CITS5508 Machine Learning

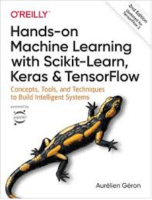
Dr Débora Corrêa (Unit Coordinator and Lecturer)

2023

Today

Chapter 5.

Hands-on Machine Learning with Scikit-Learn & TensorFlow


O'REILLY
2nd Edition
Updated for TensorFlow 2
Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
powered by
jupyter
Aurélien Géron

Chapter Five

Support Vector Machines

In this chapter, we will explain the core concepts of SVMs, how to use them, and how they work.

Here are the main topics we will go cover:

- Linear SVM Classification
- Nonlinear SVM Classification
- SVM Regression
- Under the Hood

Support Vector Machine (SVMs)

A Support Vector Machine (SVM) is a versatile Machine Learning model: it provides linear and nonlinear decision boundaries.

Used for (binary) classification and regression.

Particularly well suited for classification of complex but small- or medium-sized datasets.

Large margin classification: it maximizes the distance to the nearest points relative to both classes.

Linear SVM Classification

You can think of an SVM classifier as fitting the widest possible street (represented by the parallel dashed lines) between the classes. This is called *large margin classification*.

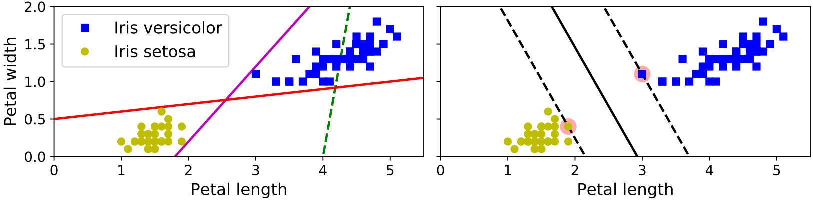


Figure 5-1. Large margin classification

Linear SVM Classification

Notice that adding more training instances “off the street” will not affect the decision boundary at all: it is fully determined (or “supported”) by the instances located on the edge of the street. These instances are called the *support vectors*.

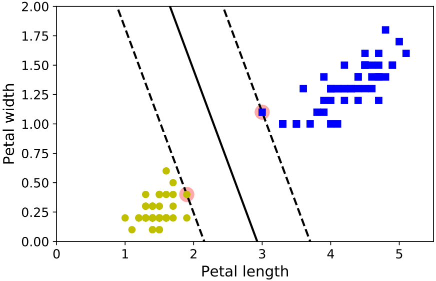


Figure 5-1. Large margin classification

Sensitive to Scale

Feature scaling is important. Remember you should fit the scalers using the training set only, and use the estimated parameters to transform the validation and test sets.

SVMs are sensitive to the feature scales. After feature scaling (e.g., using Scikit-Learn's `StandardScaler`), the decision boundary looks much better (on the right plot).

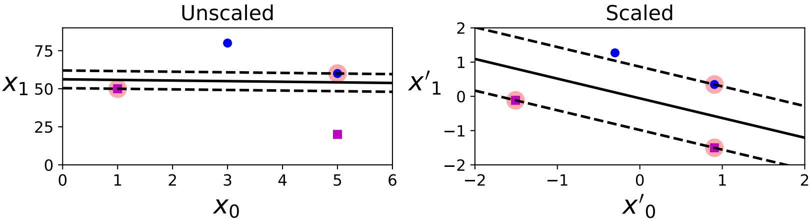
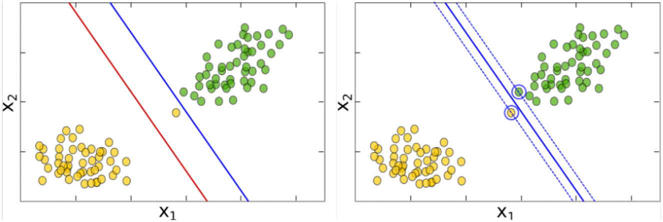


Figure 5-2. Sensitivity to feature scales

Hard Margin Classification

Strictly impose that all instances be off the street and on the correct side.



First the correct classification, then maximize the margin.

Hard Margin Classification (cont.)

- *Hard margin classification* only works if the data is linearly separable.
- Furthermore, it is quite sensitive to outliers.

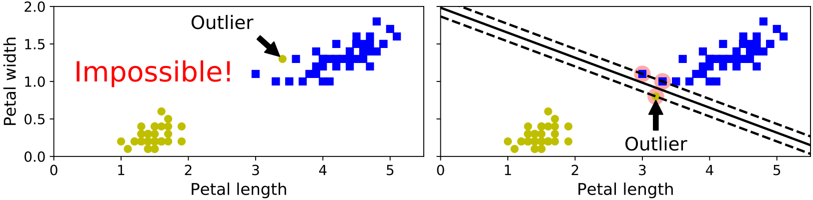


Figure 5-3. Hard margin sensitivity to outliers

Soft Margin Classification - more flexible model

The objective of *soft margin classification* is to find a good balance between keeping the street as large as possible and limiting the margin violations.

The hyperparameter C allows us to define the trade-off between the two objectives.

A small C leads to “wider street” and more margin violations.

A large C leads to “narrower street” and less margin violations.

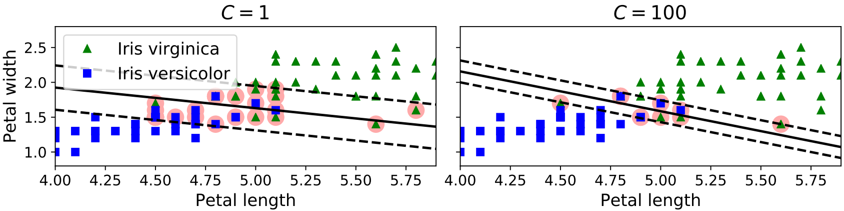


Figure 5-4. Large margin (left) versus fewer margin violations (right)

Soft Margin Classification – An example

```
from sklearn.datasets import load_iris
from sklearn.datasets import load_iris
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 2) # Iris virginica

svm_clf = make_pipeline(StandardScaler(),LinearSVC(C=1, random_state=42))
svm_clf.fit(X, y) # train the SVM classifier

>>> X_new = [[5.5, 1.7], [5.0, 1.5]]
>>> svm_clf.predict(X_new)
array([ True, False])
```

We can also the scores SVM used to make these predictions.

```
>>> svm_clf.decision_function(X_new)
array([ 0.66163411, -0.22036063])
```

Nonlinear Decision Boundaries – using Polynomial Features

Many datasets are not even close to being linearly separable.
One approach to handling nonlinear datasets is to add more features, such as polynomial features.
In some cases this can result in a linearly separable dataset.

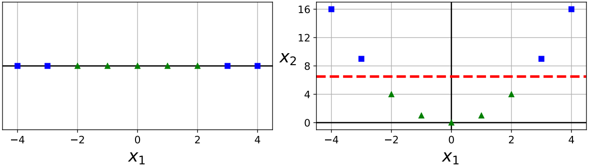


Figure 5-5. Adding features to make a dataset linearly separable

In the example, $x_2 = (x_1)^2$.

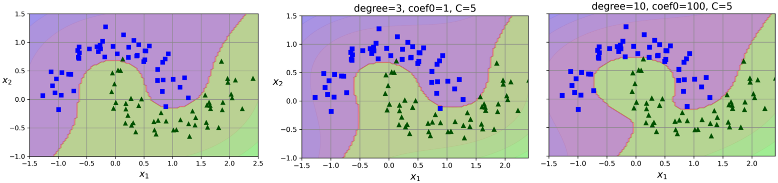
Polynomial Kernel

Using polynomial features can make the model training very slow.

When using SVMs you can apply a mathematical technique called the *kernel trick*.

It makes it possible to get the similar result as if you added many polynomial features.

```
SVC(kernel="poly", degree=3, coef0=1, C=5)
```



Adding Similarity Features

Another option to tackle nonlinear problems is to add features computed using a similarity function that measures how much each instance resembles a particular landmark. For example, let's take the one-dimensional dataset discussed earlier and add two landmarks to it at $x_1 = -2$ and $x_1 = 1$ (left plot). Next, let's define the similarity function to be the *Gaussian Radial Basis Function* (*RBF*) with $\gamma = 0.3$.

$$\phi_\gamma(x, \ell) = \exp(-\gamma \, \|x - \ell\|^2)$$

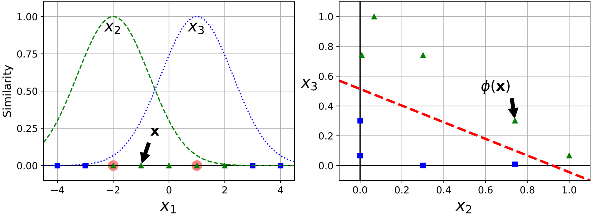


Figure 5-8. Similarity features using the Gaussian RBF

Gaussian RBF Kernel

The kernel trick again makes it possible to obtain a similar result as if you had added many similarity features, without actually having to add them.

Try the Gaussian RBF kernel using the SVC class.

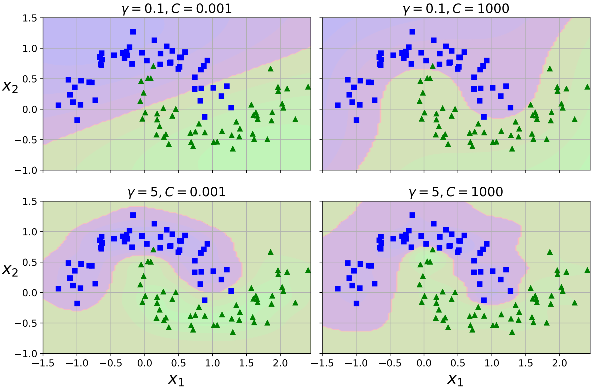
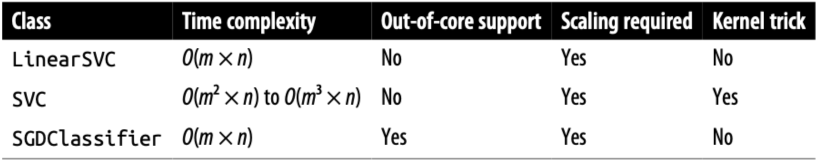


Figure 5-9. SVM classifiers using an RBF kernel

Computational Complexity

- LinearSVC implements an optimized algorithm for *linear SVMs*. No kernel trick, but it scales almost linearly with the number of training instances and the number of features.
- SVC implements an algorithm that supports the kernel trick. Unfortunately, high training time complexity means that it gets dreadfully slow when the number of training instances gets large.

Table 5-1. Comparison of Scikit-Learn classes for SVM classification



Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
`LinearSVC`	$O(m \times n)$	No	Yes	No
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes
`SGDClassifier`	$O(m \times n)$	Yes	Yes	No

SVM Regression

Instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible on the street while limiting margin violations (i.e., instances off the street).

```
from sklearn.svm import LinearSVR
svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

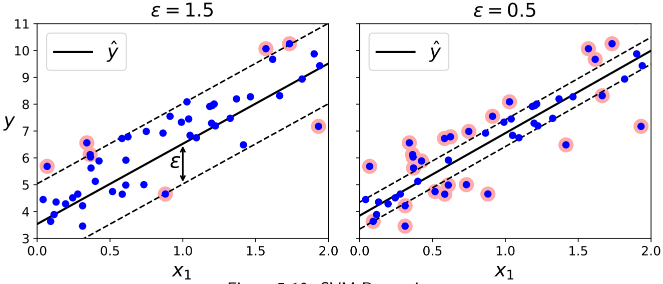


Figure 5-10. SVM Regression

Non-linear Regression

For nonlinear regression use a kernelized SVM model. E.g SVM Regression on a random quadratic training set, using a 2nd-degree polynomial kernel. There is little regularization on the left plot (i.e., a large C value), and much more regularization on the right plot (i.e., a small C value).

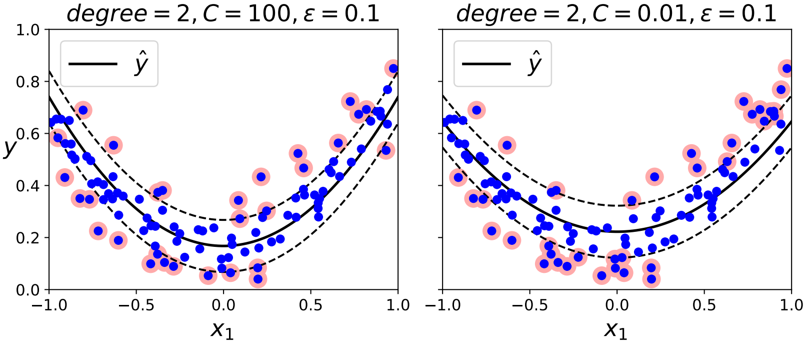


Figure 5-11. SVM Regression using a second-degree polynomial kernel

Under the Hood

Rest of chapter explains how SVMs make predictions and how their training algorithms work.
This has some heavy maths, beyond the scope of the course. We just take a high level overview.

Note: In this chapter, the bias term will be called b and the feature weights vector will be called $\mathbf{w}$.

Decision Function and Predictions

The linear SVM classifier model predicts the class of a new instance $\mathbf{x}$ by computing the decision function

$$\boldsymbol{\theta}^T\mathbf{x} = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n$$

where x_0 is the bias feature (always equal to 1). It is common to separate the bias term b (equal to θ_0) and the feature weights vector $\mathbf{w}$ (containing θ_1 to θ_n), so that the decision function is:

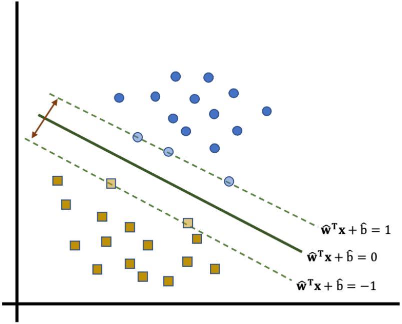
$$\hat{y} = \mathbf{w}^T\mathbf{x} + b = w_1 x_1 + \cdots + w_n x_n + b$$

and predict the class label $\hat{y}$ as follows:

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \\ 1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \end{cases}$$

Training Objective

The hard margin linear SVM classifier objective can be expressed as a constrained optimzation problem:


$\hat{\mathbf{w}}^{\mathbf{T}}\mathbf{x} + \hat{b} = 1$
$\hat{\mathbf{w}}^{\mathbf{T}}\mathbf{x} + \hat{b} = 0$
$\hat{\mathbf{w}}^{\mathbf{T}}\mathbf{x} + \hat{b} = -1$

$$\underset{\mathbf{w},b}{\text{minimize}} \ \frac{1}{2}\mathbf{w}^\top \mathbf{w}$$

$$\text{subject to: } t^{(i)}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b\right) \geq 1,$$

$$\text{for } i = 1, 2, \cdots, m$$

where $t^{(i)} = \begin{cases} +1 & \text{if training data } i \text{ is a positive instance} \\ -1 & \text{if training data } i \text{ is a negative instance} \end{cases}$

Training Objective

Width of the street: to make it larger, we need to make **w** smaller.

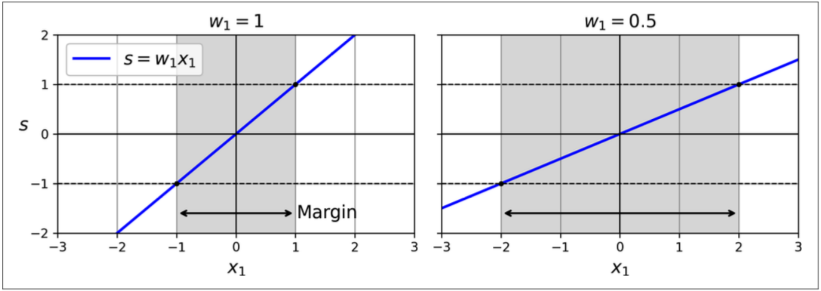


Figure 5-12. A smaller weight vector results in a larger margin

Training Objective

For the soft margin objective, we need to introduce a *slack variable* $\zeta^{(i)} \geq 0$ for each instance: $\zeta^{(i)}$ measures how much the i^{th} instance is allowed to violate the margin.

C is the hyperparameter which allows us to define the trade-off between the two objectives $\mathbf{w}^\top \mathbf{w}$ and $\sum_{i=1}^{m} \zeta^{(i)}$.

$$\underset{\mathbf{w},b,\zeta}{\text{minimize}} \ \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^{m} \zeta^{(i)}$$

$$\text{subject to: } t^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \text{ and } \zeta^{(i)} \geq 0, \text{ for } i = 1, 2, \cdots, m$$

where

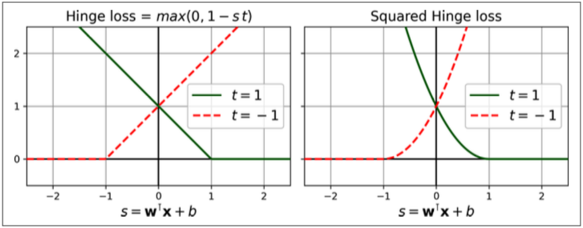
$$t^{(i)} = \begin{cases} +1 & \text{if training data } i \text{ is a positive instance} \\ -1 & \text{if training data } i \text{ is a negative instance} \end{cases}$$

Quadratic Programming

The hard margin and soft margin problems are both convex quadratic optimization problems with linear constraints. Such problems are known as *Quadratic Programming (QP) problems*.

Many off-the-shelf solvers are available to solve QP problems. We can also use gradient descent to minimize the hinge loss (or the squared hinge loss), with decision function $s = \mathbf{w}^T \mathbf{x} + b$.

The hinge loss is defined as $\text{hinge\_loss}(y) = \max(0, 1 - st)$, where $t = \pm 1$ is the ground truth label.



The Dual Problem

Given a constrained optimization problem, known as the *primal problem*, it is possible to express a different but closely related problem, called its *dual problem*.

The dual form of the linear SVM objective is:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \mathbf{x}^{(i)T} \mathbf{x}^{(j)} - \sum_{i=1}^{m} \alpha^{(i)}$$

subject to all $\alpha^{(i)} \geq 0$, for $i = 1, \cdots, m$ and $\sum_{i=1}^{m} \alpha^{(i)} t^{(i)} = 0$.

The dual problem is faster to solve than the primal when the number of training instances is smaller than the number of features. More importantly, it makes the *kernel trick* possible, while the primal does not.

From the dual solution to the primal solution

We use vector $\hat{\boldsymbol{\alpha}}$ that minimizes the dual form (using a QP solver), to compute the $\hat{\mathbf{w}}$ and $\hat{b}$ that minimize the primal problem (n_s represents the number of support vectors).

$$\hat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \mathbf{x}^{(i)}$$

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \hat{\mathbf{w}}^T \mathbf{x}^{(i)} \right)$$

Kernalized SVM, "The Kernel Trick"

Suppose that we want to transform our features $(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)})$ using the function ϕ (e.g., ϕ can be a 2nd degree polynomial) to a higher dimensional space. We then train a linear SVM classifier on the transformed training set.

Using the dual form, in the new transformed space, it seems that we will need to compute the dot product $\phi(\mathbf{x}^{(i)})^T\phi(\mathbf{x}^{(j)})$. But, if ϕ is the 2nd-degree polynomial transformation above, we can replace this dot product of the transformed vectors by $(\mathbf{x}^{(i)T}\mathbf{x}^{(j)})^2$. Let's exemplify using a couple of 2D vectors, $\mathbf{x}^{(i)} = \mathbf{a}$ and $\mathbf{x}^{(j)} = \mathbf{b}$:

$$\begin{aligned}
\phi(\mathbf{a})^T\phi(\mathbf{b}) &= \begin{bmatrix} {a_1}^2 \\ \sqrt{2}a_1a_2 \\ {a_2}^2 \end{bmatrix}^T \begin{bmatrix} {b_1}^2 \\ \sqrt{2}b_1b_2 \\ {b_2}^2 \end{bmatrix} = (a_1b_1)^2 + 2(a_1b_1)(a_2b_2) + (a_2b_2)^2 \\
&= (a_1b_1 + a_2b_2)^2 = \left[\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}^T \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right]^2 = (\mathbf{a}^T\mathbf{b})^2 = \left(\mathbf{x}^{(i)T}\mathbf{x}^{(j)}\right)^2
\end{aligned}$$

Common kernels used in SVM

The function $K(\mathbf{a}, \mathbf{b}) = (\mathbf{a}^T\mathbf{b})^2$ is called a 2nd-degree polynomial kernel. In Machine Learning, a kernel is a function capable of computing the dot product $\phi(\mathbf{a})^T\phi(\mathbf{b})$ based only on the original vectors $\mathbf{a}$ and $\mathbf{b}$, without having to compute (or even to know about) the transformation ϕ.

Linear:	$K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b}$
Polynomial of degree d:	$K(\mathbf{a}, \mathbf{b}) = (\gamma\,\mathbf{a}^T\mathbf{b} + r)^d$
Gaussian RBF:	$K(\mathbf{a}, \mathbf{b}) = \exp(-\gamma\,\lVert\mathbf{a} - \mathbf{b}\rVert^2 + r)$
Sigmoid:	$K(\mathbf{a}, \mathbf{b}) = \tanh(\gamma\,\mathbf{a}^\top\mathbf{b} + r)$

where γ, d, and r are hyperparameters.

(The hyperparameter r is the `coef0` parameter in the `SVC` and `SVR` classes of Scikit-learn)

Predictions with a kernelized SVM

With kernel trick, we plug the formula for $\hat{\mathbf{w}}$ into the decision function for a new instance $\mathbf{x}^{(n)}$:

$$h_{\hat{\mathbf{w}},\hat{b}}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) = \hat{\mathbf{w}}^T\phi\left(\mathbf{x}^{(n)}\right) + \hat{b} = \left(\sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\phi\left(\mathbf{x}^{(i)}\right)\right)^T\phi\left(\mathbf{x}^{(n)}\right) + \hat{b}$$

$$= \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\left(\phi\left(\mathbf{x}^{(i)}\right)^T\phi\left(\mathbf{x}^{(n)}\right)\right) + \hat{b}$$

$$= \sum_{\substack{i=1\\ \hat{\alpha}^{(i)}>0}}^{m}\hat{\alpha}^{(i)}t^{(i)}K\left(\mathbf{x}^{(i)},\mathbf{x}^{(n)}\right) + \hat{b}$$

And compute the bias term $\hat{b}$ using the same trick.

Summary for Chapter 5

- Linear SVM Classification: hard margin vs soft margin
- Nonlinear SVM Classification: polynomial kernels, similarity features, Gaussian RBF, Complexity
- SVM Regression
- Under the Hood: decision function, training objectives, the kernel trick

For next week

Work through Assignment 1 and attend the supervised lab. The Unit Coordinator or a casual Teaching Assistant will be there to help.

Mid-semester test is coming and will cover Chapters 1 to 6.

Read Chapter 6 on Decision Trees.

Have a good week. :)