Kernelized Support Vector Machines: pros and cons

Pros:

- Can perform well on a range of datasets.
- Versatile: different kernel functions can be specified, or custom kernels can be defined for specific data types.
- Works well for both low- and high-dimensional data.

Cons:

- Efficiency (runtime speed and memory usage) decreases as training set size increases (e.g. over 50000 samples).
- Needs careful normalization of input data and parameter tuning.
- Does not provide direct probability estimates (but can be estimated using e.g. Platt scaling).
- Difficult to interpret why a prediction was made.

Kernelized Support Vector Machines (SVC): Important parameters

Model complexity

- `kernel`: Type of kernel function to be used
  - *Default = 'rbf' for radial basis function*
  - *Other types include 'polynomial'*
- kernel parameters
  - `gamma` *(γ): RBF kernel width*
- `C`: regularization parameter
- Typically `C` and `gamma` are tuned at the same time.