Methods 3: Multilevel Statistical Modeling and Machine Learning

Week 7: *Logistic regression (machine learning)*

November 16, 2021

by: Lau Møller Andersen

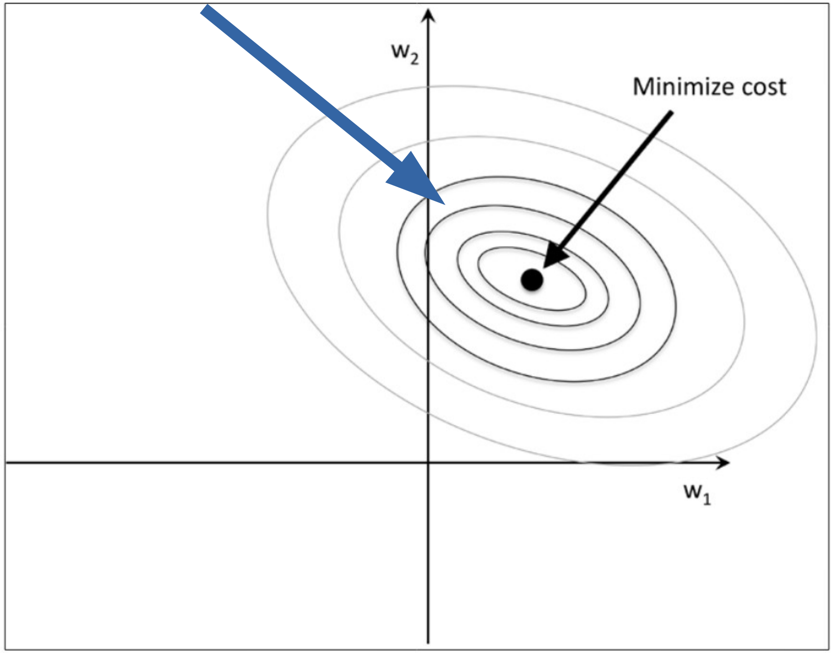
Remember to vote today!

Regularization - recap

$$J(w) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Solution space is infinite;
$\boldsymbol{w}$ can be any set of values

Solution space



(p. 113: Raschka, 2015)

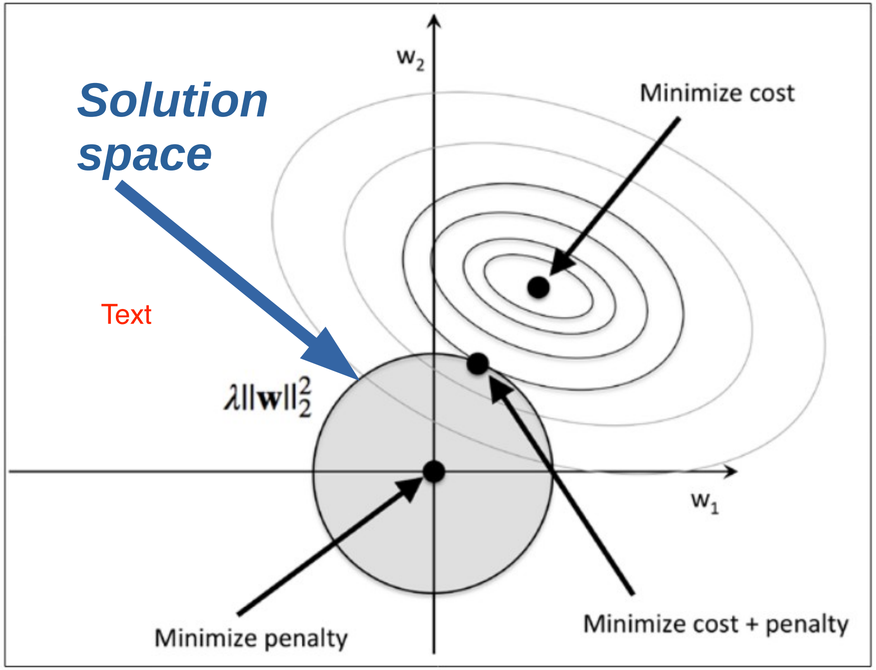
L2 regularization

Why is the *solution space* round?

Compare with a circle centred at (0,0)

$$w_1^2 + w_2^2 = r^2$$

$$L_2 \, \text{norm}: \|w\|_2 = \sqrt{(w_1^2 + w_2^2)}$$



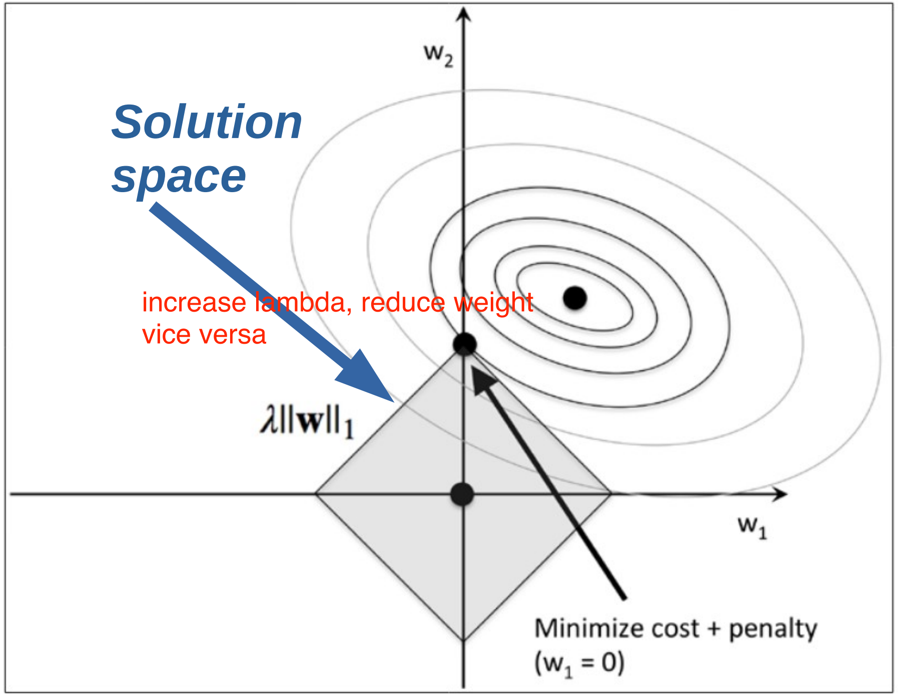
(p. 114: Raschka, 2015)

$$J(w)_{Ridge} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \lambda \| w \|_2^2$$

L1 regularization

Why is the *solution space* square?

$$\mathrm{L}_1\,\mathrm{norm}:\|w\|_1=|w_1|+|w_2|$$
$$\text{if } w_1=max(w_1) \text{ then } w_2=0$$
$$\text{if } w_2=max(w_2) \text{ then } w_1=0$$



(p. 115: Raschka, 2015)

$$J(w)_{LASSO}=\sum_{i=1}^{n}\left(y^{(i)}-\hat{y}^{(i)}\right)^2+\lambda\|w\|_1$$

Learning goals

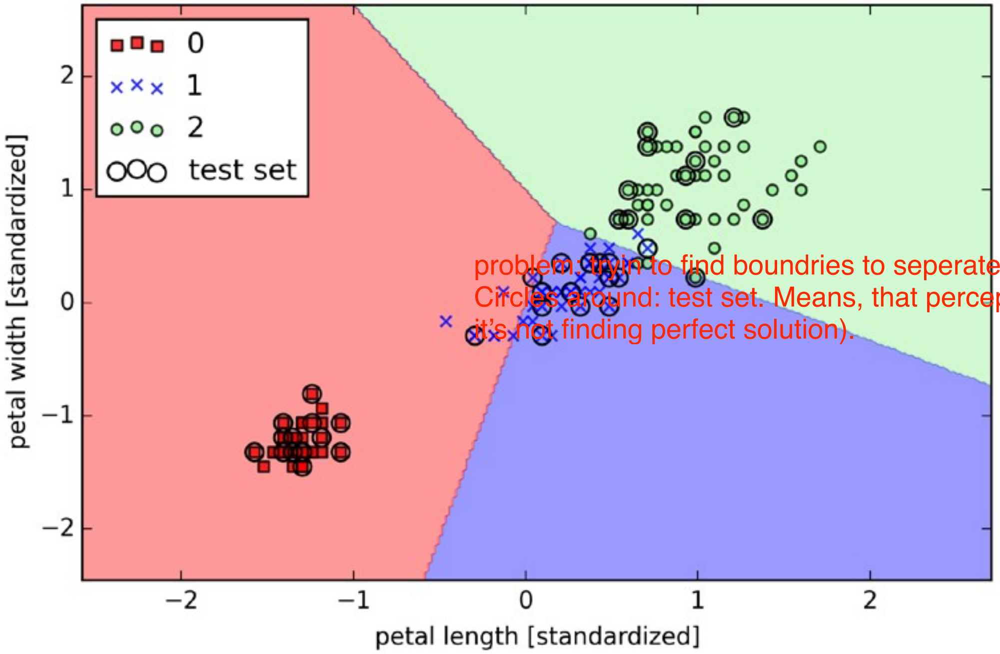
Logistic regression (machine learning)

1) Understanding of how logistic regression can be adapted to a classification framework

2) Understanding the idea of a Support Vector Machine

3) Getting acquainted with how Support Vector Machines can solve non-linear problems

The problem (Perceptron)



problem: tryin to find boundries to seperate flowers (iris df).
Circles around: test set. Means, that perceptron have to converge (becuase it's not finding perfect solution).

(p. 55: Raschka, 2015)

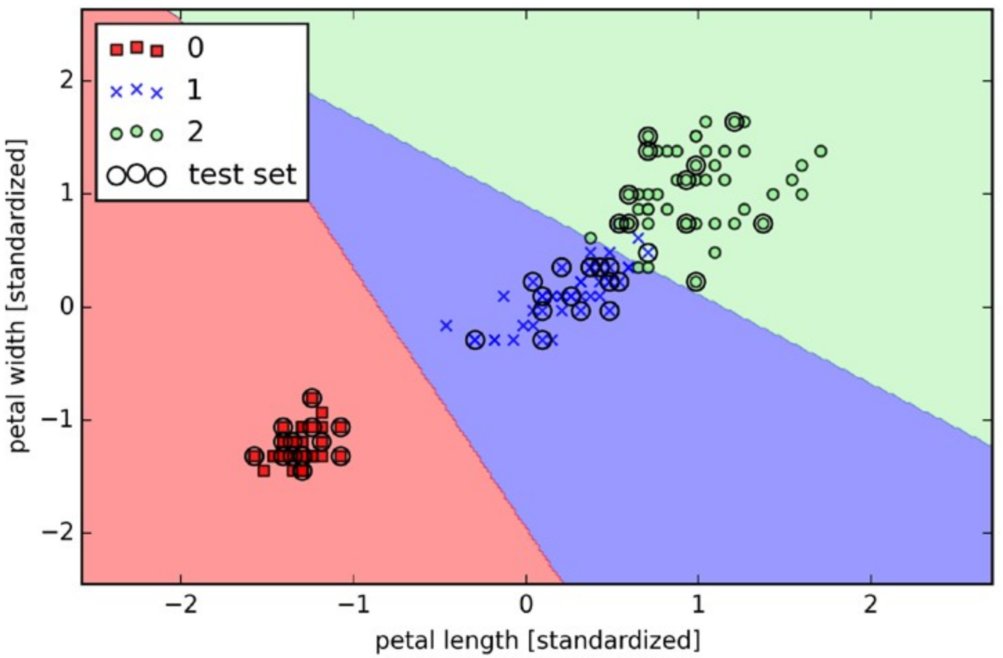
Not linearly separable ⟶ never converges

WANTED: an algorithm that converges with linearly separable regions that minimise the number of errors made

Text

Something like this

LOGISTIC REGRESSION



(p. 63: Raschka, 2015)

Separates flowers, while keeping errors at a minimum

Odds ratio

$$\text{odds ratio} = \frac{p}{1-p}$$

$$\text{log odds} = \log\left(\frac{p}{1-p}\right) = logit(p)$$

$$logit(p(y=1|\boldsymbol{x})) = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}$$

probability of observing 1 given the features (in this case, different features of the flower)

Example: an odds ratio of 7 to 1 (7:1) means that it is 7 times more likely that something is going to happen than that it is not going to happen

Text

probability that y = 1 given x.

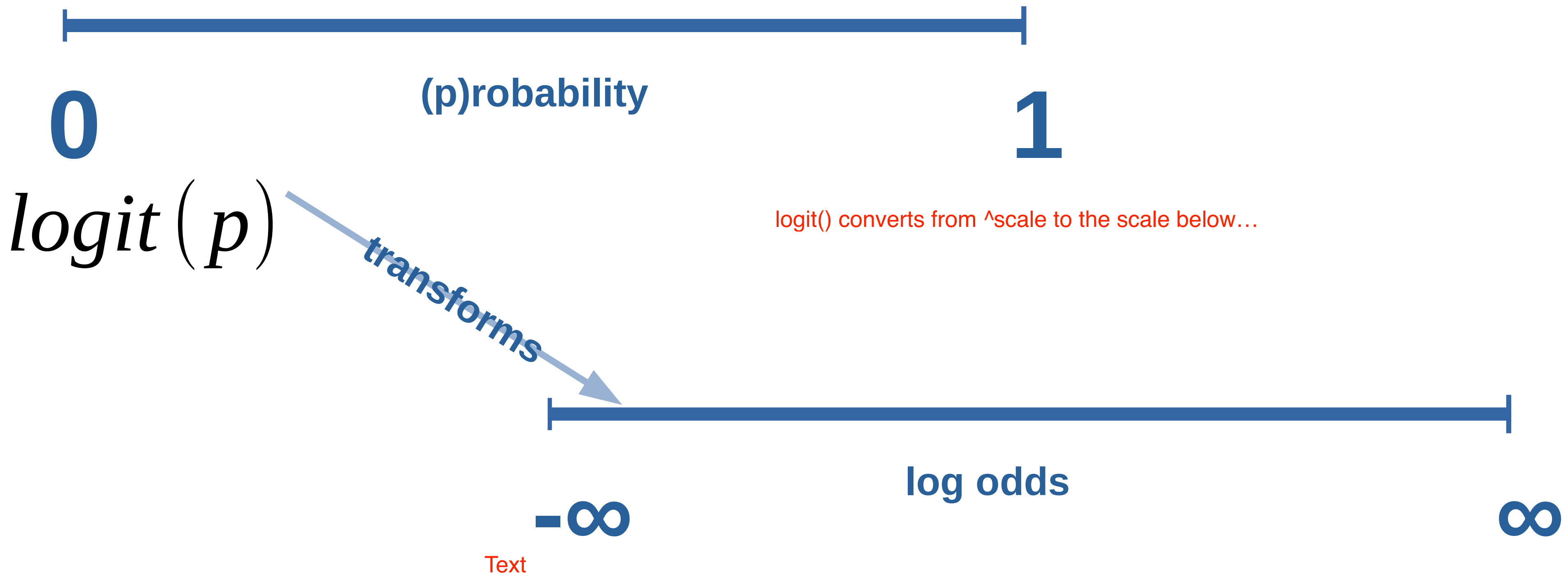
$$logit\left(p\left(y=1|\boldsymbol{x}\right)\right)=w_0 x_0+w_1 x_1+\ldots+w_m x_m=\sum_{i=0}^{m} w_i x_i=\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}$$

$$p\left(y=1|\boldsymbol{x}\right):\text{is the probability that } y=1,\text{ given } \boldsymbol{x}$$

(p)robability

0 **1**

Text


0
(p)robability
1
$logit(p)$
transforms
logit() converts from ^scale to the scale below…
log odds
-∞
∞
Text

log odds

-∞ **∞**

$$\phi(z) = p$$

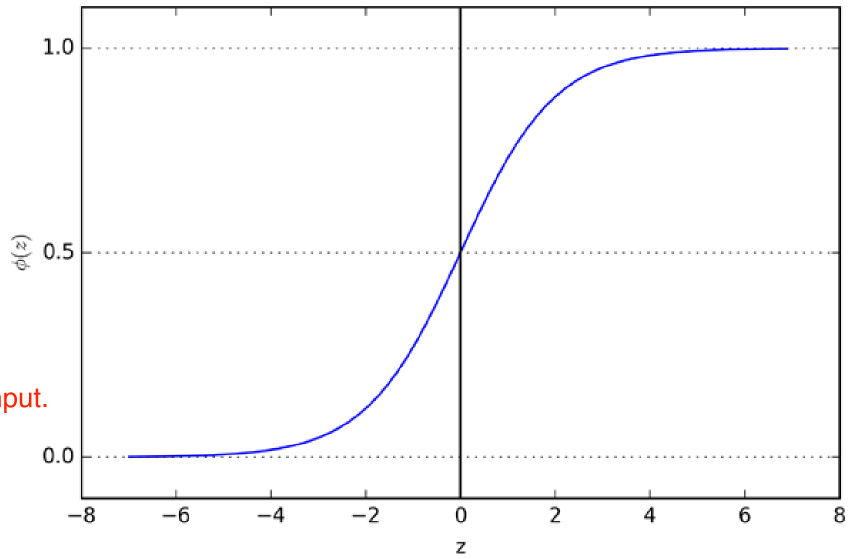
inverse function

$$logit^{-1}(z) = \phi(z) = \frac{1}{1+e^{-z}}$$

(sigmoid-shaped) Text

net input: take all weights that we've estimated, and multiply with features = net input.

$$\text{net input: } z = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m$$



(p. 58: Raschka, 2015)

what makes this a classification algorithm is this:

Quantizer

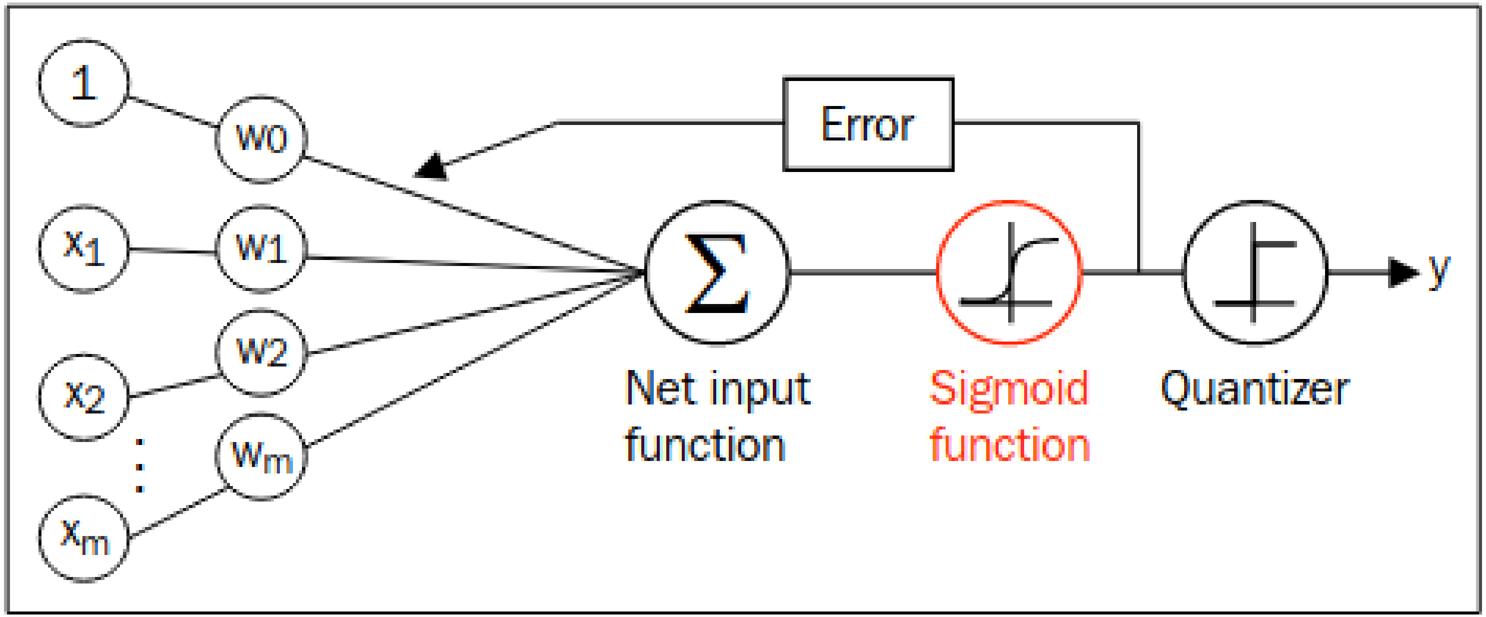
$$\hat{y} = \begin{cases} 1 & \text{if } \phi(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Text

$$\hat{y} = \begin{cases} 1 & \text{if } z \geq 0.0 \\ 0 & \text{otherwise} \end{cases}$$

Text

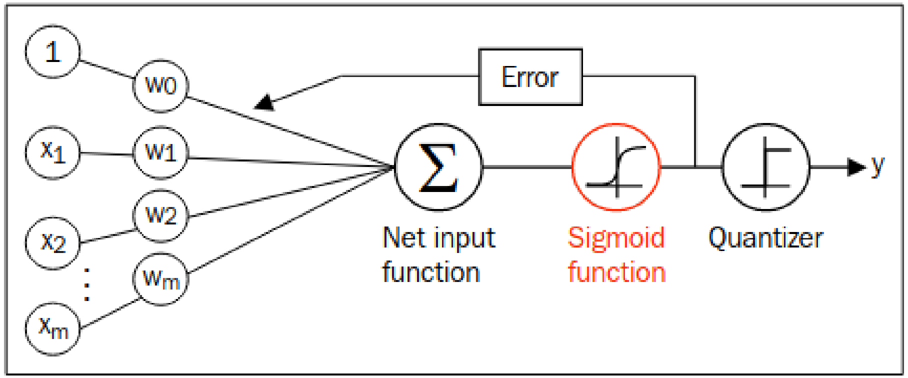
Logistic regression



we apply the sigmoid to the input funciton in this case.

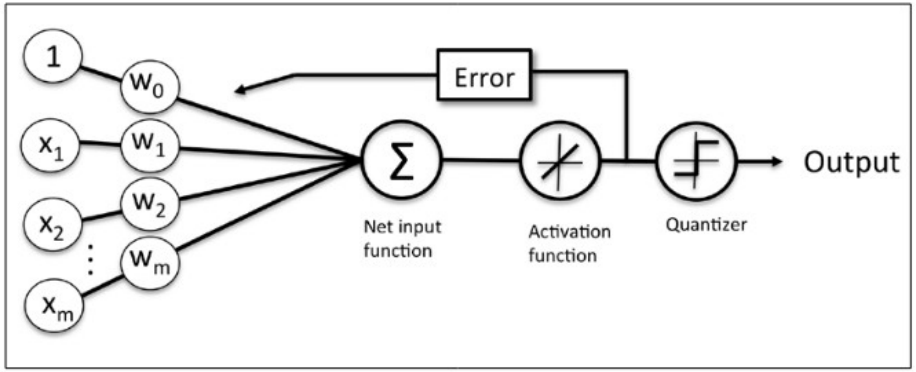
(p. 58: Raschka, 2015)

Comparison with ADALINE



(p. 58: Raschka, 2015)

$$\phi(z)=\frac{1}{1+e^{-z}}$$



(p. 33: Raschka, 2015)

gradient descent: when you update function, based on input.

$$\phi(z)=z$$

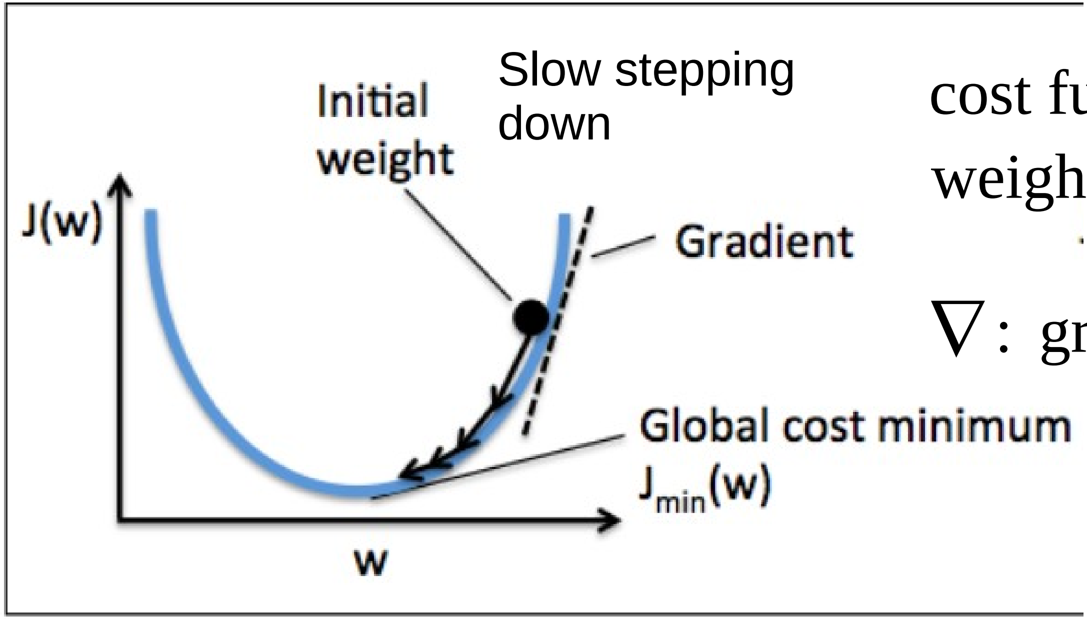
we run the log(odds) through the quantizer (not on the probability scale).

Text

Still updating with **Gradient Descent**

Gradient descent $\phi(z)=z$

difference between adaline and linReg, is that in the adaline we apply quantizer in the end.



cost function: $J(\mathbf{w})=\left(\sum(y-\hat{y})^2\right)/2$

weight change: $\Delta\mathbf{w}=-\eta\nabla J(\mathbf{w})$

∇: gradient (rate of change)

when gradient is high = go fast, when low, go smaller steps (to avoid overshooting).

(p. 40: Raschka, 2015)

Text

Gradient descent $\phi(z) = \frac{1}{1+e^{-z}}$

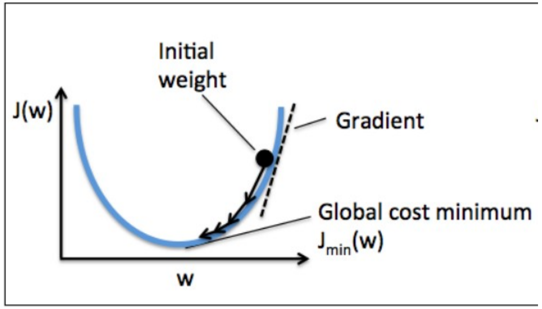
cost function: $J(\boldsymbol{w}) = -\boxed{\sum_{i}^{n} y^{(i)} \log\left(\phi\left(z^{i}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)}$

weight change: $\Delta \boldsymbol{w} = -\eta \nabla J(\boldsymbol{w})$

Text

What's this part equal to?

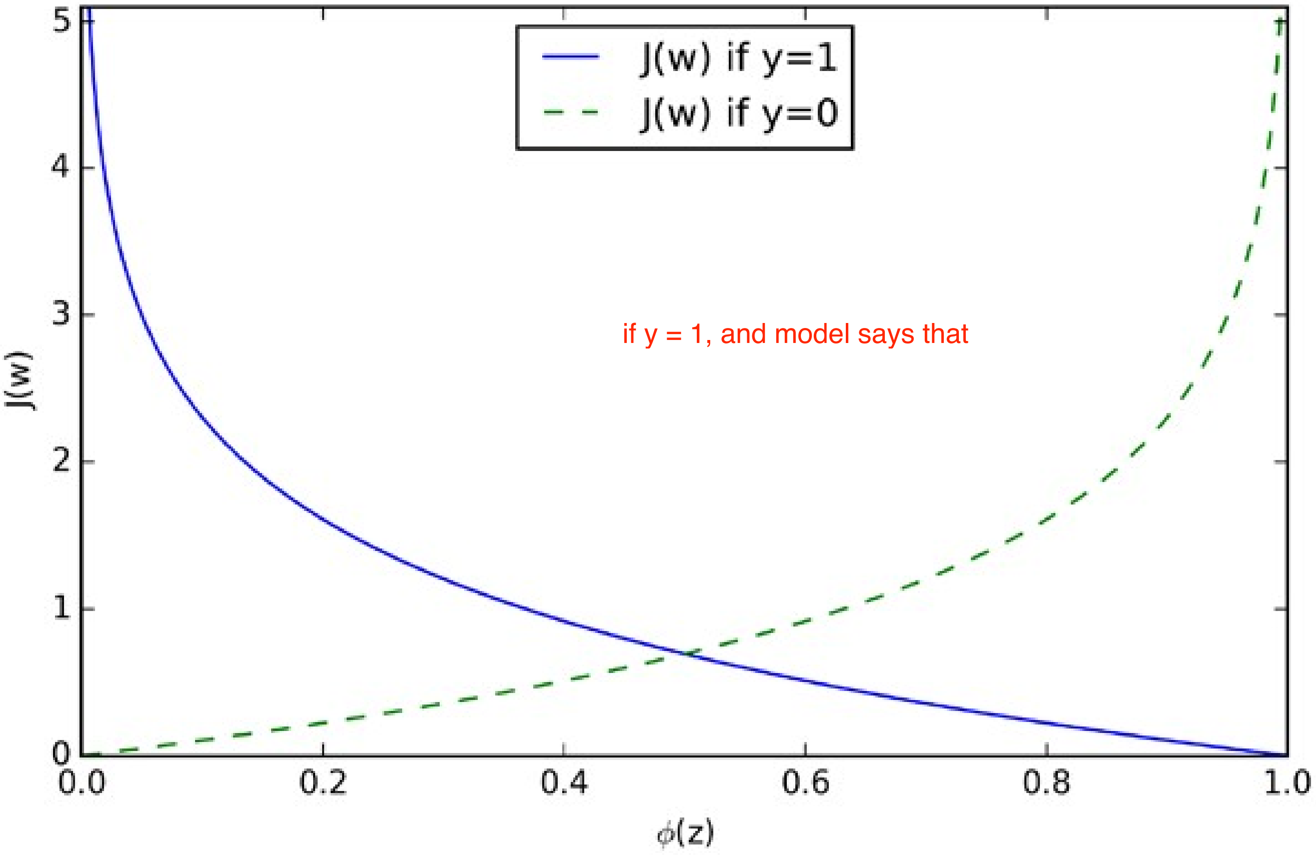
log likelihood:



(p. 40: Raschka, 2015)

Text

A closer look at the cost function



if y = 1, and model says that

(p. 61: Raschka, 2015)

A closer look at the updating of weights

$\Delta \boldsymbol{w} = -\eta \nabla J(\boldsymbol{w})$ This is common between ADALINE and Linear and Logistic Regression

but $J(\boldsymbol{w})$ is not the same across these different methods?!

A general formulation: $\Delta w_j = \eta \sum_i^n \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) x_j^{(i)}$

It can be shown that: $\frac{\delta}{\delta w_j} l(\boldsymbol{w}) = (y - \phi(z)) x_j$

$\frac{\delta}{\delta w_j} l(\boldsymbol{w})$: (partial) derivative of log-likelihood function (proof on p. 64)

A clarification on ADALINE

$$\Delta w_j = \eta \sum_{i}^{n} \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) x_j^{(i)}$$

$$\phi(z) = z = \hat{y}$$

```python
def fit(self, X, y):
    """ Fit training data.

    Parameters
    ----------
    X : {array-like}, shape = [n_samples, n_features]
        Traing vectors, where n_samples
        is the number of samples and
        n_features is the number of features.
    y : array-like, shape = [n_samples]
        Target values.

    Returns
    -------
    self : object

    """
    self.w_ = np.zeros(1 + X.shape[1])
    self.cost_ = []

    for i in range(self.n_iter):
        output = self.net_input(X)
        errors = (y - output)
        self.w_[1:] += self.eta * X.T.dot(errors)
        self.w_[0] += self.eta * errors.sum()
        cost = (errors**2).sum() / 2.0
        self.cost_.append(cost)
    return self
```

With regularisation

$$J(\boldsymbol{w}) = -\sum_{i}^{n} y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right) + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2$$

NB! Regularisation in *sklearn.linear_model.LogisticRegression* is controlled by *C*

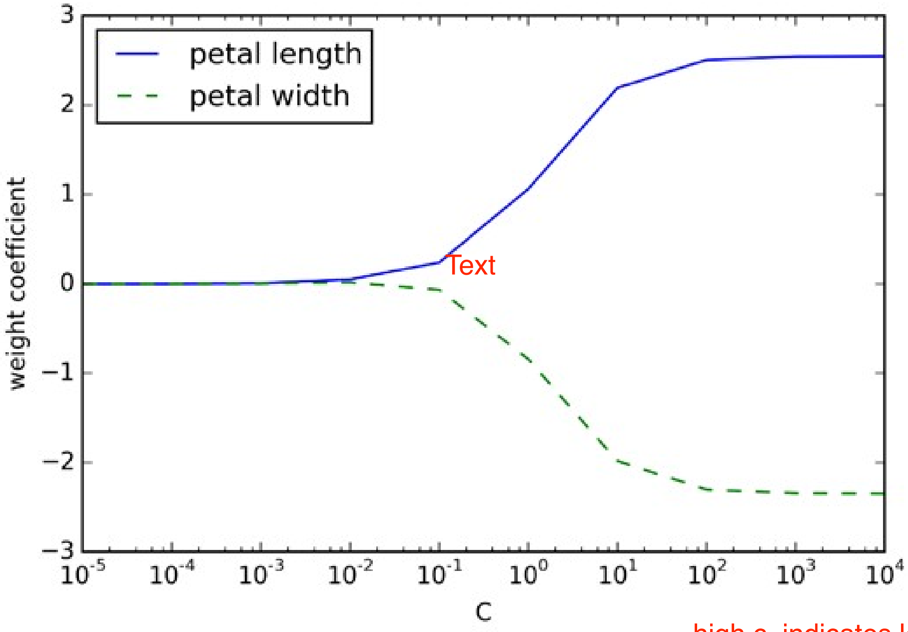
high c, indicates low lambda.
low c, indicates a high lambda

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

C = inverse

Text

Effect of C parameter



high c, indicates low lambda.
low c, indicates a high lambda

Now for the **classification**

score(*X*, *y*, *sample_weight=None*) [source]

Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters:	**X** : ***array-like of shape (n_samples, n_features)*** Test samples. **y** : ***array-like of shape (n_samples,) or (n_samples, n_outputs)*** True labels for `X`. **sample_weight** : ***array-like of shape (n_samples,), default=None*** Sample weights.
Returns:	**score** : ***float*** Mean accuracy of `self.predict(X)` wrt. `y`.

```
#%% LOGISTIC REGRESSION

from sklearn.linear_model import LogisticRegression
logR = LogisticRegression(penalty='none') # no regularisation
logR.fit(X_train_std, y_train)
print(logR.score(X_test_std, y_test))
```

Live coding

EXAMPLE_00

type(df) = df command??

df.feature_names = get names of variables in df

import train_test_split = library to split data into x partitions.

df.shape = returns number of values in variable

import StandardScaler
data_std = standradizes data

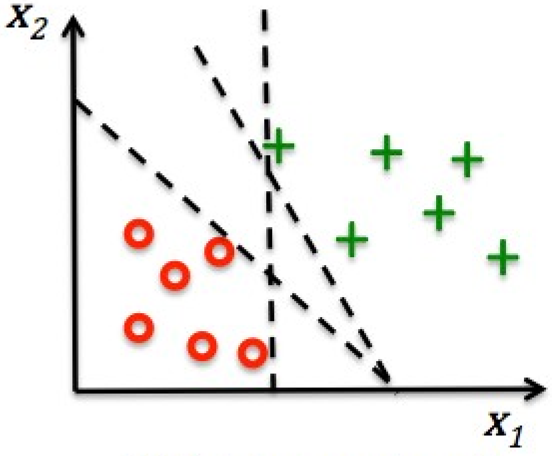
ctrl + i = brings up help menu

stratfied kfold: tries to get an equal number of targets in each folds.

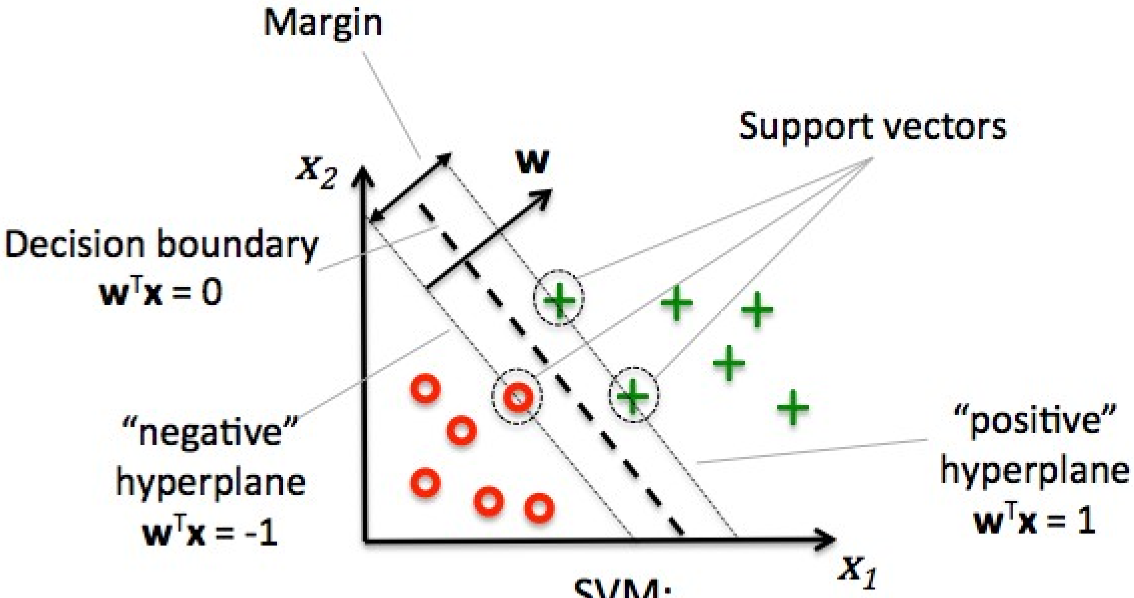
SUPPORT VECTOR MACHINES

find a hyperplane, that maximizes margin but still
doesn't misclassify any of the data

two features: length of width
of flowers



Which hyperplane?

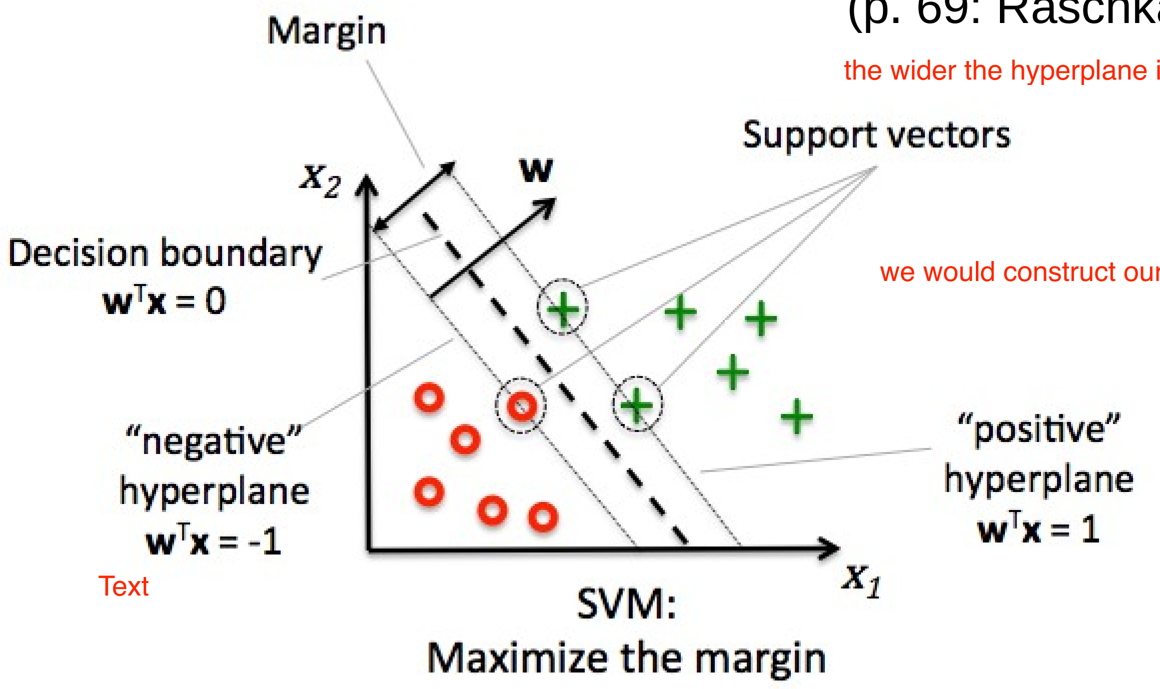


SVM:
Maximize the margin

(p. 69: Raschka, 2015)

(p. 69: Raschka, 2015)

the wider the hyperplane is, the less likeli we are to missclassify data.



we would construct our model according to how the hyperplane fits the data

Text

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{\mathbf{pos}} = 1 \qquad (1)$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{\mathbf{neg}} = -1 \qquad (2)$$

QUESTION: What is the subtraction of equation (2) from equation (1) equal to on the figure?

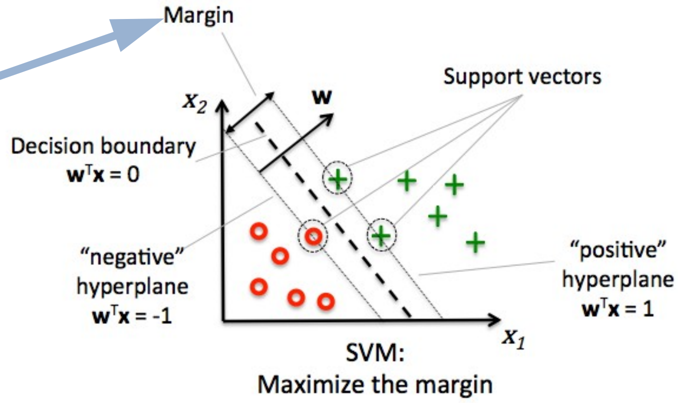
Subtraction

$$\boldsymbol{w}^T(\boldsymbol{x}_{\mathbf{pos}} - \boldsymbol{x}_{\mathbf{neg}}) = 2$$

normalizing by w's length: $\|\boldsymbol{w}\| = \sqrt{(\sum_{j=1}^{m} w_j^2)}$

... we get: $\boldsymbol{w}^T \frac{(\boldsymbol{x}_{\mathbf{pos}} - \boldsymbol{x}_{\mathbf{neg}})}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$

This will optimise the width of the margin



(p. 69: Raschka, 2015)

So we want to maximize: $\dfrac{2}{\|\boldsymbol{w}\|}$

under the constraints:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1$$
$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \; y^{(i)} = -1$$

In practice, the following is minimized: $\dfrac{1}{2}\|\boldsymbol{w}\|^2$

This faces a problem that the **Perceptron** also faces

Which?

there might be no hyperplane, that perfectly seperates the points, and thus the model won’t converge (similar to perceptrons (see ealier slides)).

SOLUTION: introducing a *slack* variable ξ (xi)

slack variable:

Introducing ξ

We add slack to the constraints so we are not as strict i fitting the hyperplane and thus not risking our model to not converge.

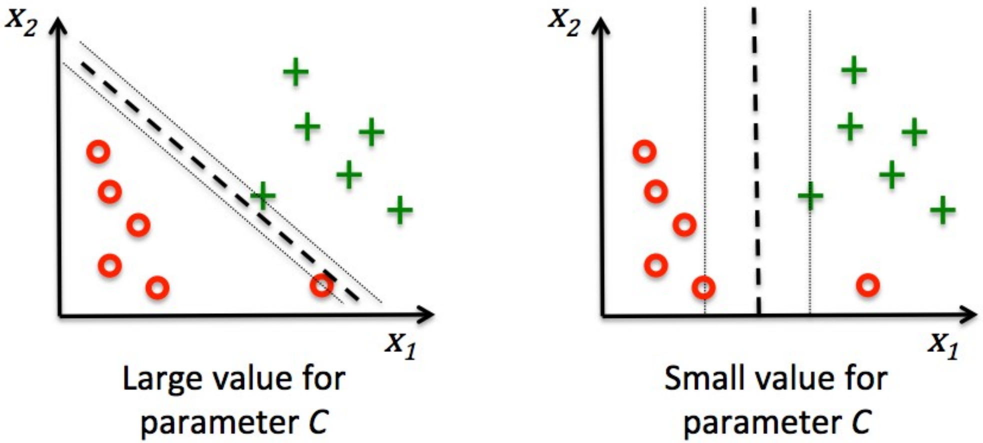
$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{\mathbf{pos}} = 1 - \xi^{(i)}$$
$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{\mathbf{neg}} = -1 + \xi^{(i)}$$

the large C = narrow margin (penalize errors harshly), or small c = large margin. So C and hyperplane are connected, and slack (xi) defines the slackyness of these constraints.

now we need to minimize: $\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i^n \xi^{(i)}\right)$ (p. 72: Raschka, 2015)

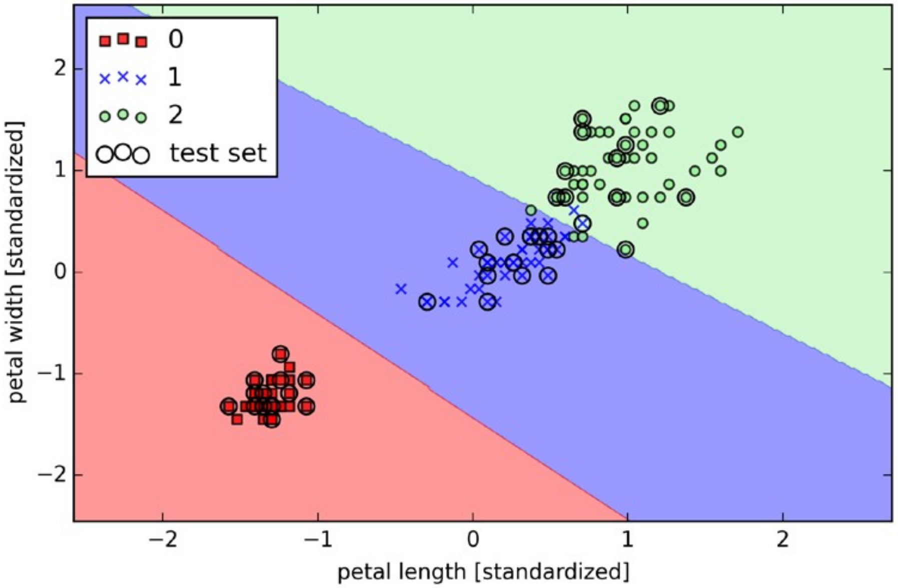
to maximise the margin

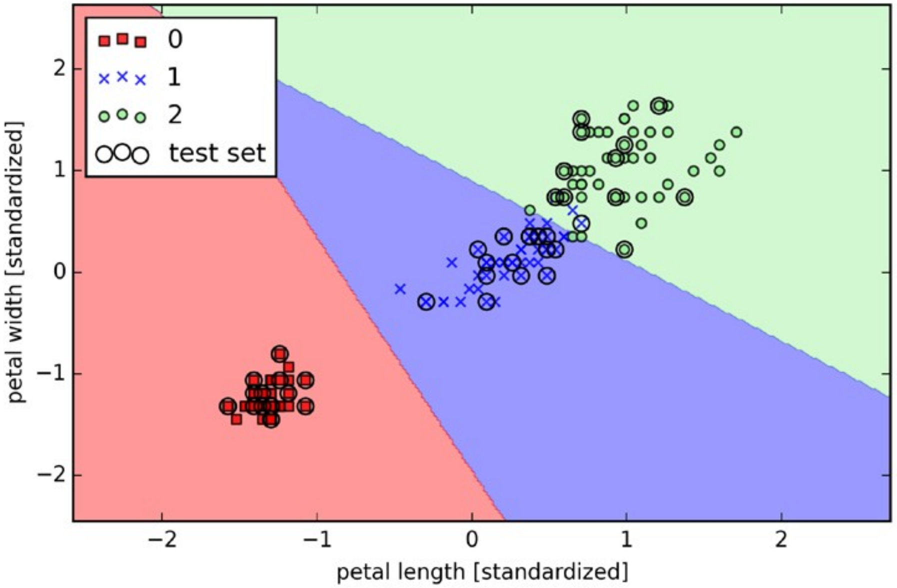
Large *C*: narrow margin – penalises errors harshly
Small *C*: wide margin – penalises errors mildly



Comparison to logistic regression

Text





Support Vector Machine (p.763, Raschka 2015)

Logistic regression (p. 63, Raschka 2015)

Logistic regression versus SVM

In practical classification tasks, linear logistic regression and linear SVMs often yield very similar results. Logistic regression tries to maximize the conditional likelihoods of the training data, which makes it more prone to outliers than SVMs. The SVMs mostly care about the points that are closest to the decision boundary (support vectors). On the other hand, logistic regression has the advantage that it is a simpler model that can be implemented more easily. Furthermore, logistic regression models can be easily updated, which is attractive when working with streaming data.

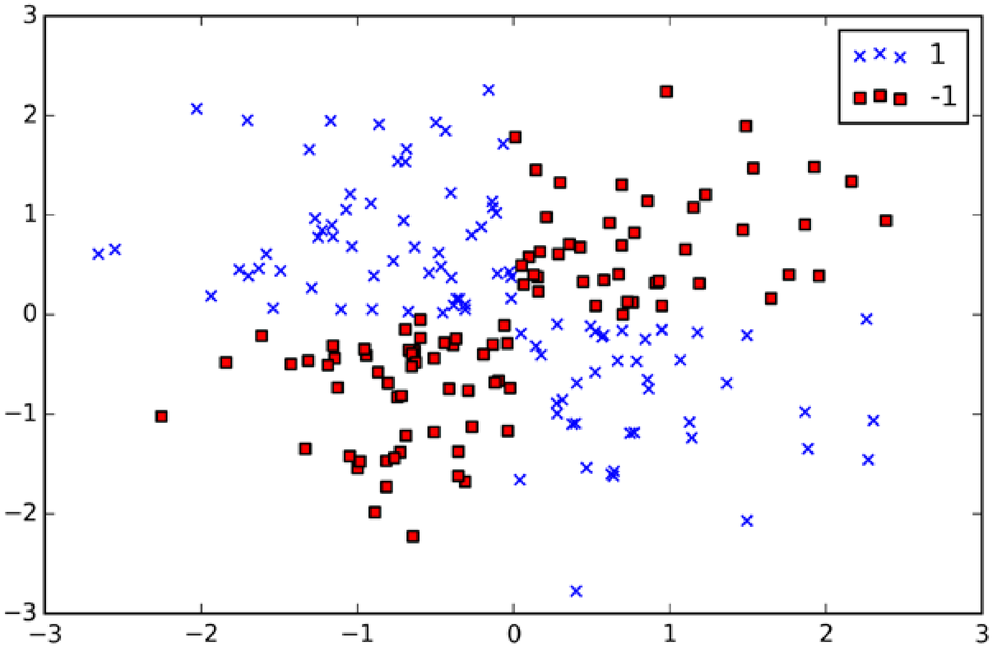
(p. 74: Raschka, 2015)

Live coding

EXAMPLE_00

A brief look at a non-linear problem

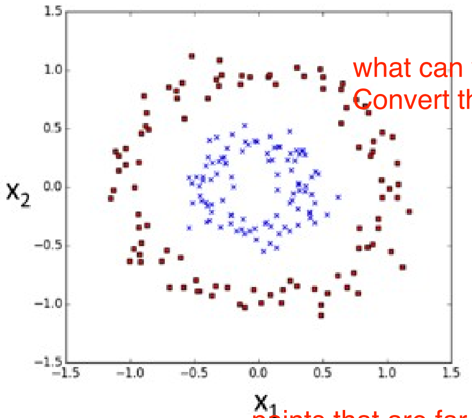
non-linear problems:



(p. 75: Raschka, 2015)

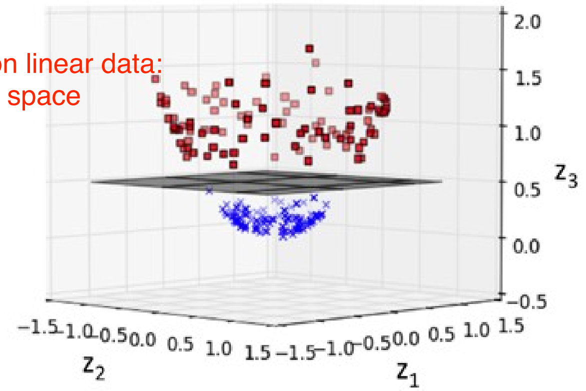
Map onto higher-dimensional space

$$\phi(\cdot)$$



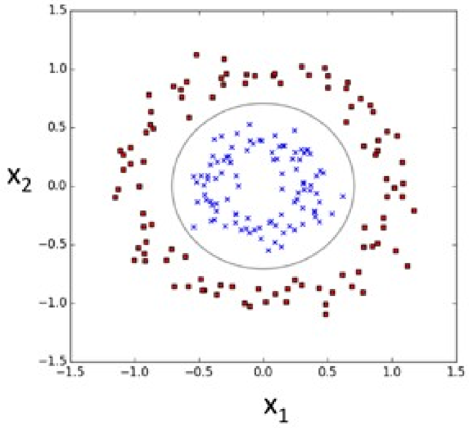
what can we do about non linear data:
Convert the 2D into a 3D space

ϕ



points that are far from the mean, are projected into a higher dimension, than the ones that are closer to the mean.

ϕ^{-1}



Text

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

Note that a linear solution is made

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, \boxed{x_1^2 + x_2^2})$$

Creating the higher dimensions can be computationally expensive

gamma = hyperparameter

calculates the simularity between two features

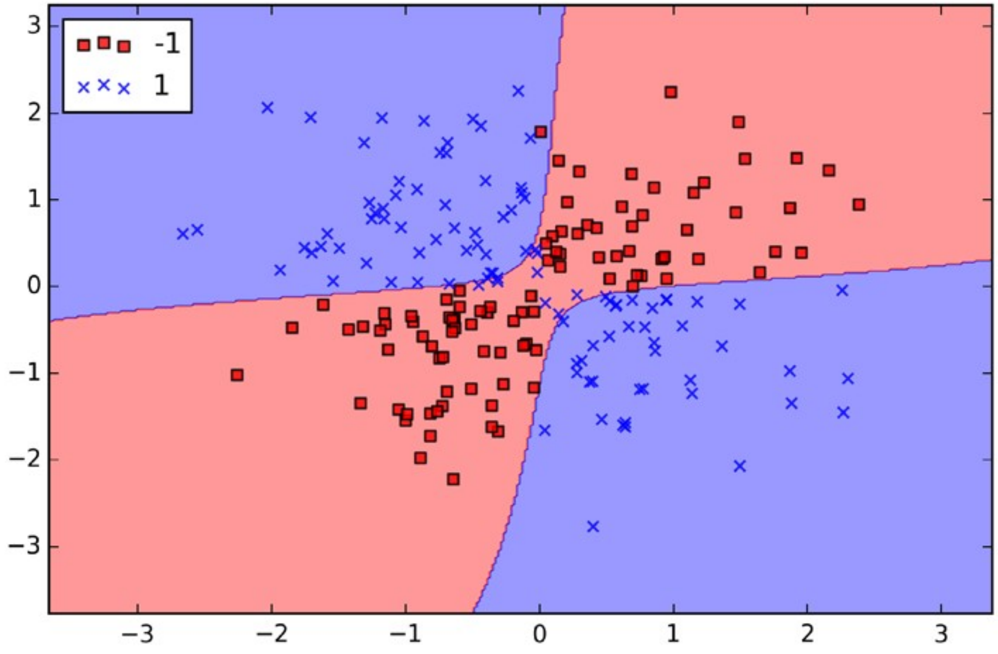
$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = e^{\left(-\gamma \left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|^2\right)}$$

$$\left(\gamma = \frac{1}{2\sigma^2}, \text{ also called the precision}\right)$$

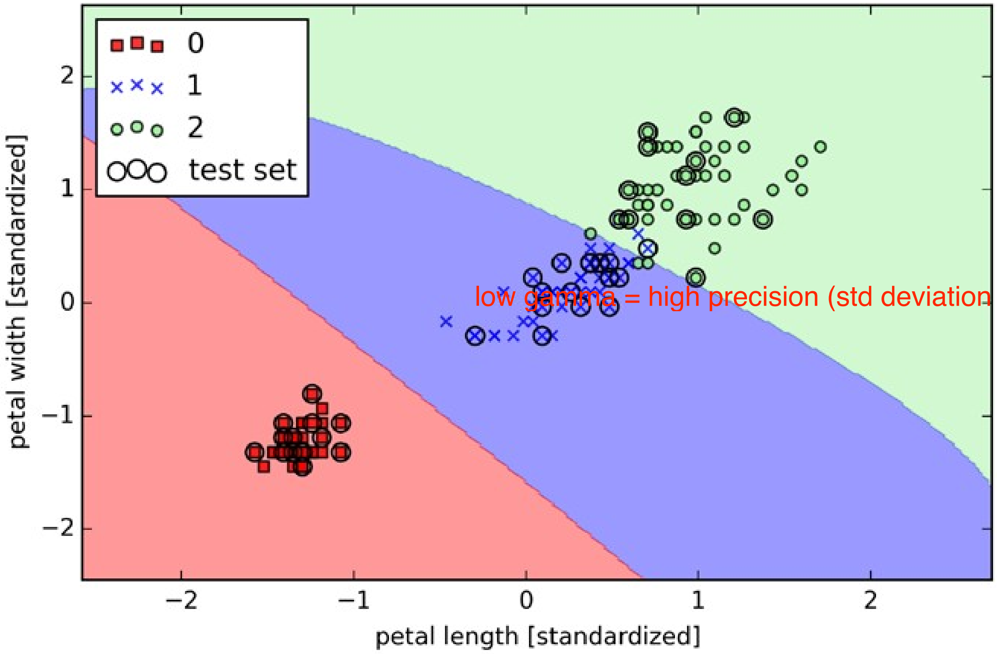
the o

Non-linear decision boundaries



(p. 78: Raschka, 2015)

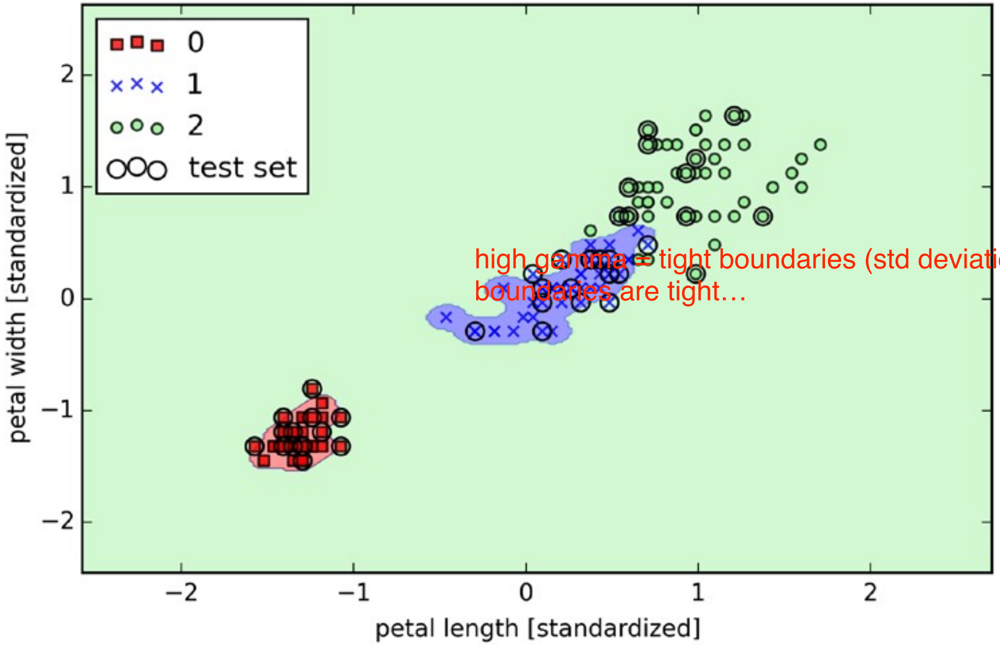
Low γ - soft boundary



low gamma = high precision (std deviation is big), and you get curving boundaries)

(p. 79: Raschka, 2015)

High γ - tight boundary



high gamma = tight boundaries (std deviation is small), and as can be seen boundaries are tight…

(p. 80: Raschka, 2015)

Live coding

EXAMPLE_00

Did you learn?

Logistic regression (machine learning)

1) Understanding of how logistic regression can be adapted to a classification framework

2) Understanding the idea of a Support Vector Machine

3) Getting acquainted with how Support Vector Machines can solve non-linear problems

References

- Raschka, S., 2015. Python Machine Learning. Packt Publishing Ltd.