Applied Machine Learning

Kernelized Support Vector Machines

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

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

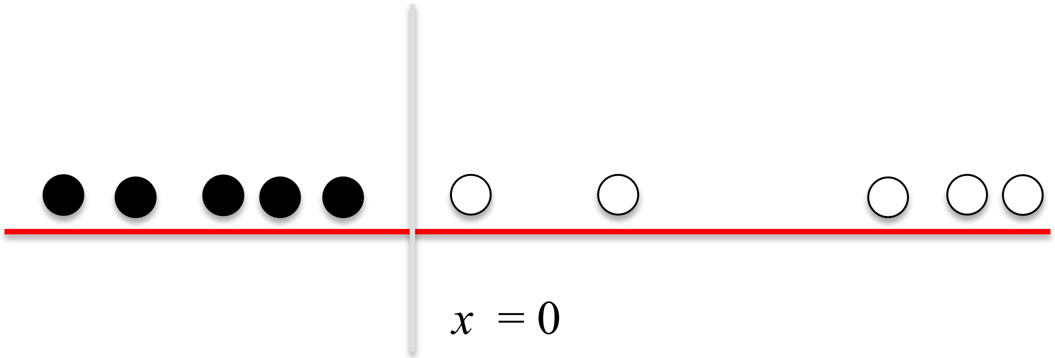
Easy for a linear classifier

But what about more complex binary classification problems?

Easy for a linear classifier

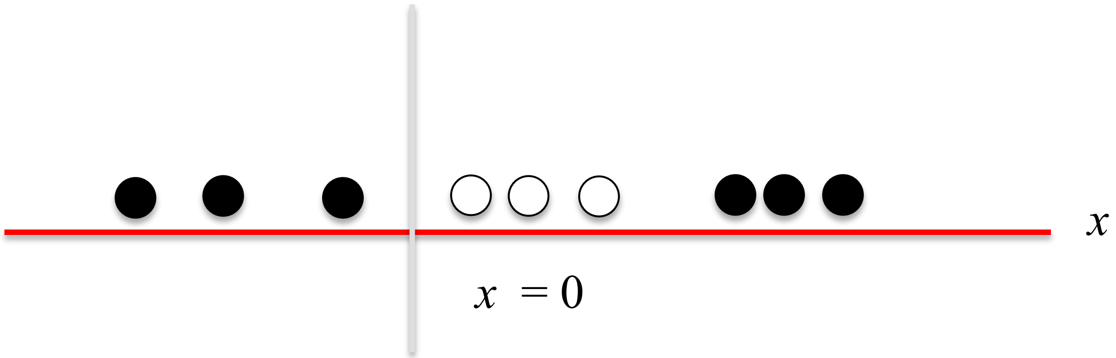
Difficult/impossible for a linear classifier

A simple 1-dimensional classification problem for a linear classifier

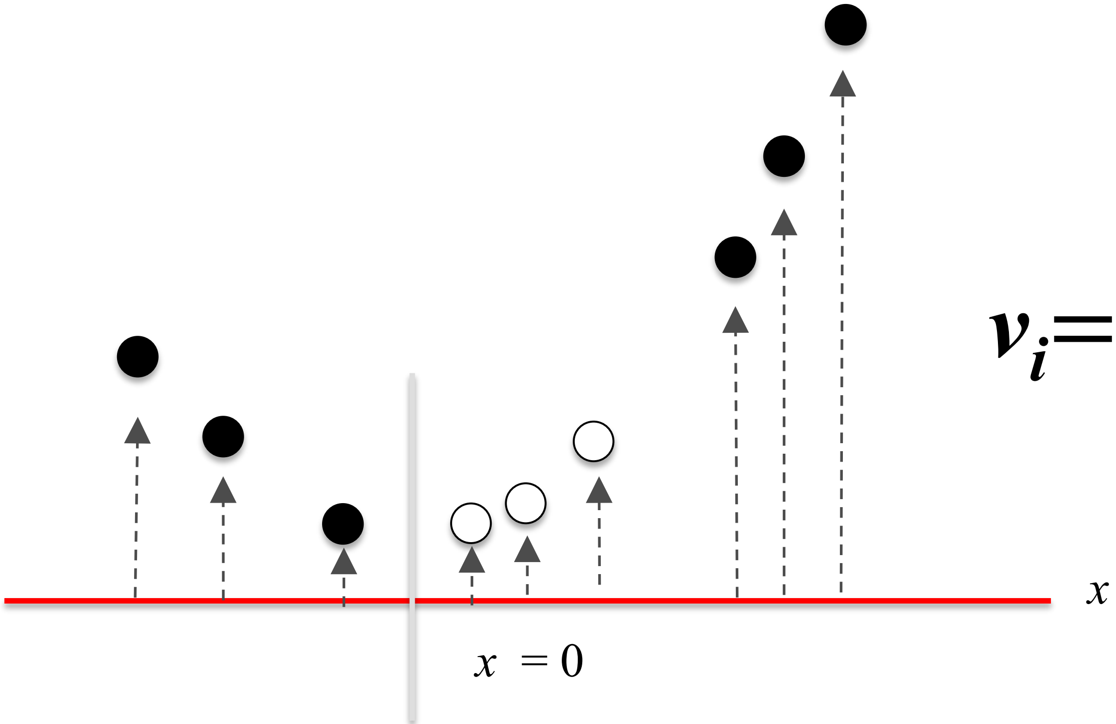


A more perplexing 1-d classification problem for a linear classifier

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

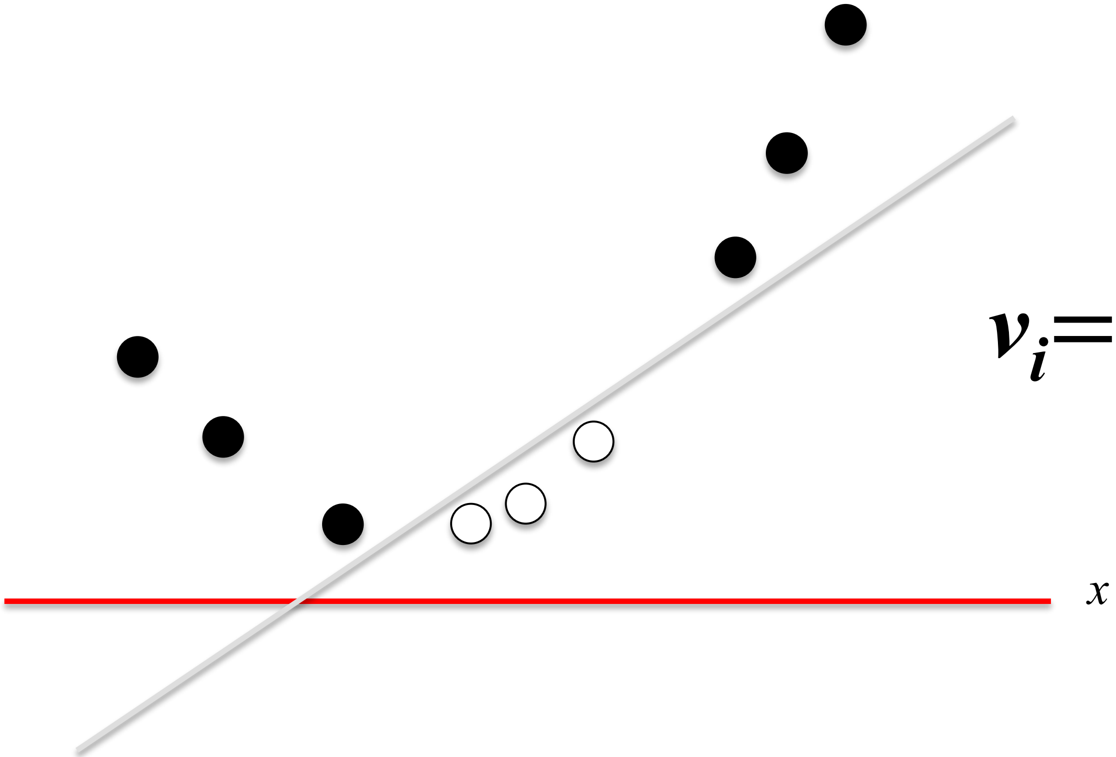


Let's transform the data by adding a second dimension/feature (set to the squared value of the first feature)



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

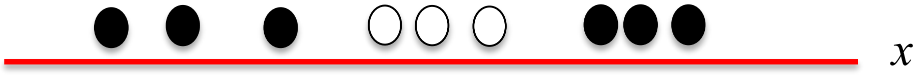
The data transformation makes it possible to solve this with a linear classifier



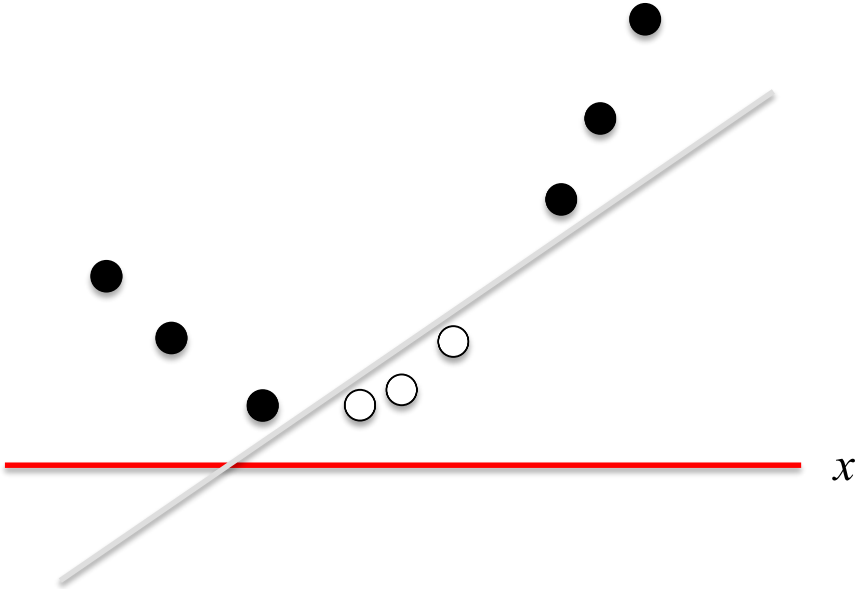
$$v_i = (x_i, x_i^2)$$

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

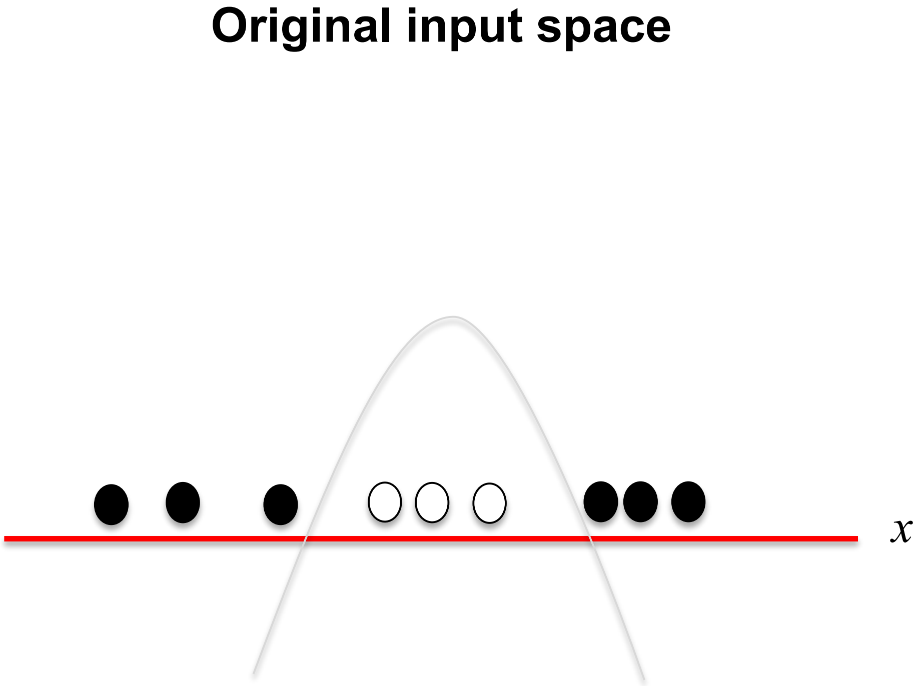
Original input space

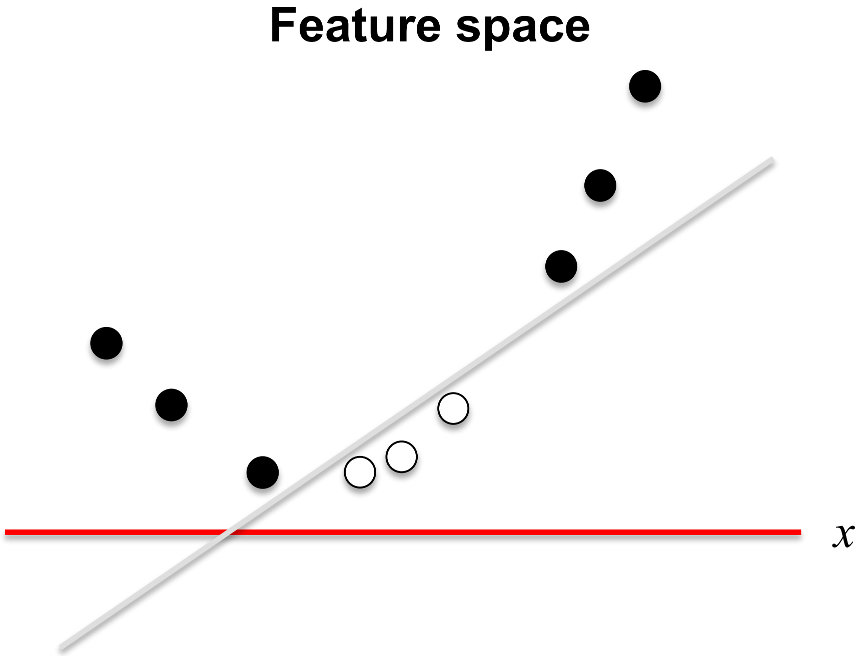


Feature space

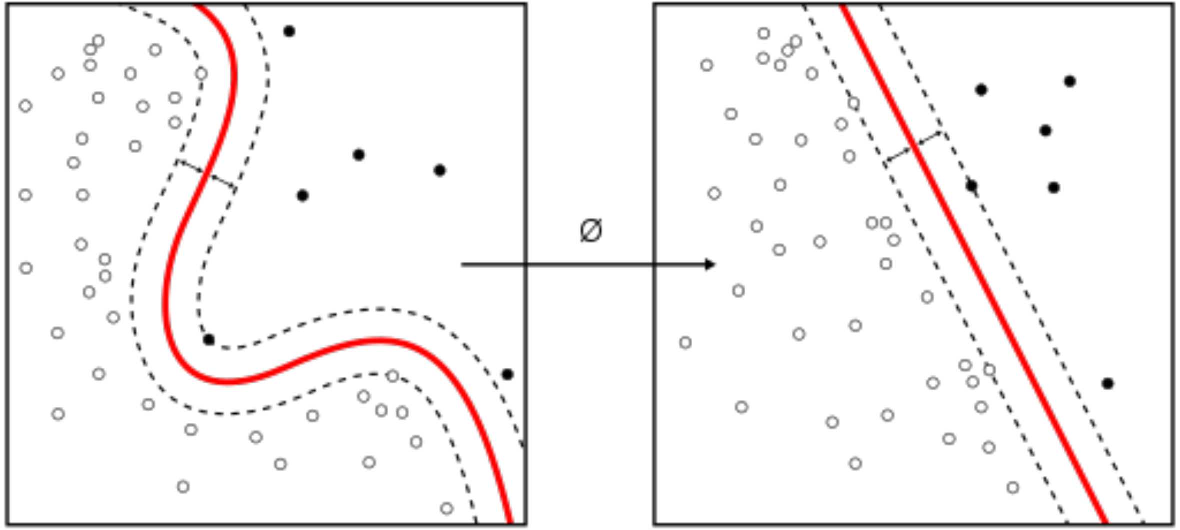


What does the linear decision boundary correspond to in the original input space?





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



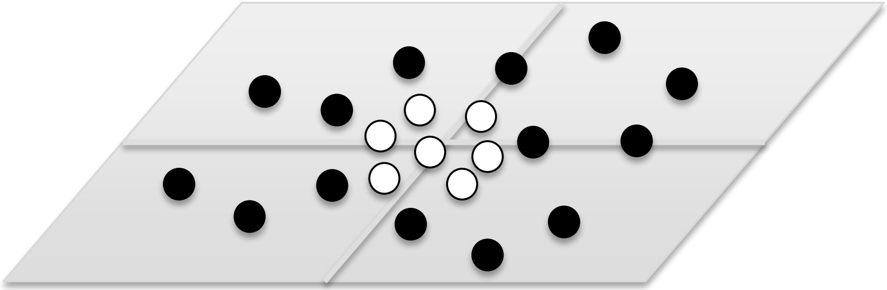
Original input space

Feature space

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

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

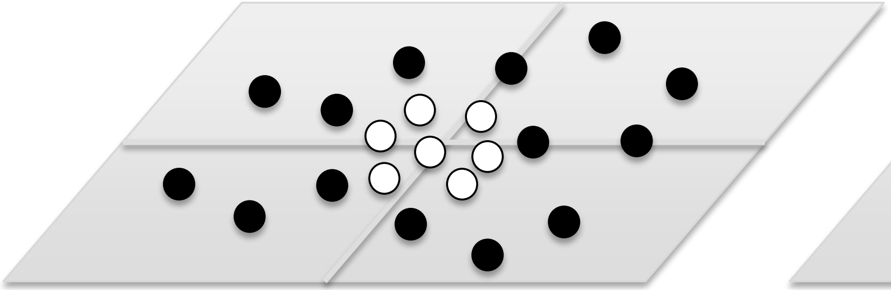
Original input space



$$x_i = (x_0, x_1)$$

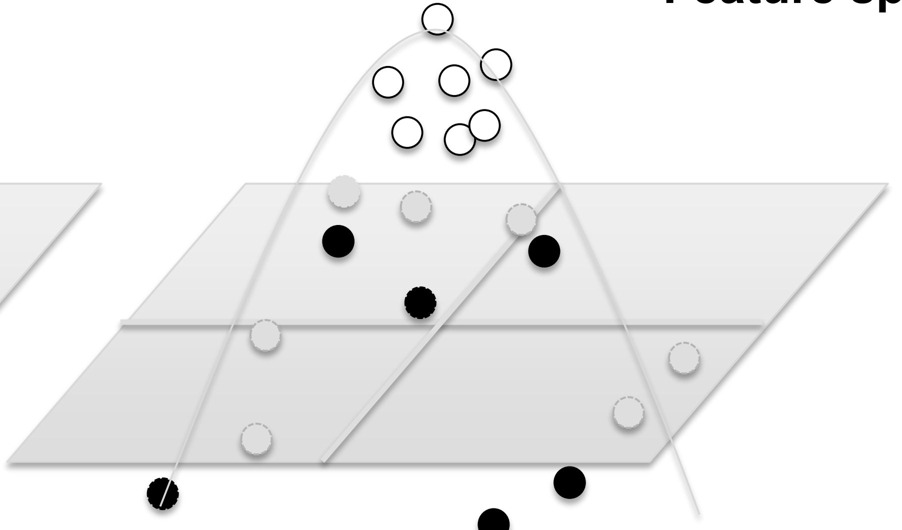
Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

Original input space



$$x_i = (x_0, x_1)$$

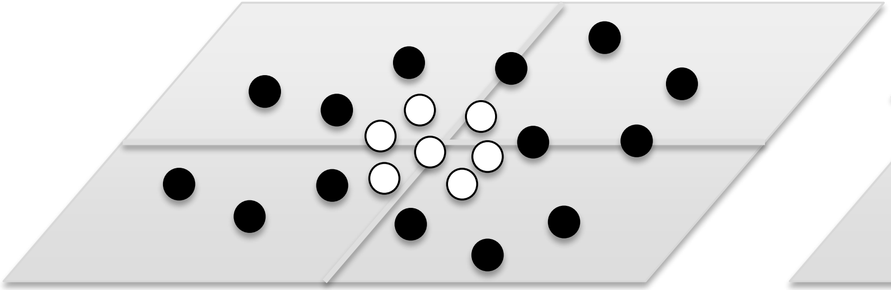
Feature space



$$\boldsymbol{v_i} = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

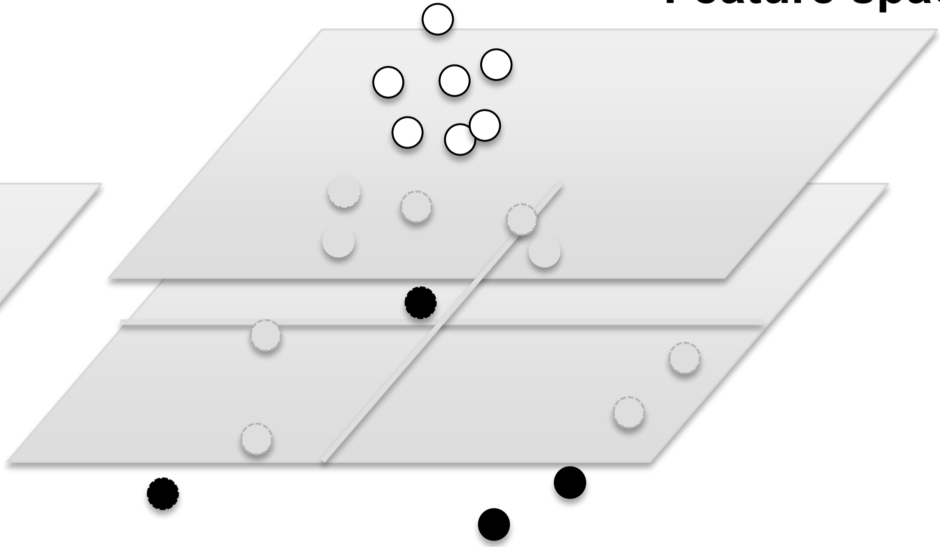
Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

Original input space



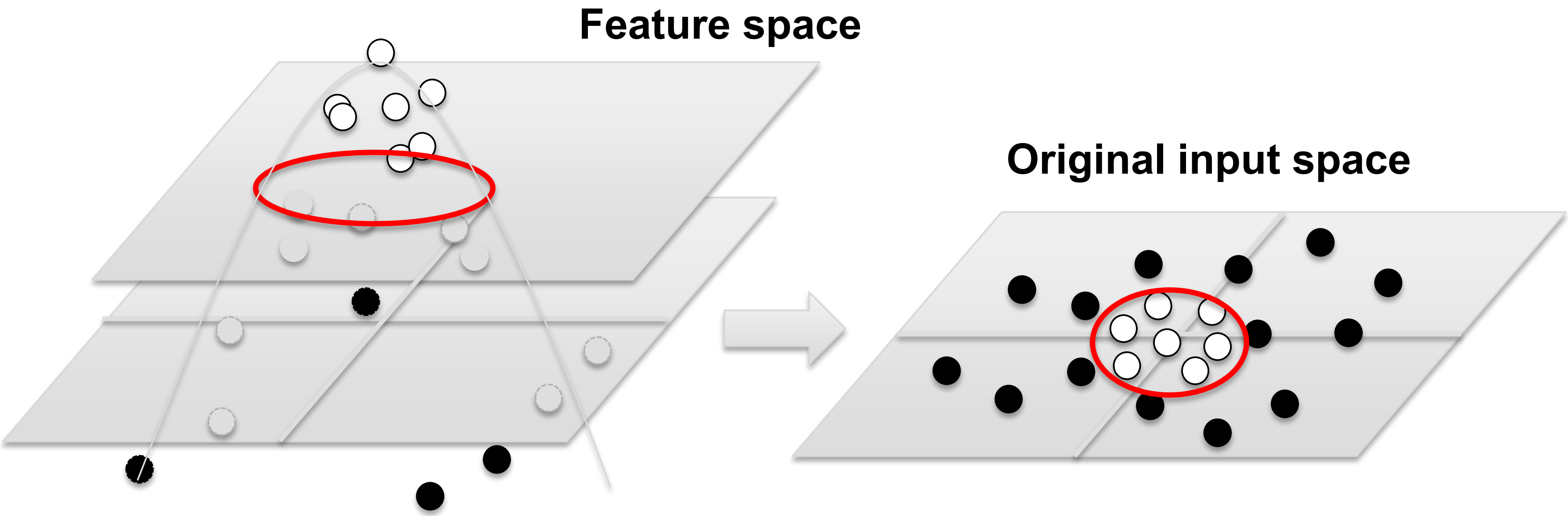
$$x_i = (x_0, x_1)$$

Feature space



$$\boldsymbol{v_i} = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

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

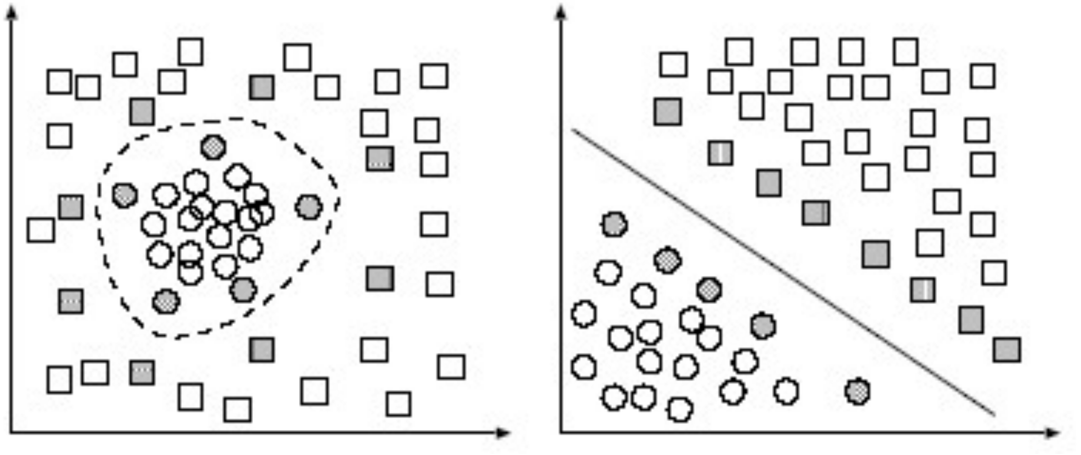


$$\boldsymbol{v_i} = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

$$\boldsymbol{x_i} = (x_0, x_1)$$

Radial Basis Function Kernel

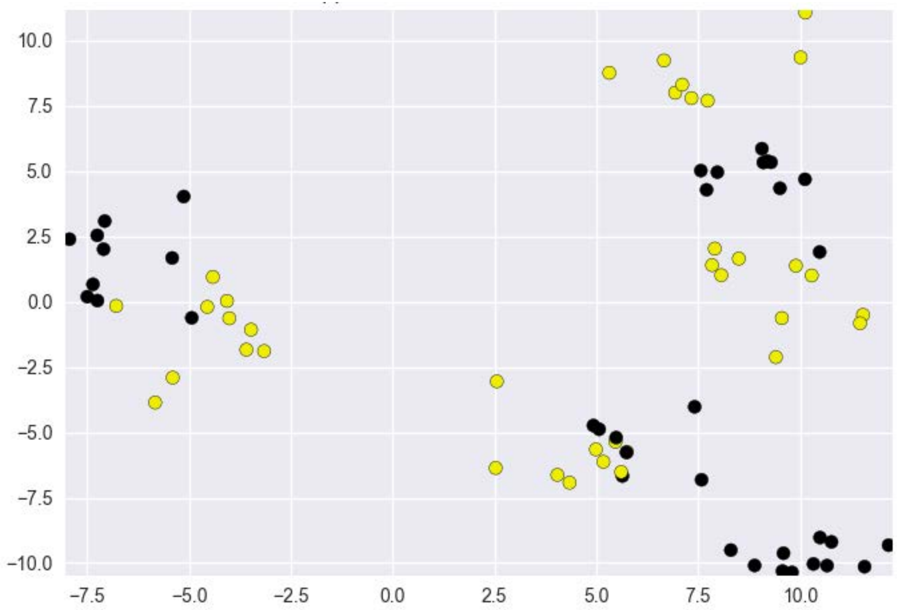
$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\ [-\gamma \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2]$$

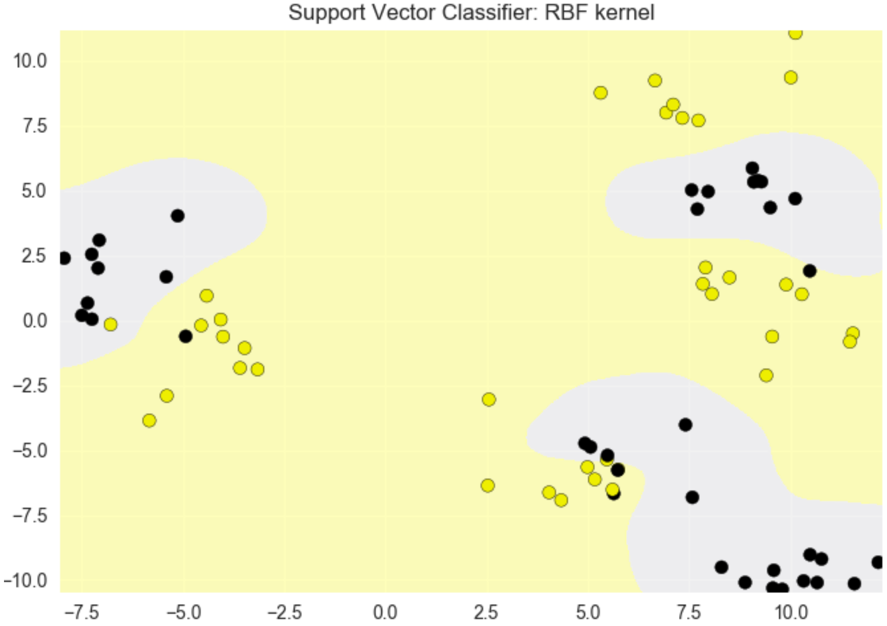


Original input space Feature space

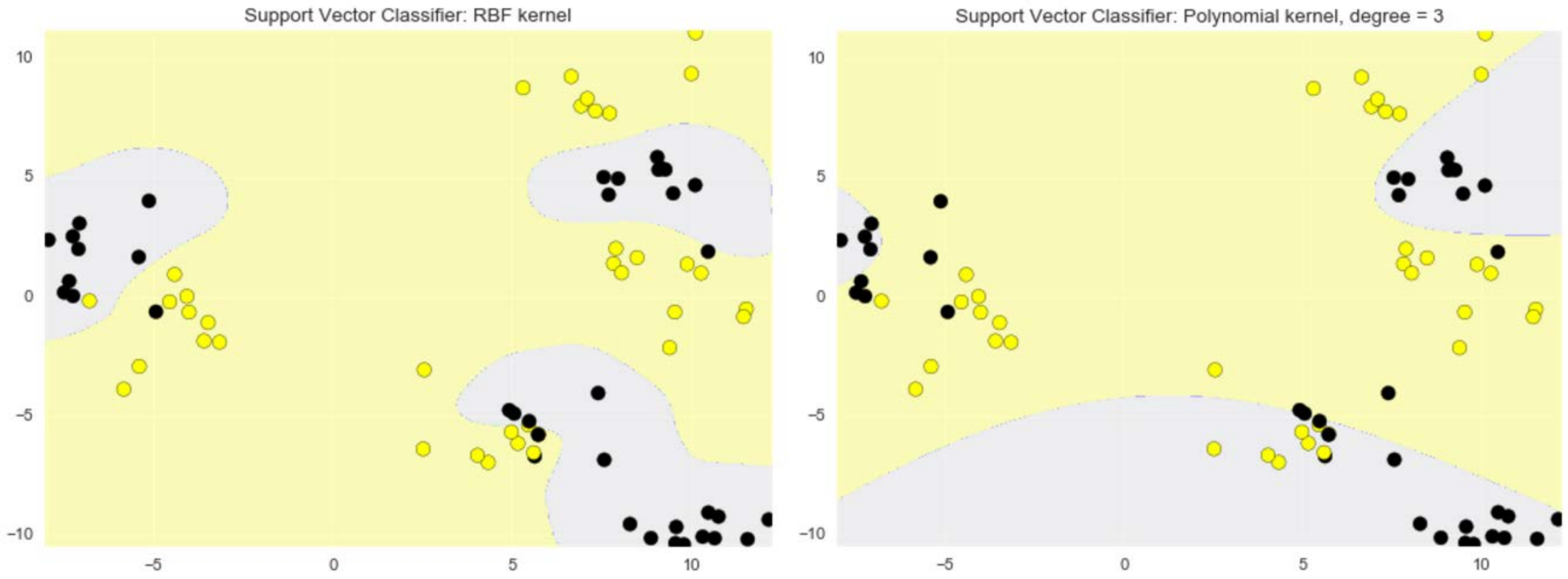
A kernel is a similarity measure (modified dot product) between data points

Applying the SVM with RBF kernel





Radial Basis Kernel vs Polynomial Kernel



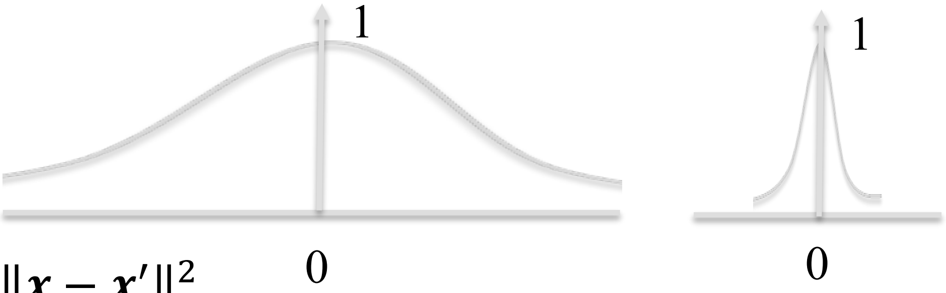
Radial Basis Function kernel: Gamma Parameter

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

gamma (γ): kernel width parameter

small gamma (0.01)

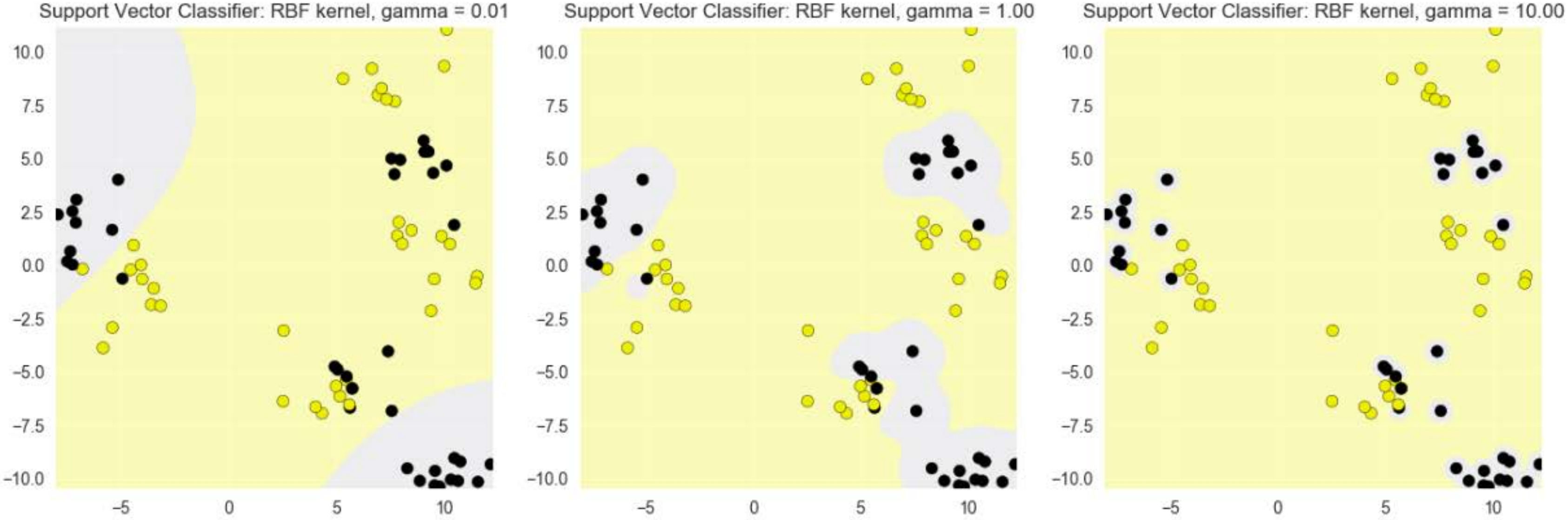
large gamma (10)



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

Squared distance between x and x'

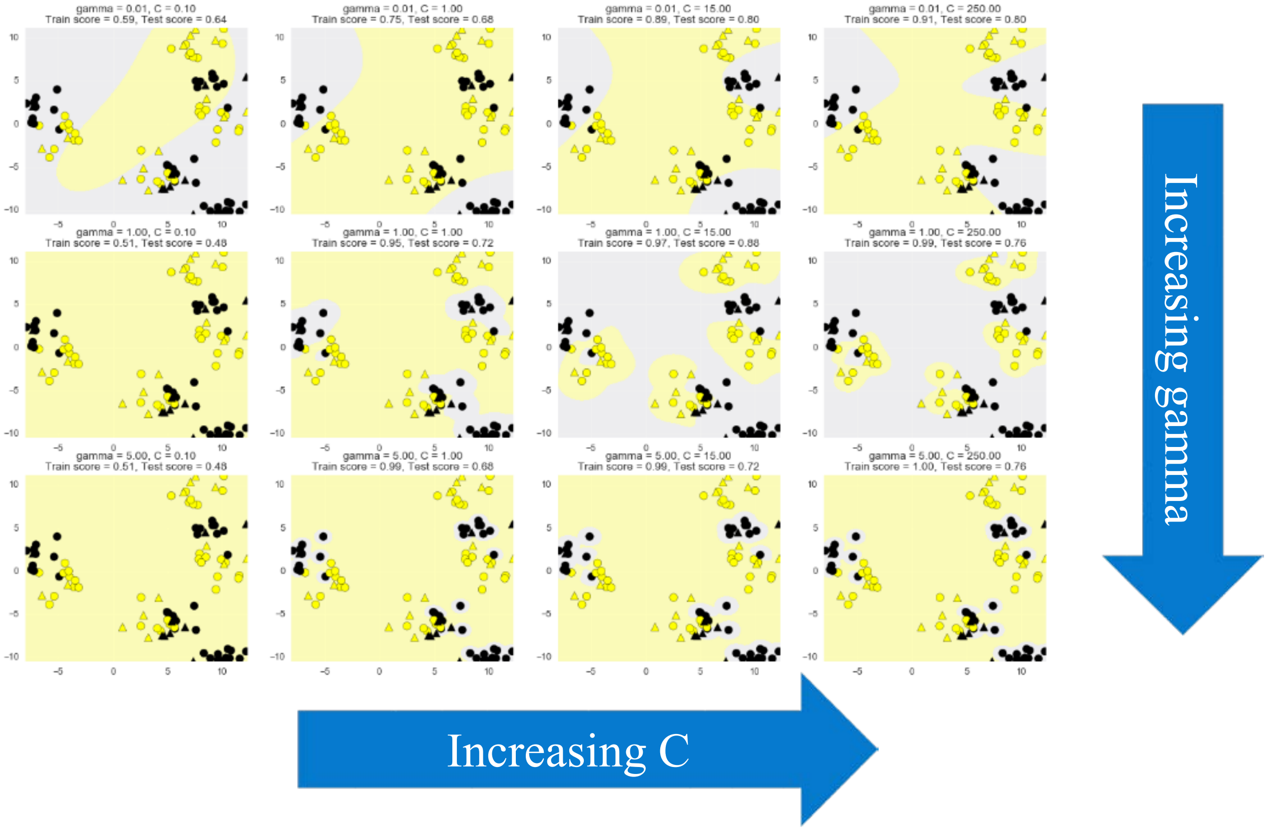
The effect of the RBF gamma parameter on decision boundaries



gamma = 0.01

gamma = 1.0

gamma = 10


gamma = 0.01, C = 0.10
Train score = 0.59, Test score = 0.64
gamma = 0.01, C = 1.00
Train score = 0.75, Test score = 0.68
gamma = 0.01, C = 15.00
Train score = 0.89, Test score = 0.80
gamma = 0.01, C = 250.00
Train score = 0.91, Test score = 0.80
gamma = 1.00, C = 0.10
Train score = 0.51, Test score = 0.48
gamma = 1.00, C = 1.00
Train score = 0.95, Test score = 0.72
gamma = 1.00, C = 15.00
Train score = 0.97, Test score = 0.88
gamma = 1.00, C = 250.00
Train score = 0.99, Test score = 0.76
gamma = 5.00, C = 0.10
Train score = 0.51, Test score = 0.48
gamma = 5.00, C = 1.00
Train score = 0.99, Test score = 0.68
gamma = 5.00, C = 15.00
Train score = 0.99, Test score = 0.72
gamma = 5.00, C = 250.00
Train score = 1.00, Test score = 0.76
Increasing gamma
Increasing C

Reminder: Using a scaler object: fit and transform methods

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
clf = SVC().fit(X_train_scaled, y_train)
accuracy = clf.score(X_test_scaled, y_test)
```

Tip: It can be more efficient to do fitting and transforming together on the training set using the `fit_transform` method.

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

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.**