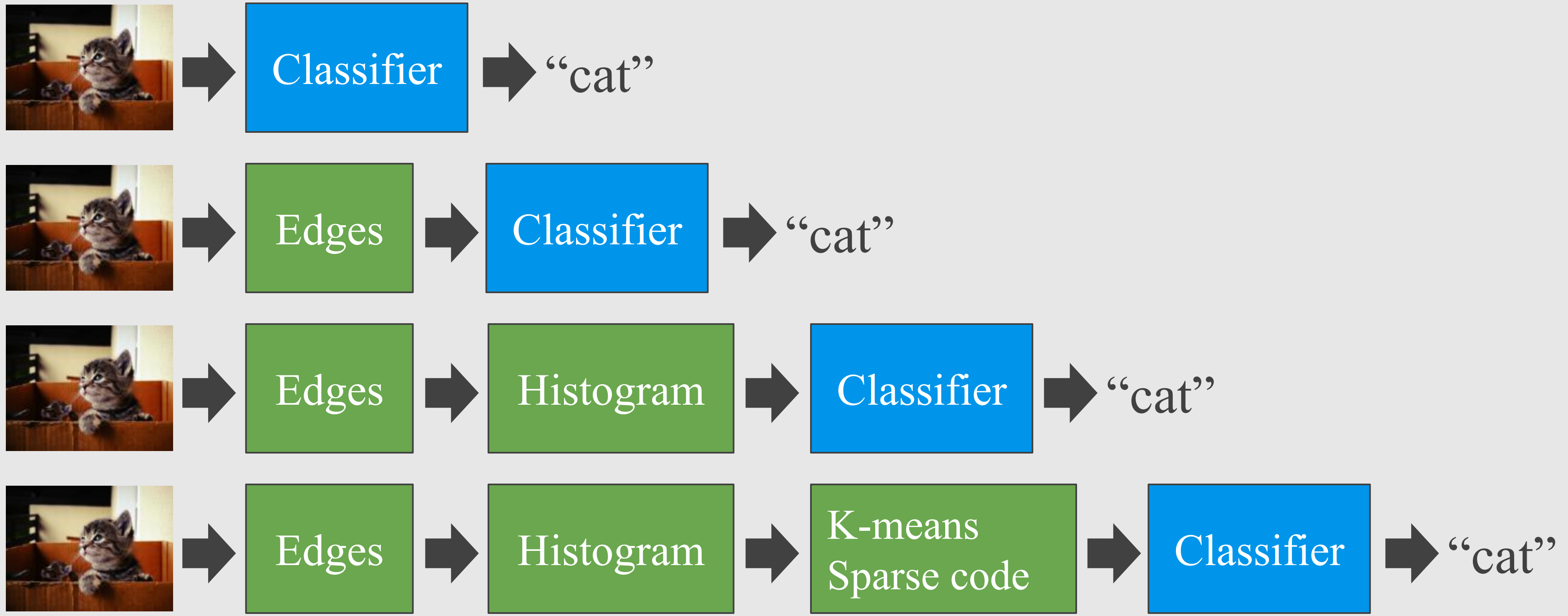
Machine Learning and Pattern Recognition
A High Level Overview

Prof. Anderson Rocha
(Main bulk of slides kindly provided by **Prof. Sandra Avila**)
Institute of Computing (IC/Unicamp)

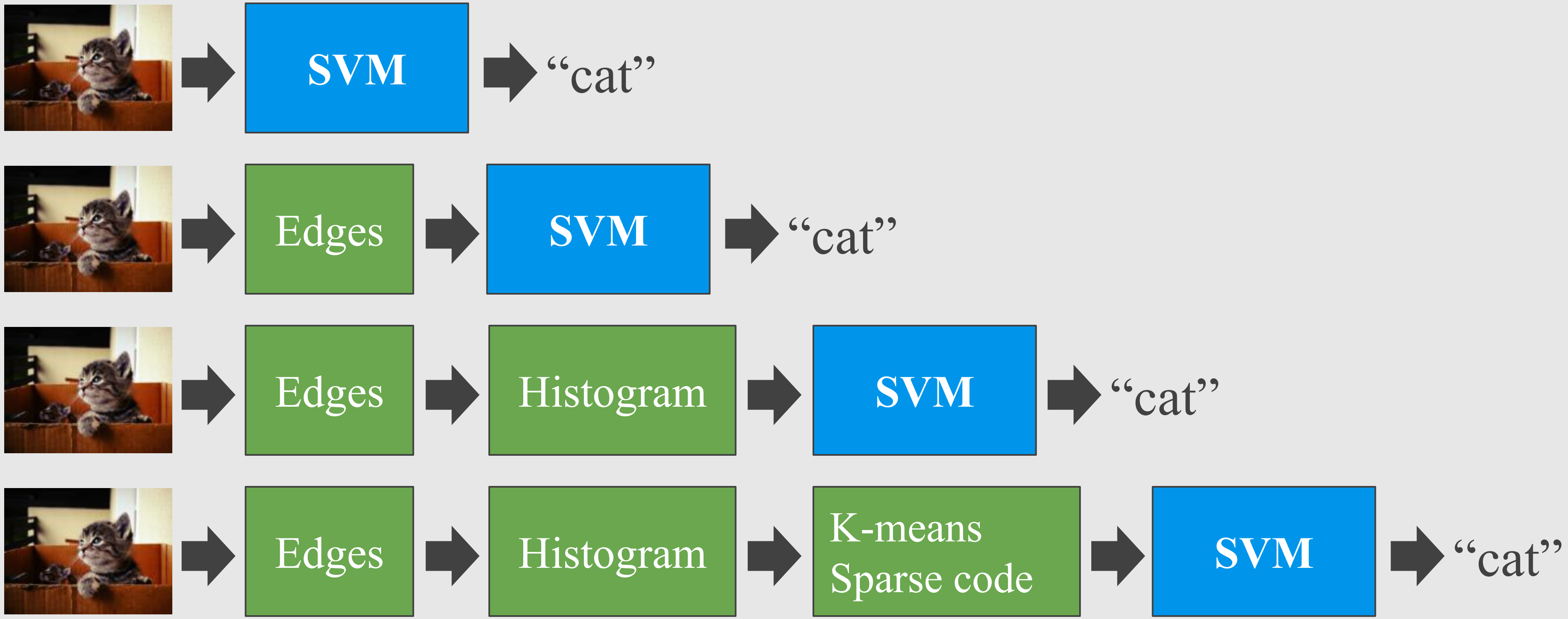
MC886/MO444

SVMs are among the best “off-the-shelf” supervised learning algorithm.

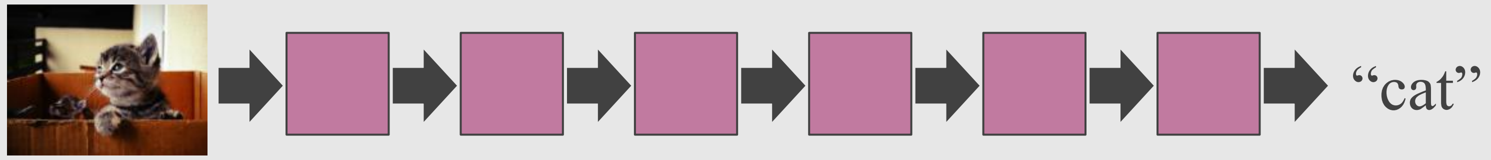
Traditional Recognition



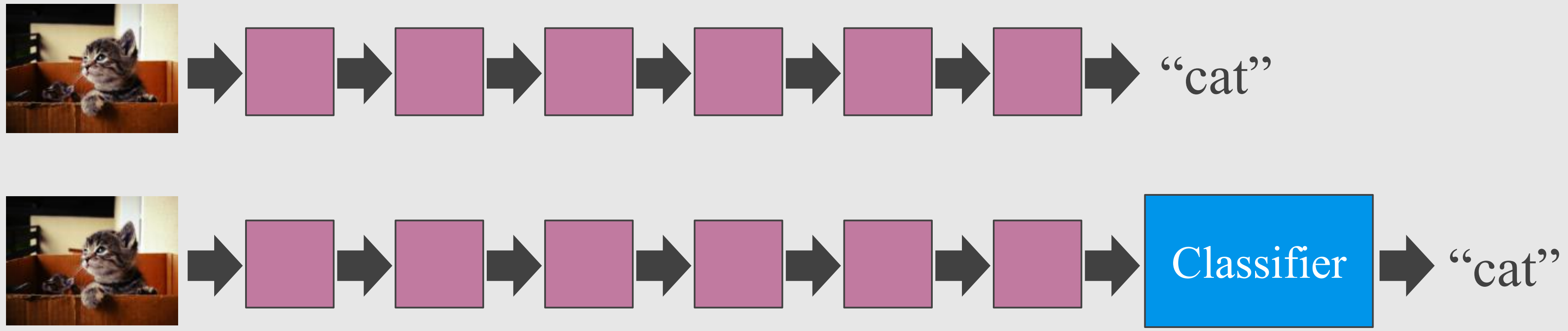
Traditional Recognition



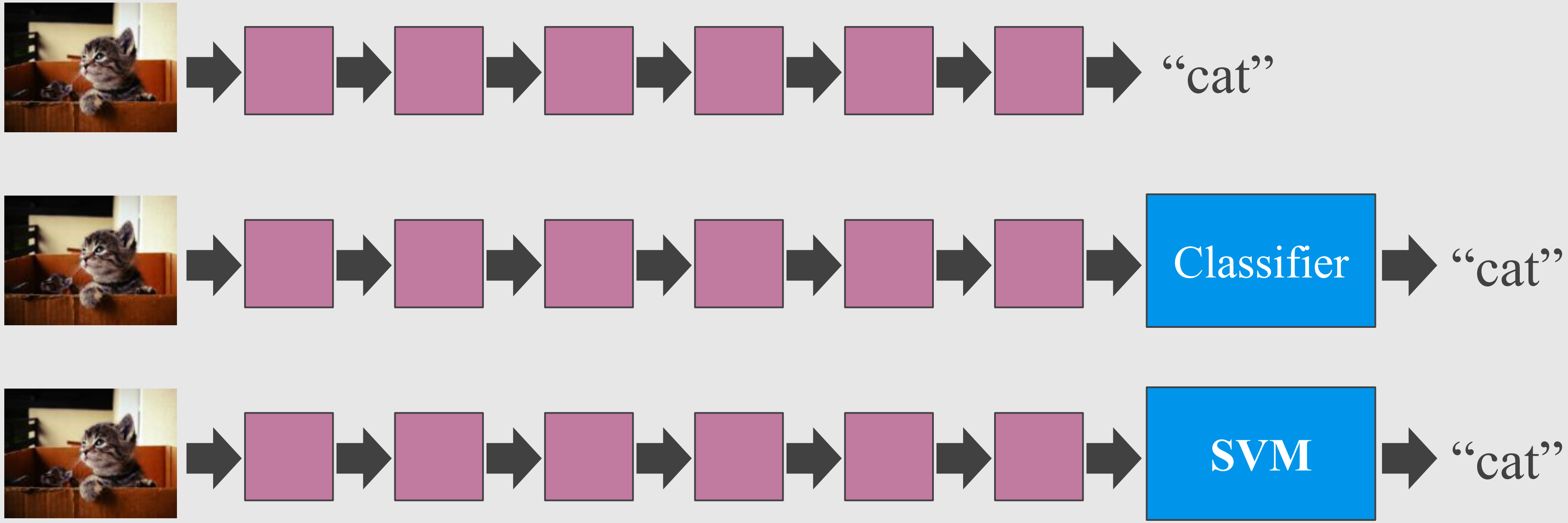
Deep Learning



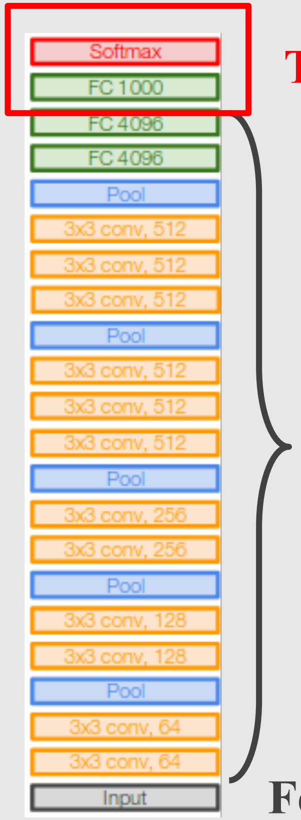
Deep Learning



Deep Learning



Transfer Learning

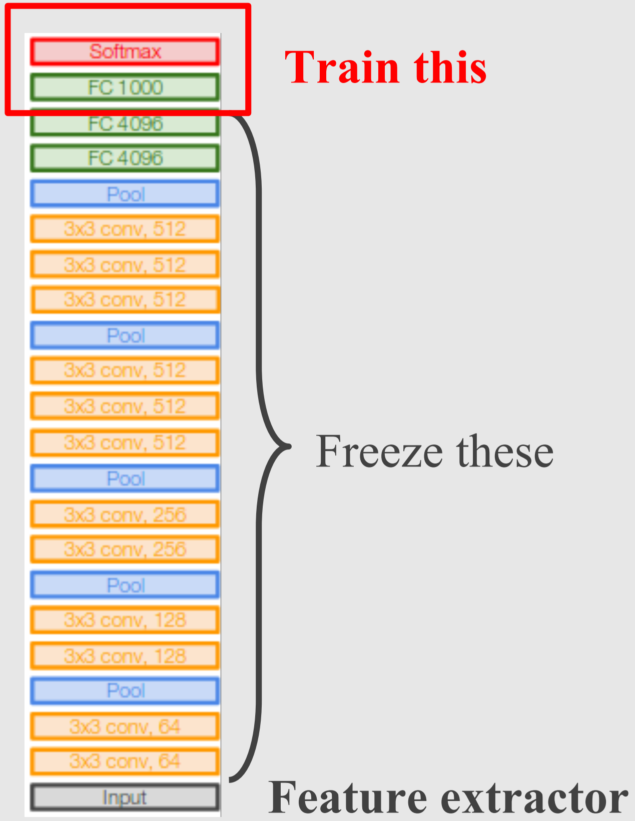


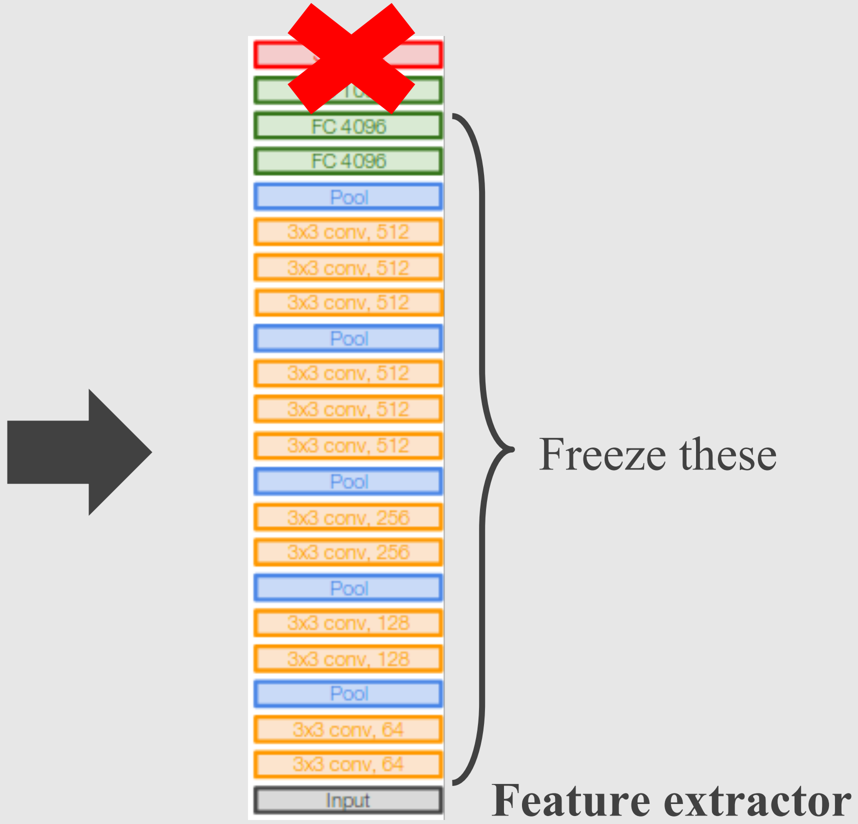
Train this

Freeze these

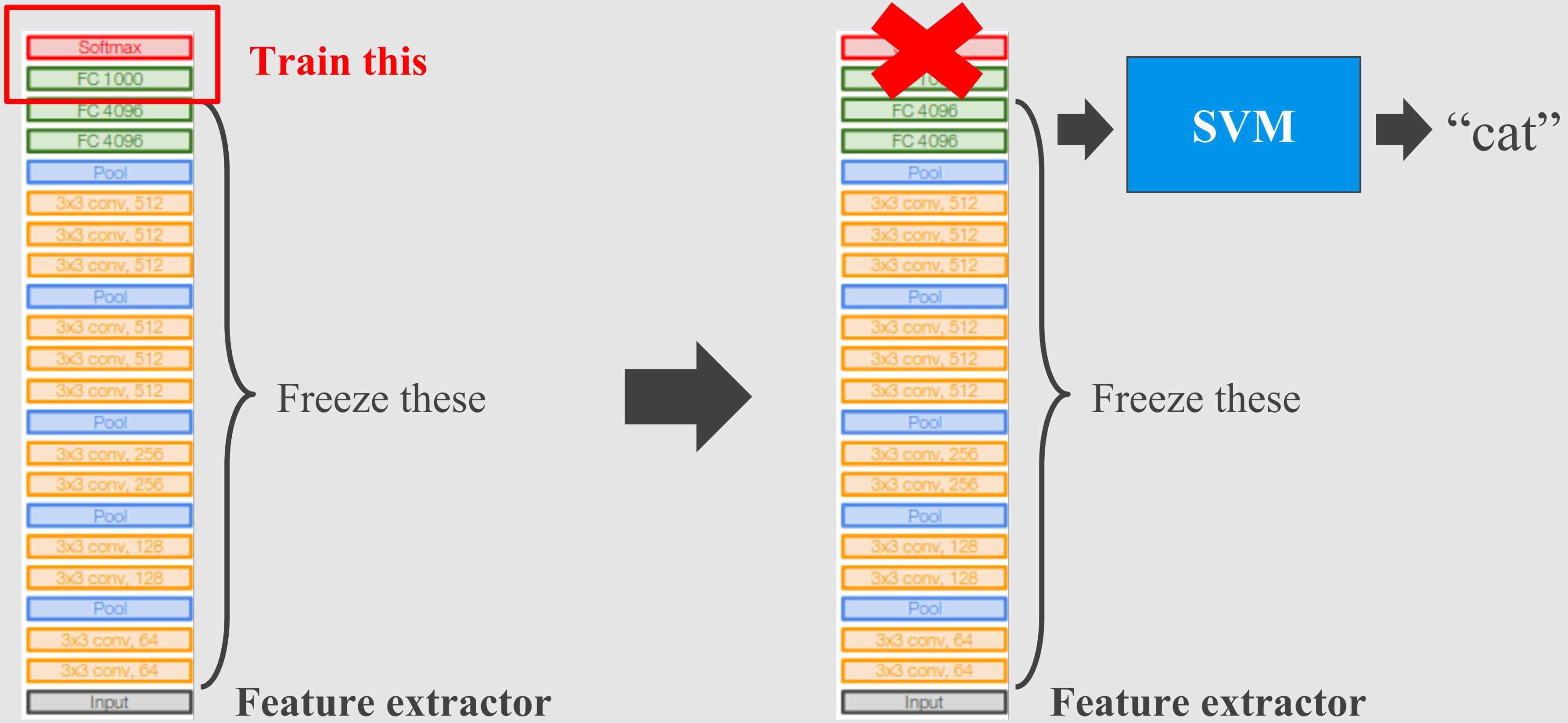
Feature extractor

Transfer Learning





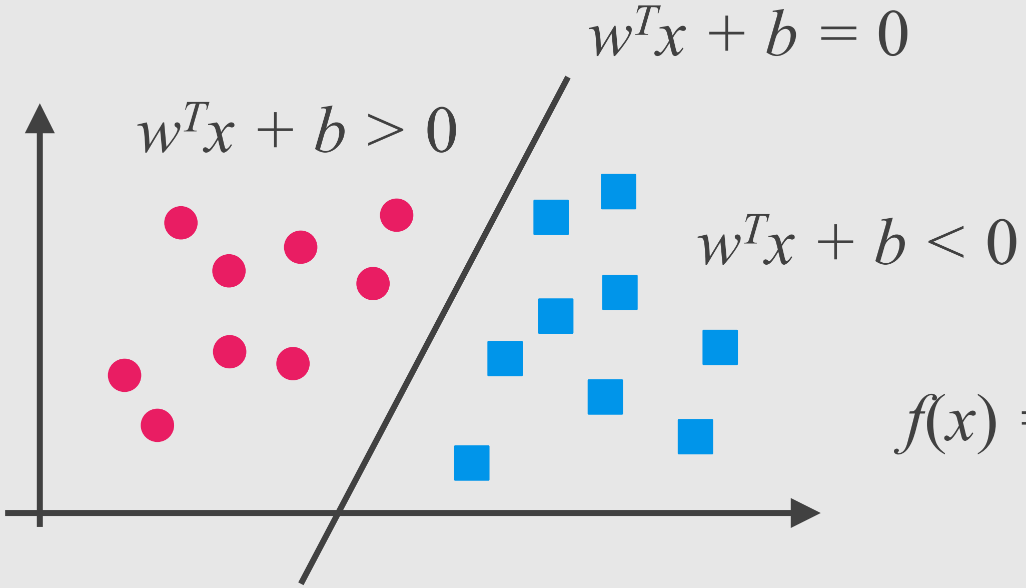
Transfer Learning



What is Support Vector Machine?

Linear Classifier

Binary classification can be viewed as the task of separating classes in feature space:

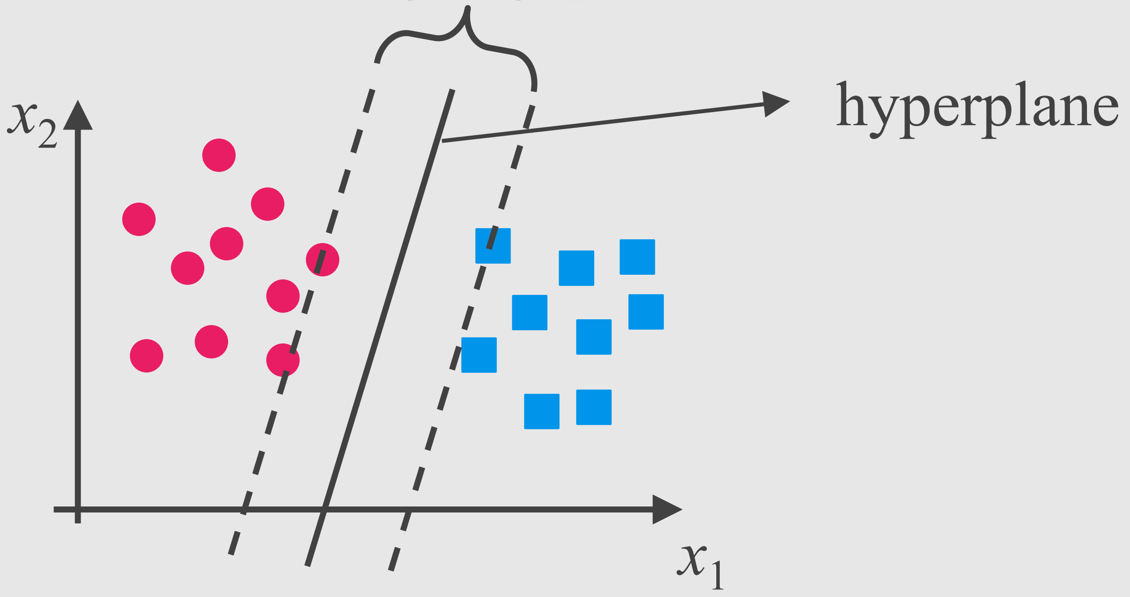


$f(x) = \text{sign}(w^Tx + b)$

Support Vector Machine

[Vapnik and Chervonenkis, 1964; Vapnik, 1982; Vapnik, 1995]

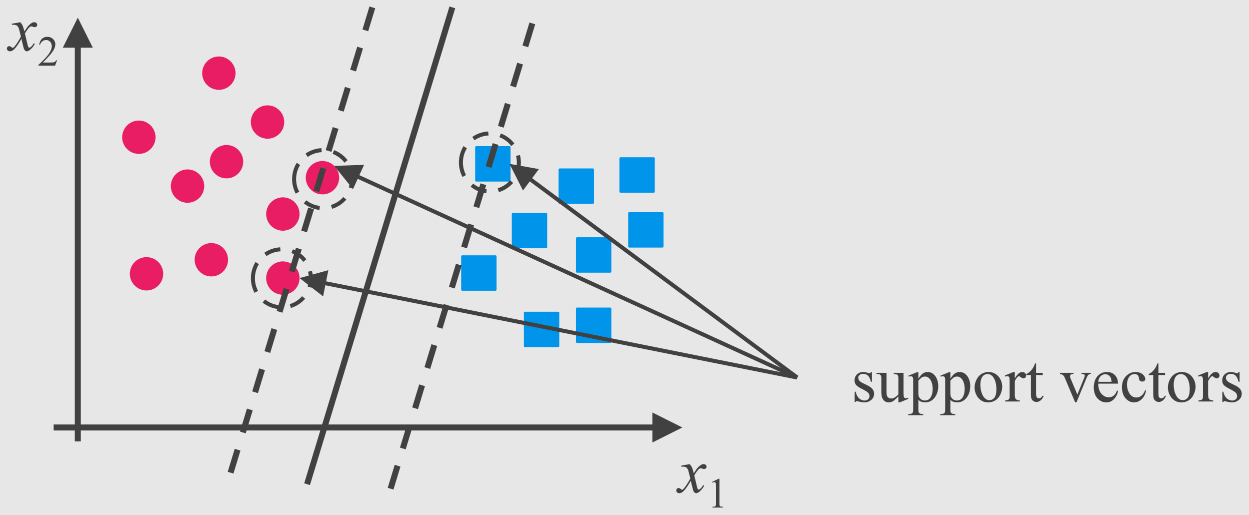
Idea of separating data with **a large "gap"**.



Support Vector Machine

[Vapnik and Chervonenkis, 1964; Vapnik, 1982; Vapnik, 1995]

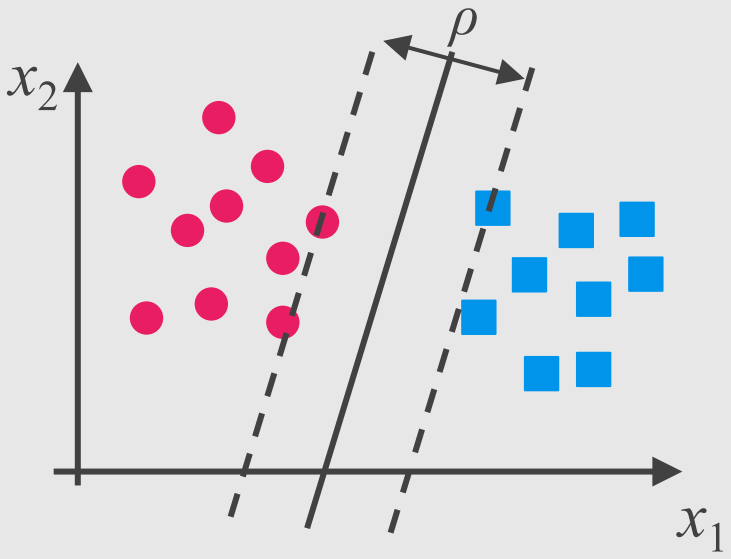
Examples closest to the hyperplane are support vectors.



Support Vector Machine

[Vapnik and Chervonenkis, 1964; Vapnik, 1982; Vapnik, 1995]

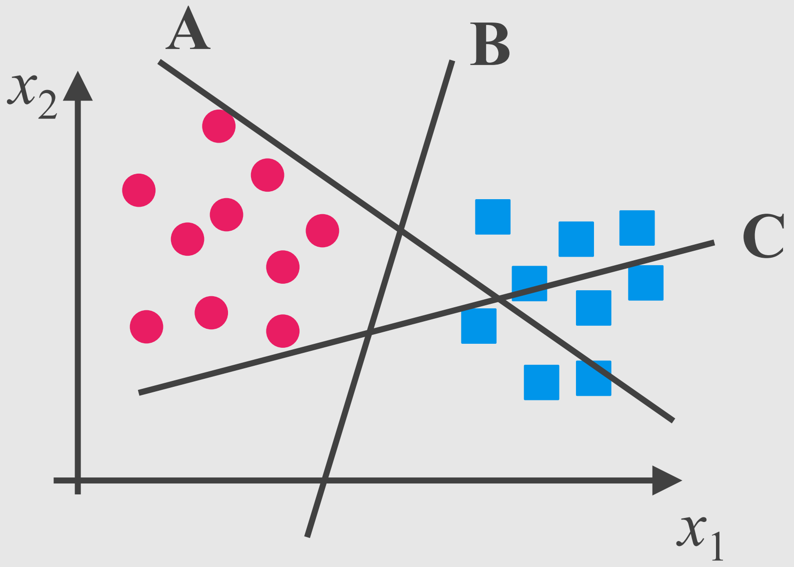
Margin ρ of the separator is the distance between support vectors.



How does SVM work?

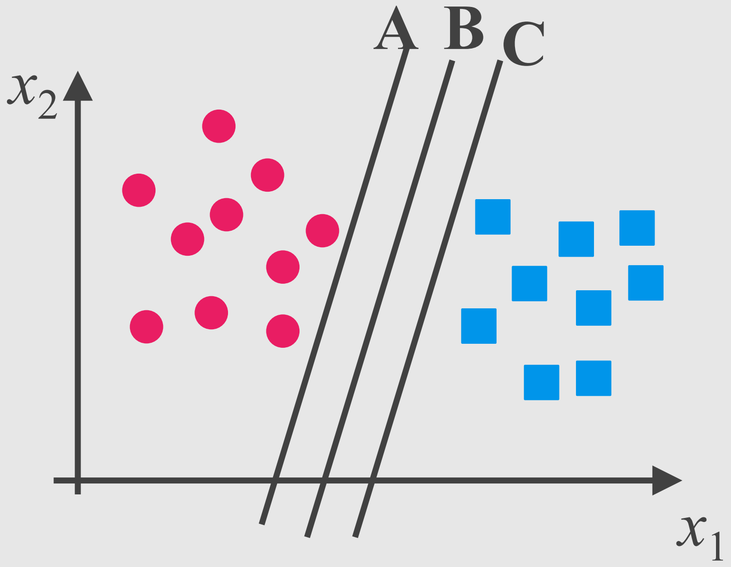
How can we identify the right hyperplane?

Scenario 1



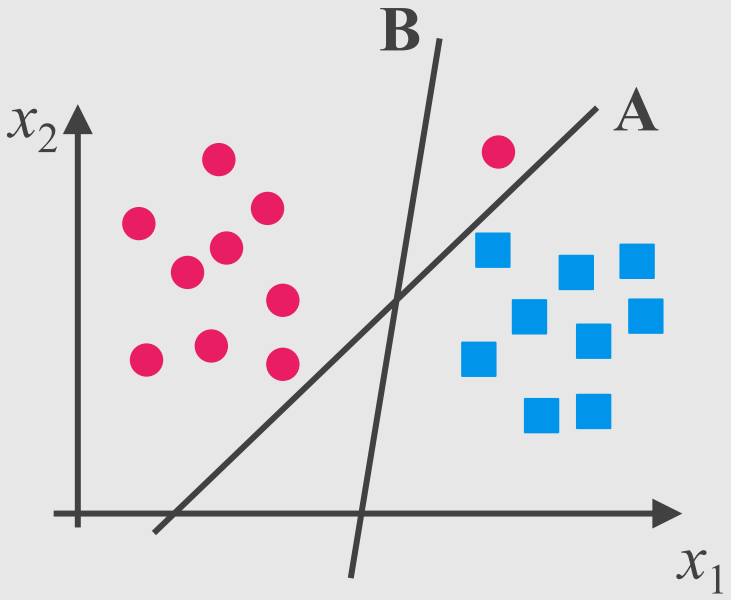
How can we identify the right hyperplane?

Scenario 2



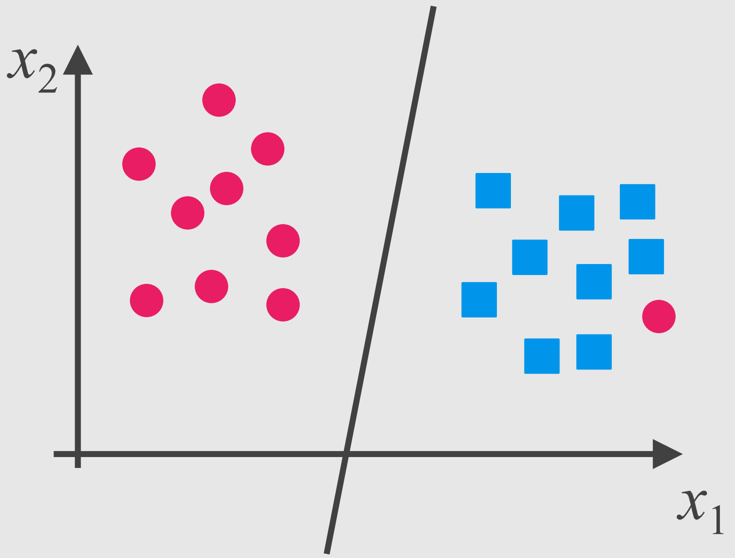
How can we identify the right hyperplane?

Scenario 3



How can we identify the right hyperplane?

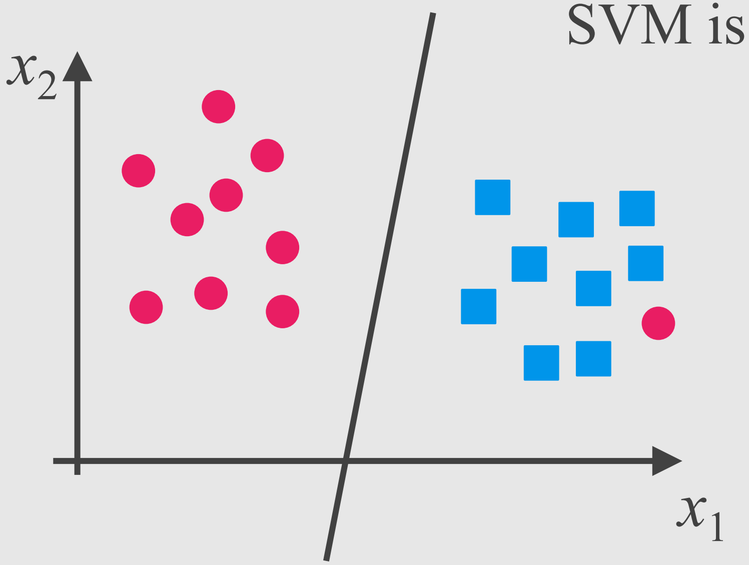
Scenario 4



How can we identify the right hyperplane?

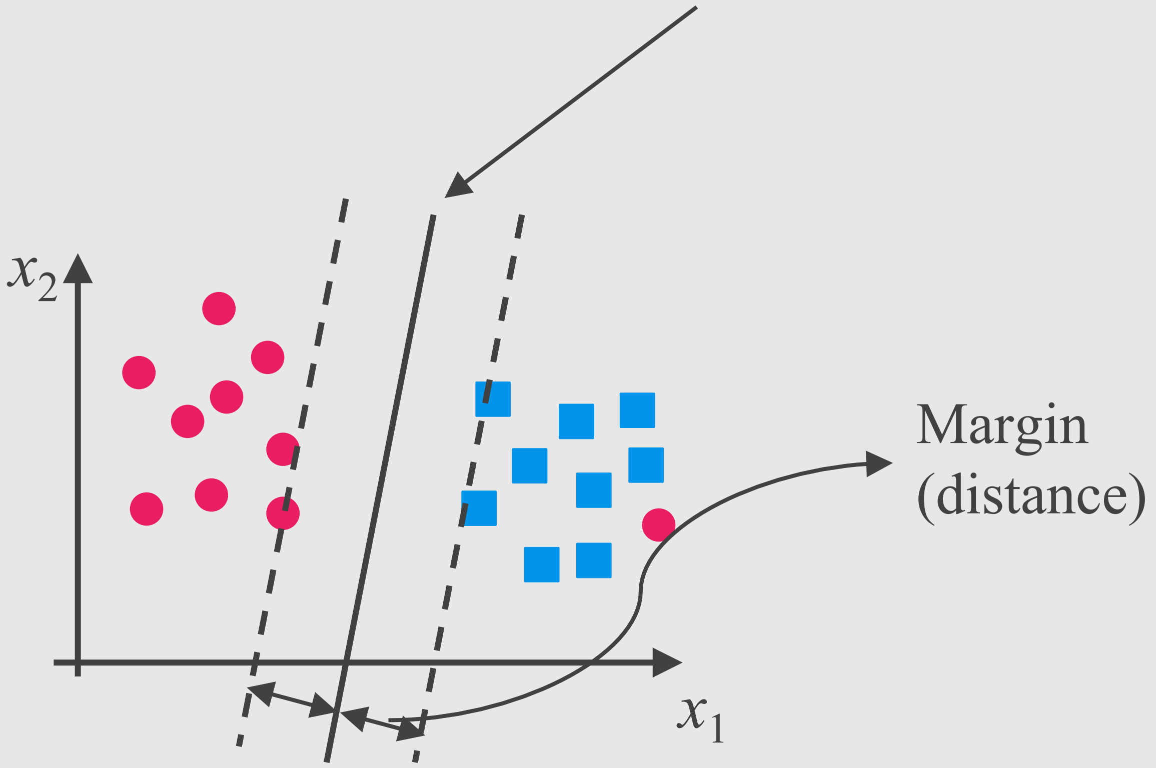
Scenario 4

SVM is robust to outliers



How can we identify the right hyperplane?

Scenario 4



SVM: Notation

We will be considering a **linear classifier for a binary classification** problem with labels y and features x.

SVM: Notation

We will be considering a **linear classifier for a binary classification** problem with labels y and features x.

- Class labels: $y \in \{-1,1\}$ (instead of $\{0,1\}$)
- Parameters: w, b (instead of vector θ)

SVM: Notation

We will be considering a **linear classifier for a binary classification** problem with labels y and features x.

- Class labels: $y \in \{-1,1\}$ (instead of $\{0,1\}$)
- Parameters: w, b (instead of vector θ)
- Classifier: $h_{w,b}(x) = g(w^T x + b)$
    - $g(z) = 1$ if $z \geq 0$, and $g(z) = -1$ otherwise

SVM: The Optimal Hyperplane

Let training set $\{(x_i, y_i)\}_{i=1..m}$, $x_i \in \Re^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ. Then for each training example (x_i, y_i):

$$\begin{array}{l} w^T x + b \leq -\rho/2 \text{ if } y_i = -1 \\ w^T x + b \geq \rho/2 \text{ if } y_i = 1 \end{array} \quad \Longleftrightarrow \quad y_i(w^T x + b) \geq \rho/2$$

SVM: The Optimal Hyperplane

For every support vector x_s the previous inequality is an equality. After rescaling w and b by ρ/2 in the equality, we obtain that distance between each x_s and the hyperplane is

$$\mathrm{r} = \frac{y_s\,(w^T x_s + b)}{||w||}$$

SVM: The Optimal Hyperplane

Then we can formulate the quadratic optimization problem:

$$\min_{w,b} \ \tfrac{1}{2}||w||^2$$
$$\text{s.t. } y_i(w^T x + b) \geq 1,\ i = \{1, \ldots, m\}$$

http://cs229.stanford.edu/notes/cs229-notes3.pdf

Need to optimize a quadratic function subject to linear constraints

SVM: The Optimal Hyperplane

The solution involves constructing a dual problem where a Lagrange multiplier α i is associated with every inequality constraint in the primal (original) problem:

Find $\alpha_i \ldots \alpha_n$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\, \alpha_i - \frac{1}{2} \Sigma\Sigma\, \alpha_i \alpha_j\, y_i y_j\, \boldsymbol{x_i}^T \boldsymbol{x_j}$ is maximized and

(1) $\Sigma\, \alpha_i y_i = 0$

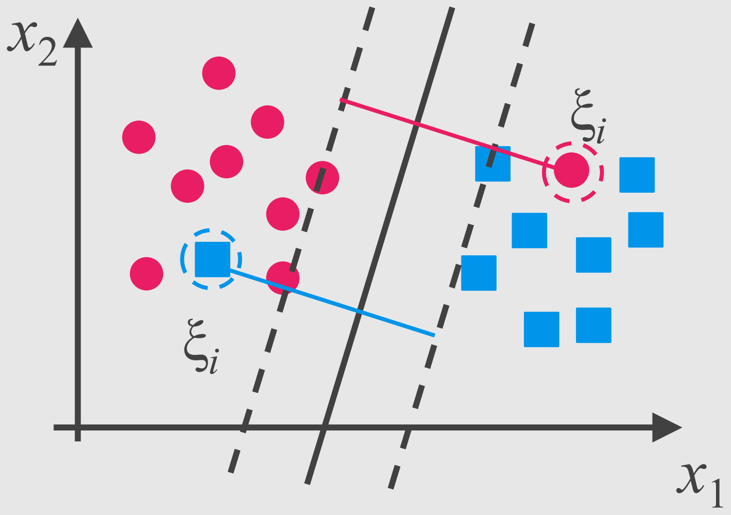
(2) $\alpha_i \geq 0$ for all α_i

Soft Margin Classification

What if the training set is not linearly separable?

Soft Margin Classification

Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called **soft**.



Soft Margin Classification

Modified formulation incorporates slack variables:

$$\boxed{\begin{gathered}\min_{w,b,\xi}\ \tfrac{1}{2}||w||^2\ + C\Sigma\xi_i \\ \text{s.t. } y_i(w^Tx + b) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = \{1, \ldots, m\}\end{gathered}}$$

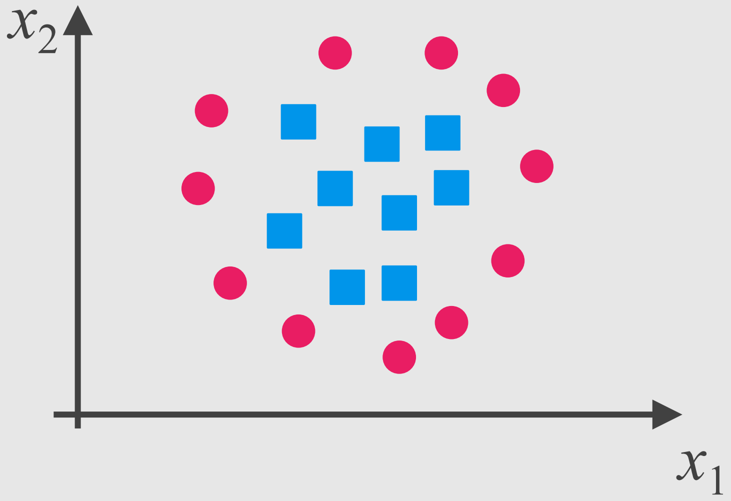
Parameter *C* can be viewed as a way to control overfitting:
it “trades off” the relative importance of maximizing the margin and fitting the training data.

Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrangian multipliers α_i.

How can we identify the right hyperplane?

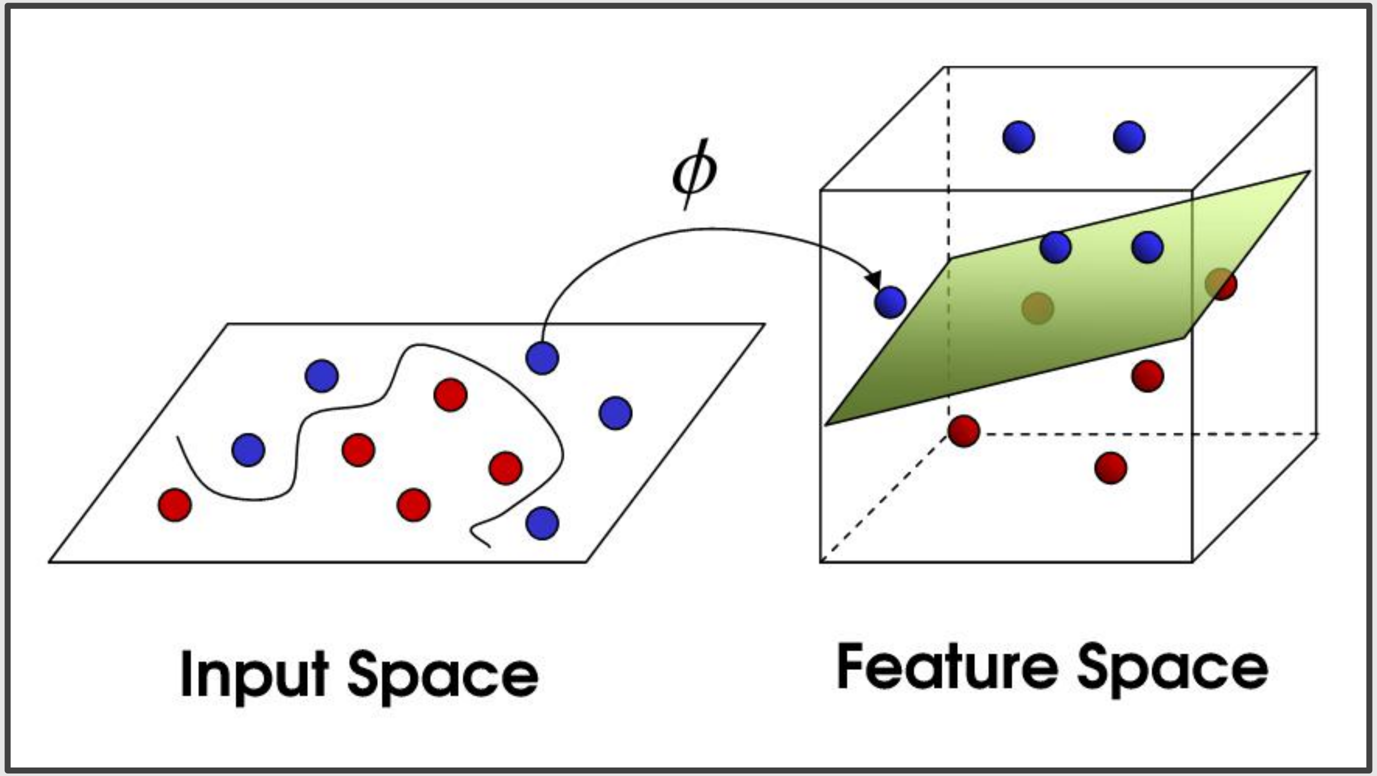
Scenario 5



Non-linear SVMs: Feature spaces

General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable.

$$\Phi : x \longrightarrow \varphi(x)$$



Kernel Trick

- The linear classifier relies on inner product between vector
  $K(x_i, x_j) = x_i^{\mathrm{T}} x_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \varphi(x)$, the inner product becomes:

$$\boldsymbol{K(x_i, x_j) = \varphi(x_i)^{\mathrm{T}} \varphi(x_j)}$$

- A kernel function is a function that is equivalent to an inner product in some feature space.

Kernelized SVM

Suppose you want to apply a 2nd degree polynomial transformation to a 2-dimensional training set, then train a linear SVM classifier on the transformed training set.

$$\varphi(x) = \varphi((x_1\ x_2)) = (x_1^2\ \ \sqrt{2}x_1x_2\ \ x_2^2)$$

The transformed vector is 3-dimensional instead of 2-dimensional.

Kernelized SVM

Let's look at what happens to a couple 2-dimensional vectors **a** and **b** if we apply this 2nd degree polynomial mapping then compute the dot product of the transformed vectors.

$$\varphi(\mathbf{a})^T \varphi(\mathbf{b}) =$$

Kernelized SVM

Let’s look at what happens to a couple 2-dimensional vectors **a** and **b** if we apply this 2nd degree polynomial mapping then compute the dot product of the transformed vectors.

$$\varphi(\mathbf{a})^T\varphi(\mathbf{b}) = (a_1^2 \;\; \sqrt{2}a_1a_2 \;\; a_2^2)^T(b_1^2 \;\; \sqrt{2}b_1b_2 \;\; b_2^2) =$$

Kernelized SVM

Let's look at what happens to a couple 2-dimensional vectors **a** and **b** if we apply this 2nd degree polynomial mapping then compute the dot product of the transformed vectors.

$$\begin{aligned}
\varphi(\mathbf{a})^{\mathrm{T}}\varphi(\mathbf{b}) &= (a_1{}^2 \;\; \sqrt{2}a_1a_2 \;\; a_2{}^2)^{\mathrm{T}}(b_1{}^2 \;\; \sqrt{2}b_1b_2 \;\; b_2{}^2) = \\
&= a_1{}^2b_1{}^2 + 2a_1a_2b_1b_2 + a_2{}^2b_2{}^2 = \\
&= (a_1b_1 + a_2b_2)^2 = \\
&= ((a_1 \; a_2)^{\mathrm{T}}(b_1 \; b_2))^2 = \\
&= (\mathbf{a}^{\mathrm{T}} \cdot \mathbf{b})^2
\end{aligned}$$

Kernelized SVM

The dot product of the transformed vectors is equal to the square of the dot product of the original vectors: $\varphi(\mathbf{a})^T\varphi(\mathbf{b}) = (\mathbf{a}^T \cdot \mathbf{b})^2$

Kernelized SVM

The dot product of the transformed vectors is equal to the square of the dot product of the original vectors: $\varphi(\mathbf{a})^T\varphi(\mathbf{b}) = (\mathbf{a}^T \cdot \mathbf{b})^2$

So **you don't actually need to transform the training instances** at all: just replace the dot product by its square.

This is the essence of the kernel trick.

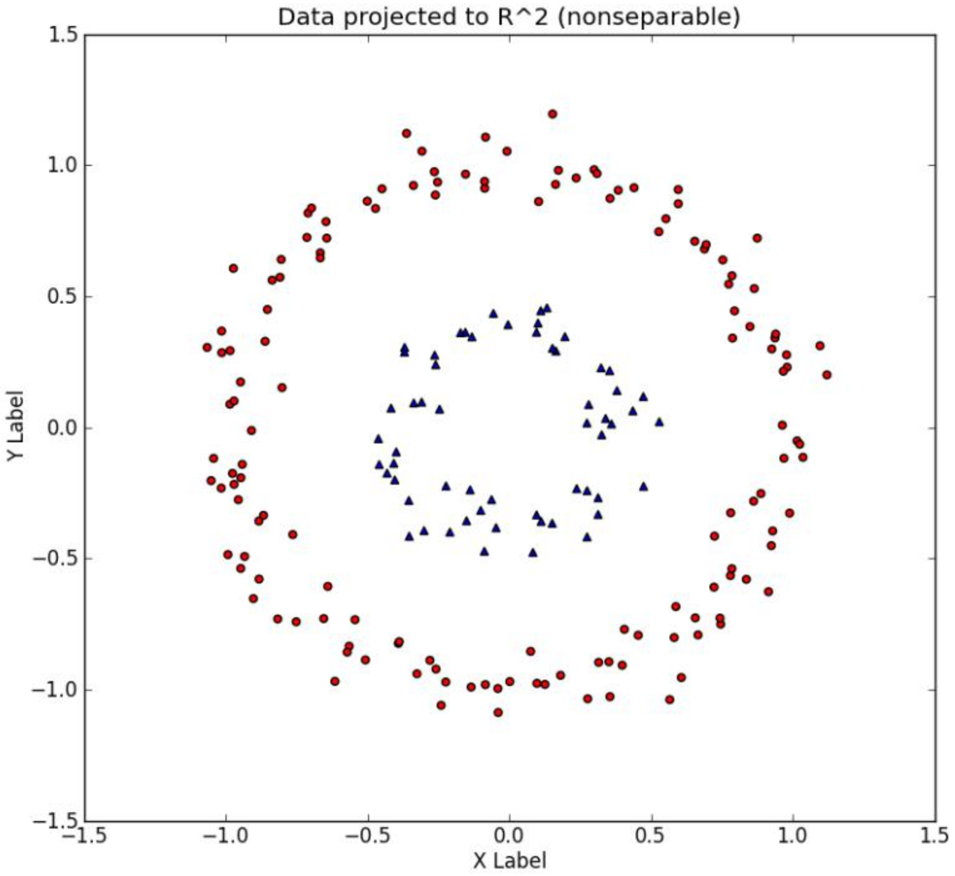
Kernelized SVM

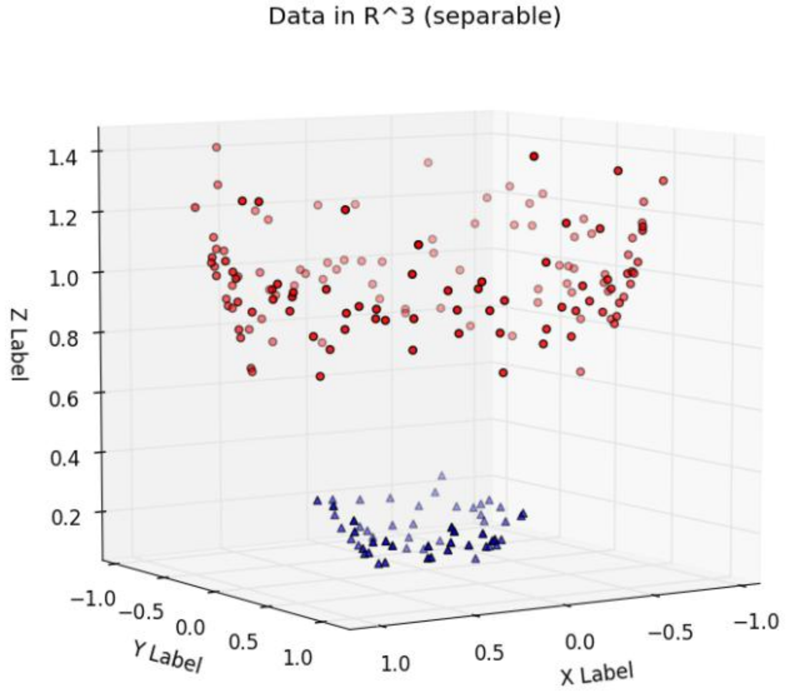
In Machine Learning, a ***kernel*** is a function capable of computing the dot product $\varphi(\mathbf{a})^T\varphi(\mathbf{b})$ based only on the original vectors **a** and **b**, without having to compute (or even to know about) the transformation φ.

Kernel Functions

- Linear SVM = Kernel SVM: $K(x_i, x_j) = x_i^T x_j$
- Nonlinear SVM: $K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$

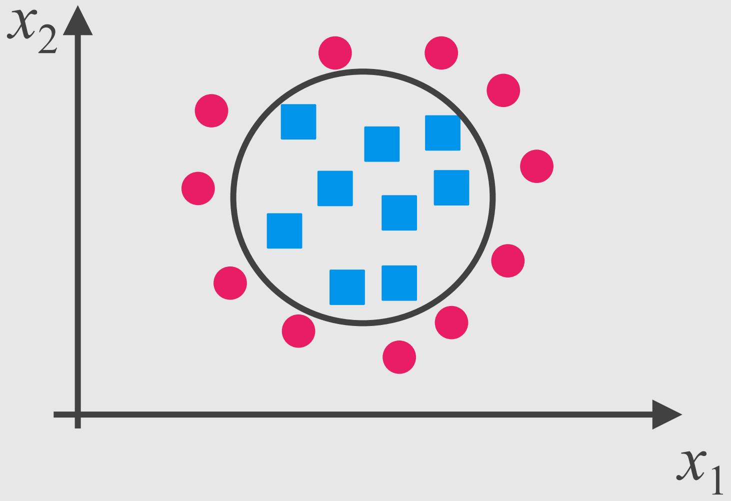
- Gaussian kernel: $K(x_i, x_j) = \exp(-\|x_i - x_j\|^2/(2\sigma^2))$
- Polynomial kernel: $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$, d degree
- Chi-square kernel, histogram intersection kernel, string kernel, ...


Data projected to R^2 (nonseparable)
X Label
Y Label
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5


Data in R^3 (separable)
Z Label
Y Label
X Label
1.4
1.2
1.0
0.8
0.6
0.4
0.2
−1.0
−0.5
0.0
0.5
1.0

How can we identify the right hyperplane?

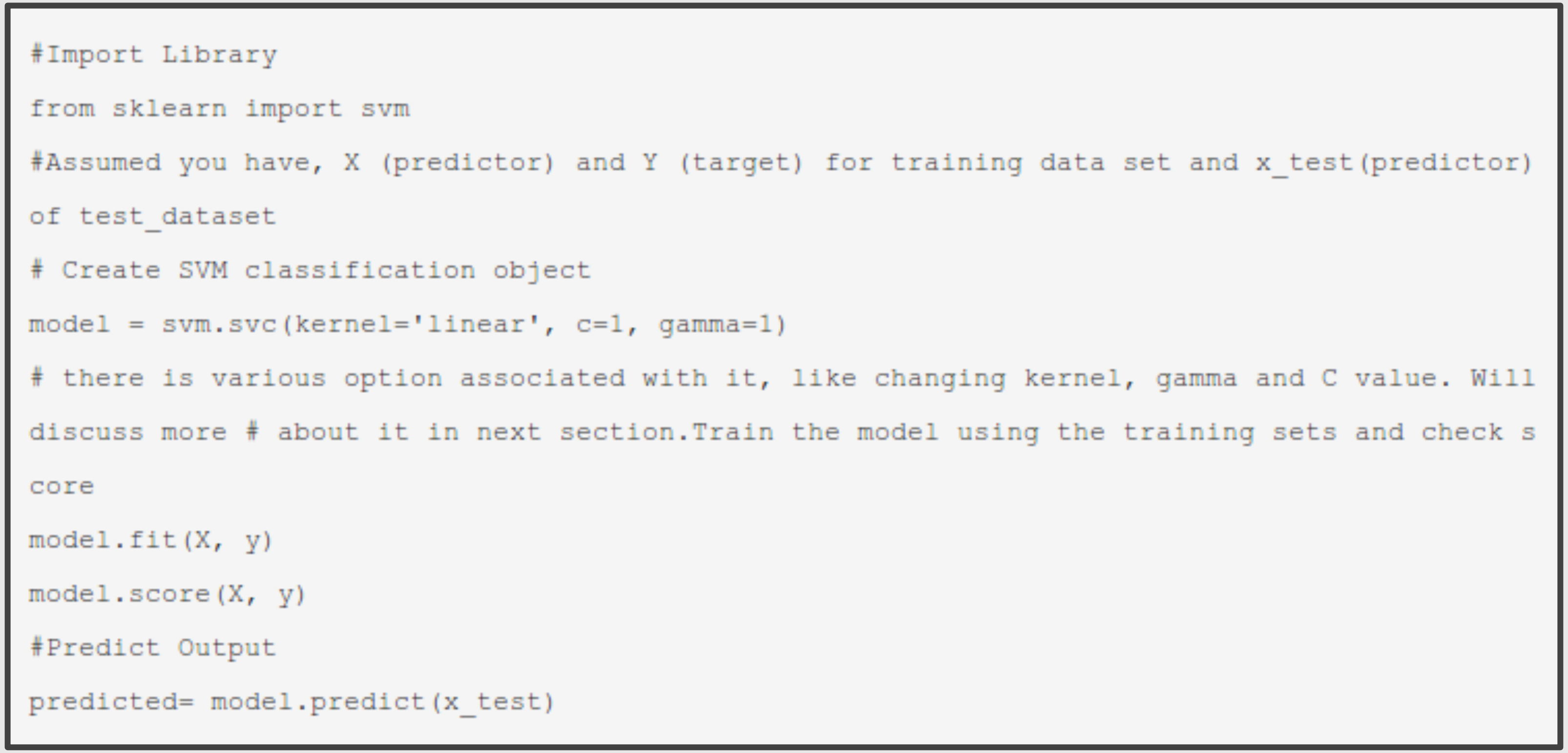
Scenario 5



SVM & scikit-learn

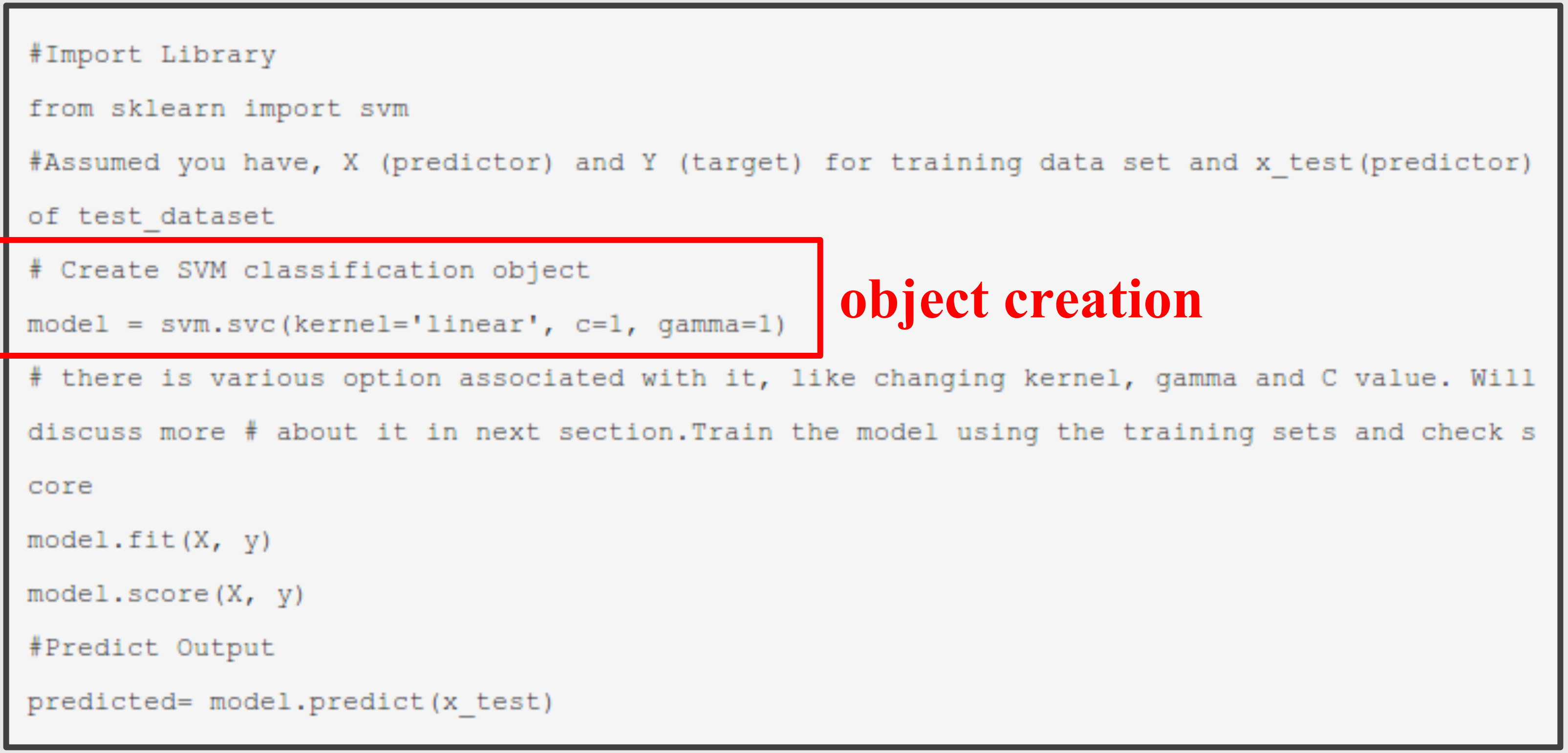
SVM is also available in scikit-learn library and follow the same structure : import library, object creation, fitting model and prediction.

SVM & scikit-learn



```
#Import Library

from sklearn import svm

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor)

of test_dataset

# Create SVM classification object

model = svm.svc(kernel='linear', c=1, gamma=1)

# there is various option associated with it, like changing kernel, gamma and C value. Will

discuss more # about it in next section.Train the model using the training sets and check s

core

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

SVM & scikit-learn



```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor)
of test_dataset
# Create SVM classification object
model = svm.svc(kernel='linear', c=1, gamma=1)
# there is various option associated with it, like changing kernel, gamma and C value. Will
discuss more # about it in next section.Train the model using the training sets and check s
core
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

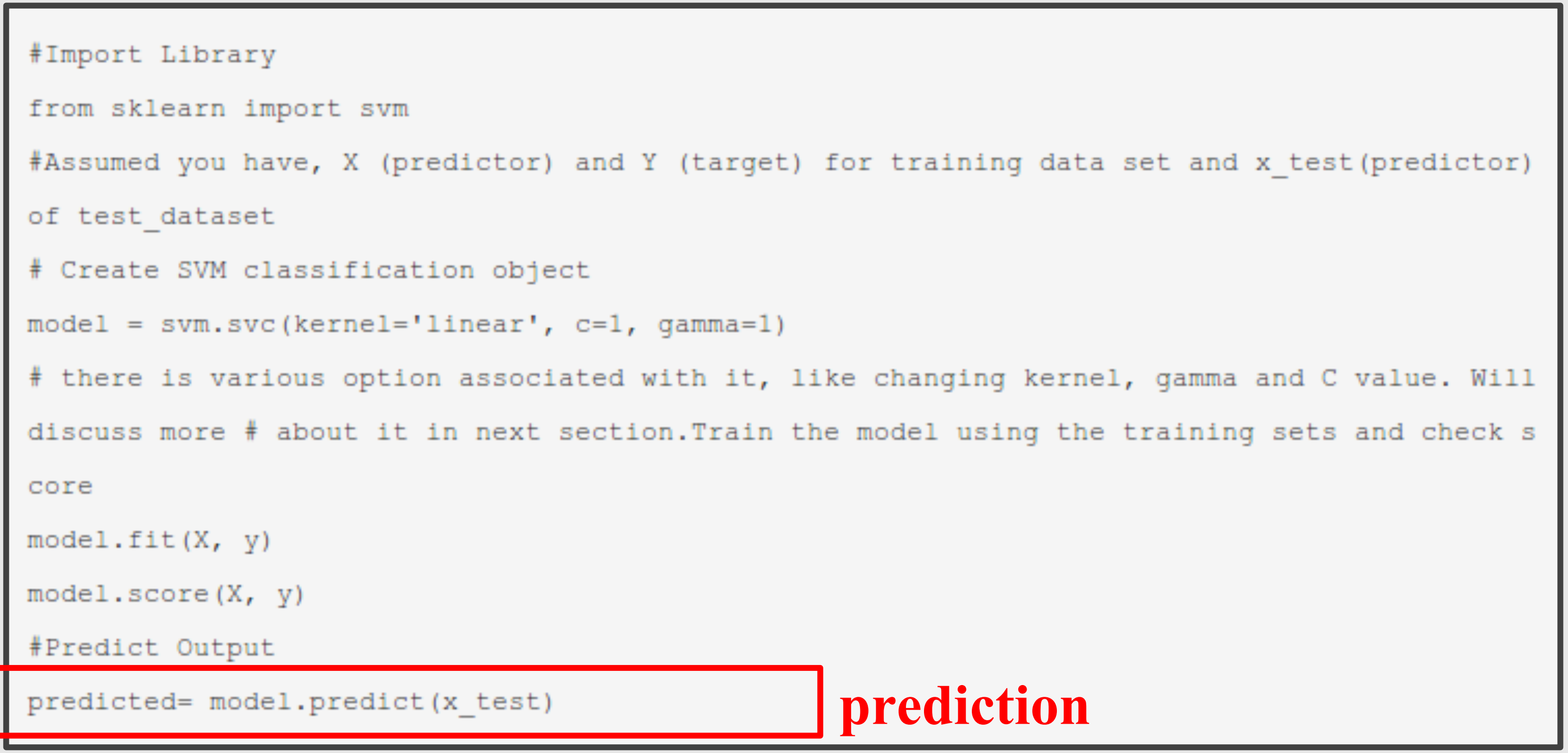
object creation

SVM & scikit-learn

```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor)
of test_dataset
# Create SVM classification object
model = svm.svc(kernel='linear', c=1, gamma=1)
# there is various option associated with it, like changing kernel, gamma and C value. Will
discuss more # about it in next section.Train the model using the training sets and check s
core
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

fitting model

SVM & scikit-learn



```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor)
of test_dataset
# Create SVM classification object
model = svm.svc(kernel='linear', c=1, gamma=1)
# there is various option associated with it, like changing kernel, gamma and C value. Will
discuss more # about it in next section.Train the model using the training sets and check s
core
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

prediction

Important Parameters

Important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

Important Parameters

Important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

C: Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.

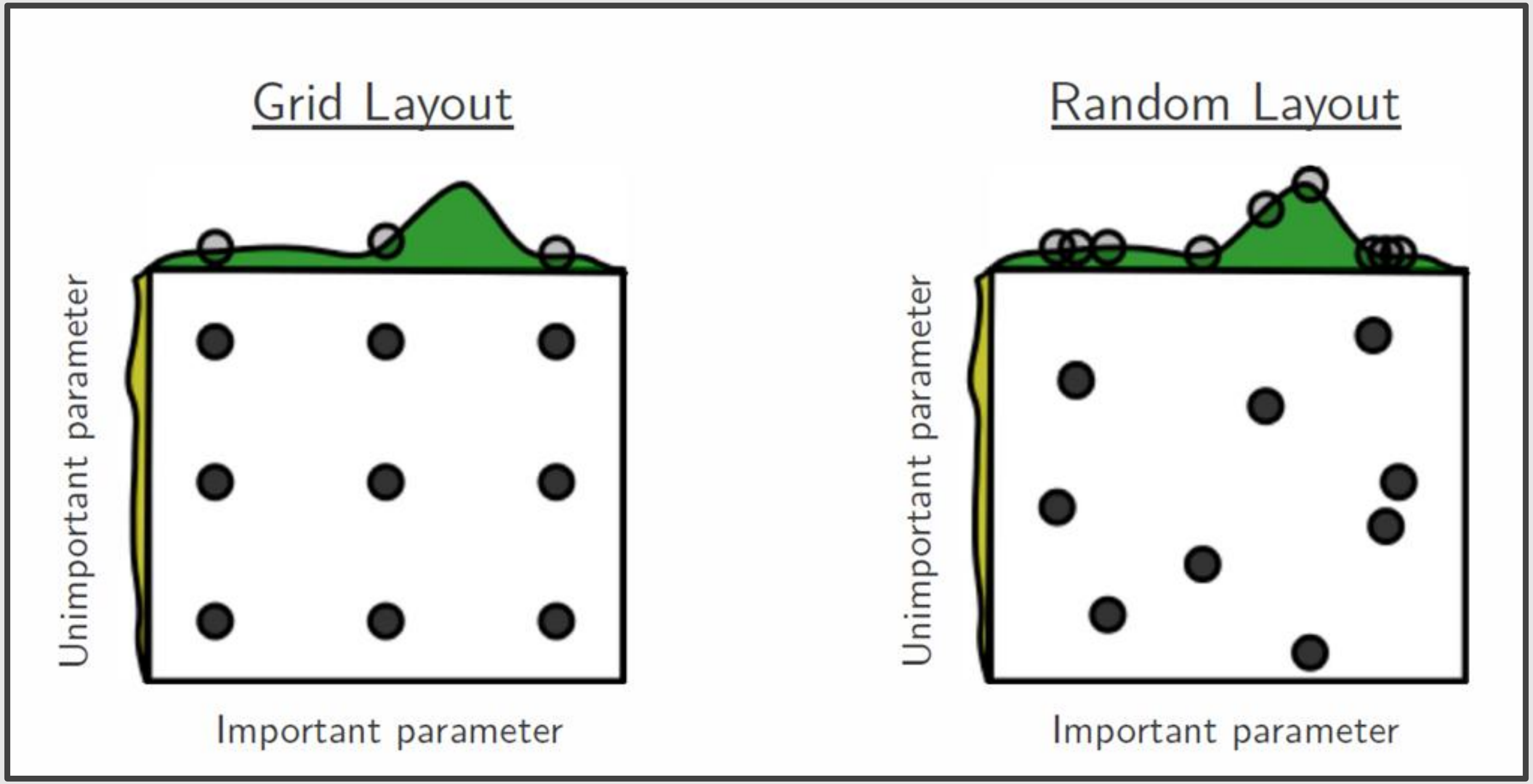
Important Parameters

Important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

C: Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.

The parameters can be tuned using grid-search.

Grid Search


Grid Layout
Random Layout
Unimportant parameter
Important parameter
Unimportant parameter
Important parameter

References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 5
- Pattern Recognition and Machine Learning, Chap. 6 & 7

Machine Learning Courses

- https://www.coursera.org/learn/machine-learning, Week 7
- http://cs229.stanford.edu/syllabus.html, http://cs229.stanford.edu/notes/cs229-notes3.pdf