Methods 3: Multilevel Statistical Modeling and Machine Learning

Week 07: Linear and logistic regression revisited (machine learning)
October 22, 2024

The course plan

Week 1: *Introduction*

Instructor sessions: *Setting up* R *and* Python *and recollection of the general linear model*

Week 2: *Multilevel linear regression*

Instructor sessions: *Modelling subject level effects – and how do they differ from group level effects?*

Week 3: *Link functions and fitting generalised linear multilevel models*

Instructor sessions: *What to do when the response variable is not continuous?*

Week 4: *Evaluating Generalised linear mixed models*

Instructor sessions: *How do we assess how models compare to one another?*

Week 5: *Explanation and Prediction*

Instructor sessions: *Code review*

Week 6: *Mid-way evaluation and Machine Learning Intro*

Instructor sessions: *Getting Python Running*

Week 7: *Linear and logistic regression revisited (machine learning)*

Instructor sessions: *Categorizing responses based on informed guesses*

Week 8: *Model evaluation and hyperparameter tuning*

Instructor sessions: *Code review*

Week 9: *Dimensionality Reduction, Principled Component Analysis (PCA)*

Instructor sessions: *What to do with very rich data?*

Week 10: *Outlook, unsupervised classification and neural networks*

Instructor sessions: *Data with no labels and networks*

Week 11: *Organising and preprocessing messy data*

Instructor sessions: *Code review*

Week 12: *Final evaluation and wrap-up of course*

Instructor sessions: *Ask anything!*

Mid-way evaluation

TOTAL ANSWERS = 51

n≥3

LIKE	*count*
Lectures	17
Assignments	14
Lau is engaged	10
Code review	9
Good examples	9
Structure of course	9
Lau listens to feedback	7
Ambitions are high	7
Thorkild’s classes	6
Discussion with peers	5
Detailed feedback on assignment 1	4
That assignments are done in class	4
Recap	4
Amount of assignments	3
Amount of math	3
Learning goals	3

n<3

Relevant knowledge	2
Good readings	2
Good slides	2
Relevant data	2
Reading questions	2
Good pace	2
Lau makes us laugh	1
Doing individual assignments	1
Discussion of code	1
Tutorial paper (Winter)	1
Open for questions	2
Live coding	2
Lau encourages using AI tools	1
Machine learning	1
Machine learning book	1
Gelman book	1
Different lecture rooms	1

n≥3

DID NOT LIKE	*count*
Assignment interpretation	18
Fast paced	12
Too many assignments	7
Gelman book	5
Timing of handing in assignments	5
Setting up the conda environment	4
“New” questions in assignments	3
Assignments are hard	3
Methods 1 and 2 didn’t prepare us	3

n<3

Live coding	2
Doing individual assignments	2
Structure between readings, assignment and lectures	2
Too little math	2
Slides without the answers	1
Having to do two individual assignments	1
That answers are not provided for exam assignments	1
Ambitions are high	1
The load is high	1
X and Z matrices	1
Python	1
Class titles	1

n≥2

CHANGE	*count*
Clearer wording in assignments	7
Adding other material (e.g. videos)	5
More text on slides	3
More instructions in classes	3
Amount of assignments	2
Amount of individual assignments	2
Shorter assignments	2
Lecture recaps in class	3
Have exercises in lectures	2
Get answers on slides (don't get slides before lecture	2
Slow down	2
More feedback	2
Change methods 1 and 2	2
Fewer slides	2

n<2

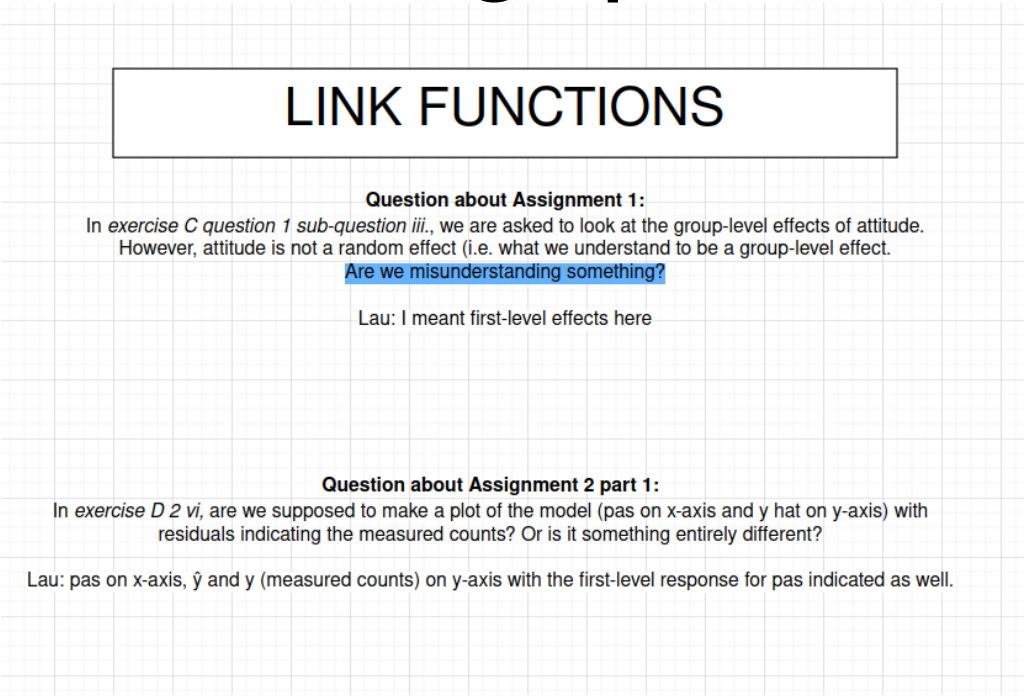
Filming classes	1
Code aid	1
Making course less compact	1
Solutions following assignments	1
More code reviews	1
More time for questions in the lecture	1
Change assignment deadline	1
Design experiment, gather data and analyse	1
Discard the Gelman book	1
Have even more discussions	1
Vote at the end on what topics were the most hard	1
Mid-way evaluation method	1
Prepare conda environment better	1
Clearer link between assignment and readings	1
Pen and paper exercises	1
Frame why topic is useful	1
Make real-life applicable	1
More reading questions	1
Lecture on doing good plots	1
More coding in class	1

Summary – what we promise to change

I will be very careful and aim at writing questions that are easy to understand

Thorkild will add some good videos

What you should change – start asking questions!



LINK FUNCTIONS

Question about Assignment 1:

In *exercise C question 1 sub-question iii.*, we are asked to look at the group-level effects of attitude. However, attitude is not a random effect (i.e. what we understand to be a group-level effect.
Are we misunderstanding something?

Lau: I meant first-level effects here

Question about Assignment 2 part 1:

In *exercise D 2 vi,* are we supposed to make a plot of the model (pas on x-axis and y hat on y-axis) with residuals indicating the measured counts? Or is it something entirely different?

Lau: pas on x-axis, $\hat{y}$ and y (measured counts) on y-axis with the first-level response for pas indicated as well.

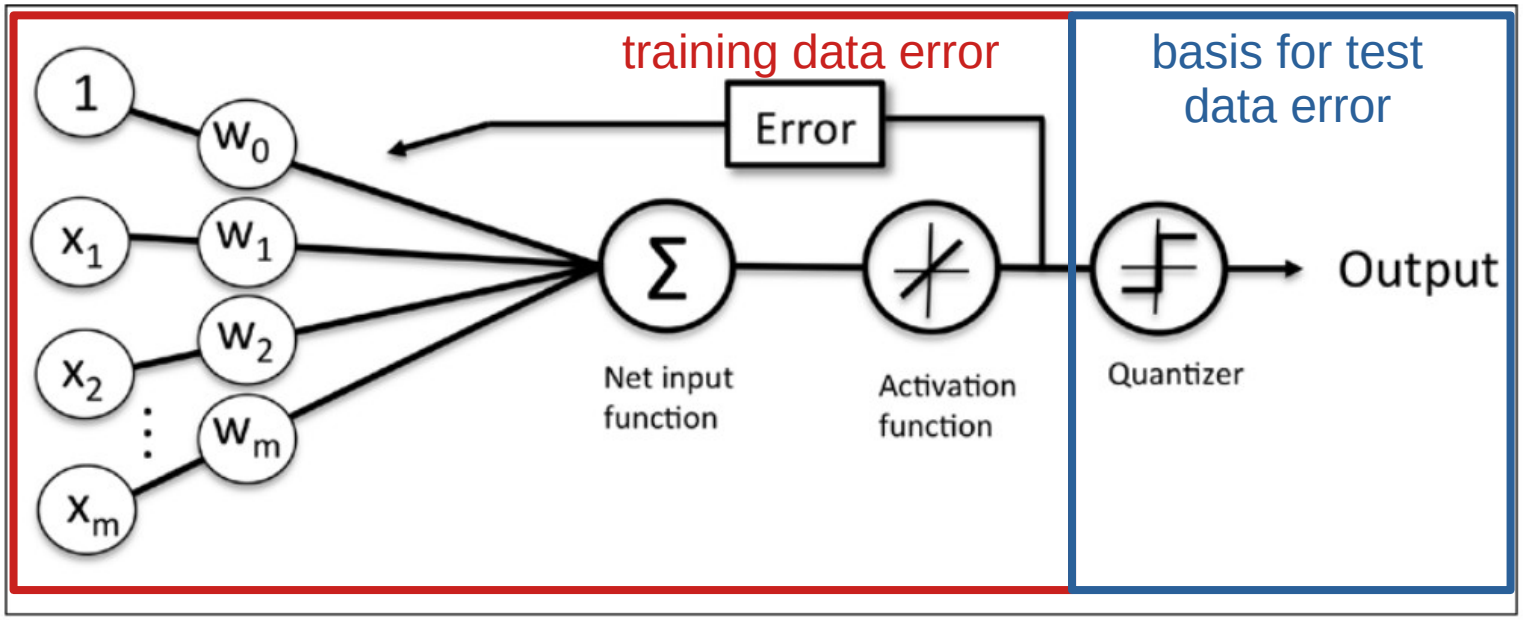
Next year – conda installation

- I will be able to borrow a Windows computer and a Mac computer to test the setup

Recap

- We have *feature* vectors, $x^{(i)}$, for which we try to find the optimal weights, *w*.
  - Similar to $X\beta$
- The optimal *w*'s for the are the ones that minimise the error
  - Two kinds of error:
    - Perceptron
      - training data error: *n* misclassifications on training data
      - test data error: *n* misclassifications when generalised to test data
    - ADALINE
      - training data error: sum of squared errors on training data
      - test data error: *n* misclassifications when generalised to test data
    - Linear regression
      - training data error: sum of squared errors on training data
      - test data error: mean of squared errors when generalised to test data
- Our primary outcome is the test data error

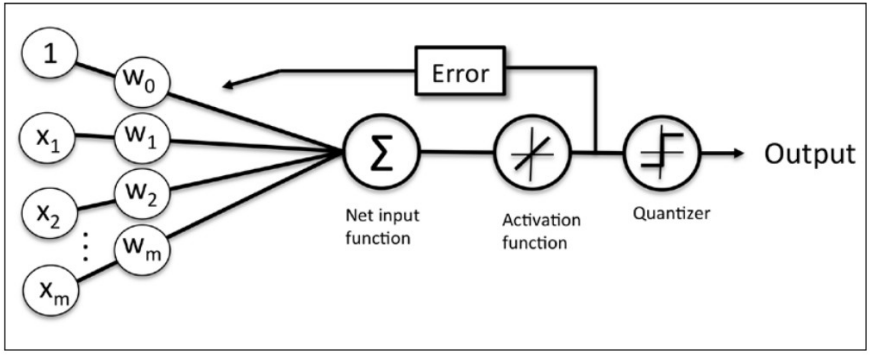
Recap



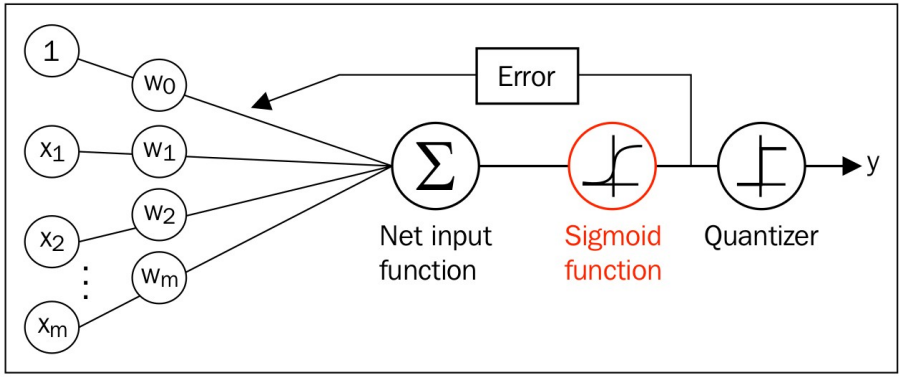
Features ***Weights***

adapted from (p. 33: Raschka, 2015)

Moving to logistic regression



(p. 33: Raschka, 2015)



(p. 58: Raschka, 2015)

Learning goals and outline

Linear and logistic regression revisited (machine learning)

1) Understanding how linear and logistic regression can be adapted to a classification framework
2) Understanding how gradient descent can provide a common framework across regression fits
3) Getting acquainted with Support Vector Machines

WHY LOGISTIC REGRESSION?

Let’s have a look at an iris flower

se·pal (sē′pəl)

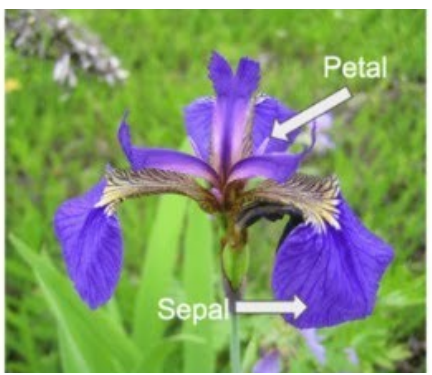
n.

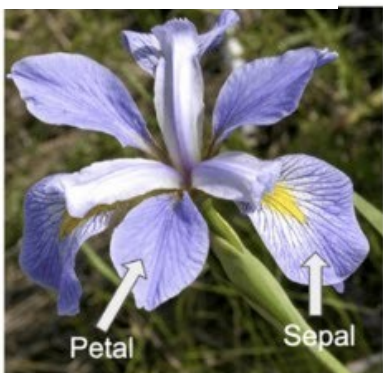
One of the usually green leaflike structures composing the outermost part of a flower. Sepals often enclose and protect the bud and may remain after the fruit forms.

pet·al (pĕt′l)

n.

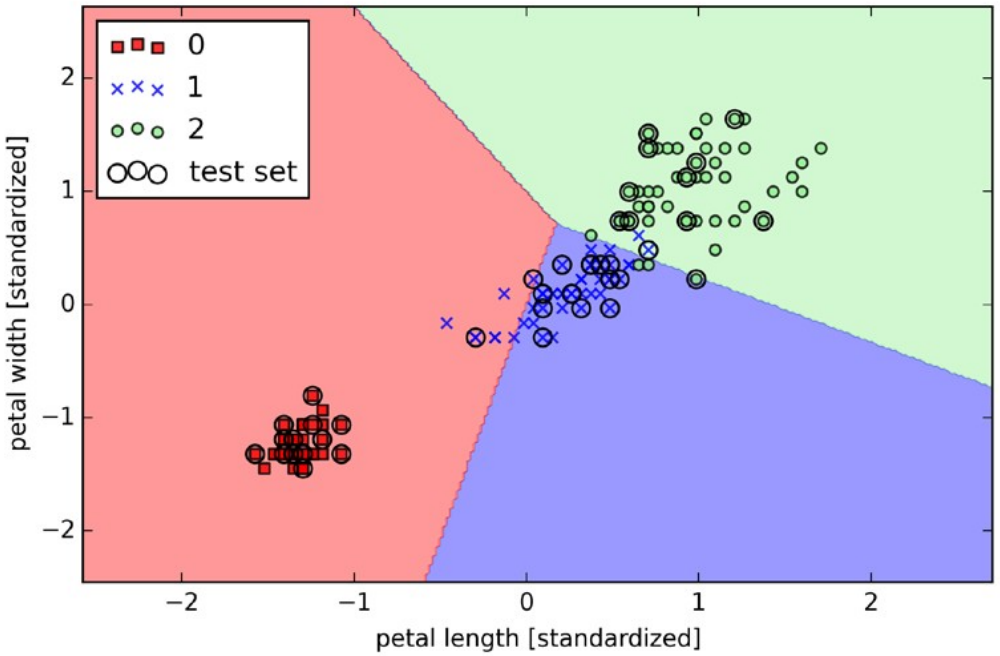
One of the often brightly colored parts of a flower immediately surrounding the reproductive organs; a division of the corolla.





https://i.ytimg.com/vi/E8Nrl9QLNq8/maxresdefault.jpg

The problem (Perceptron)



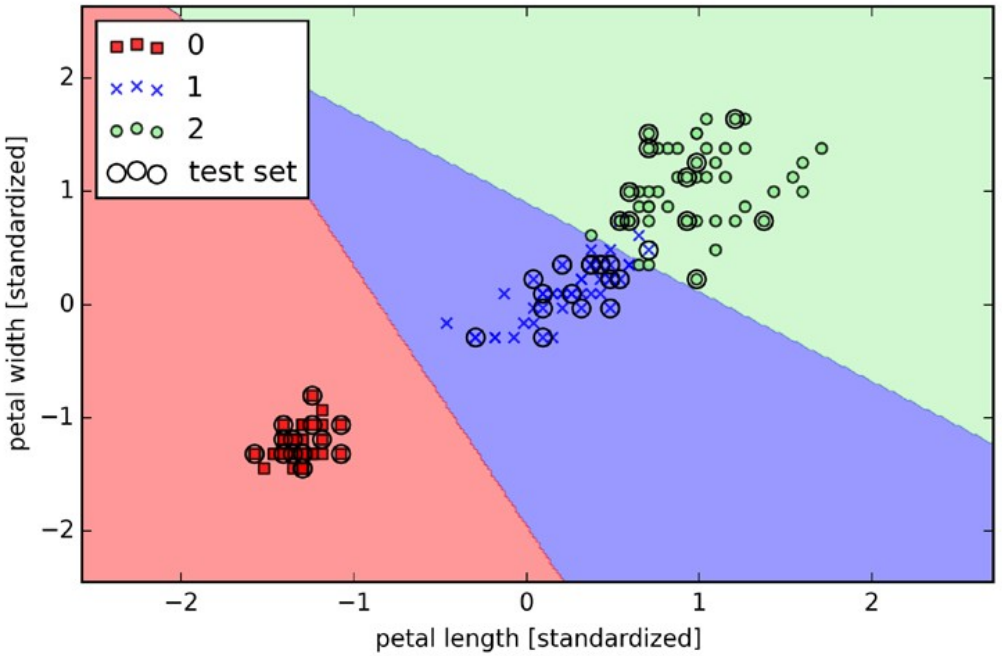
(p. 55: Raschka, 2015)

Not linearly separable ⟶ never converges

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

Something like this

LOGISTIC REGRESSION

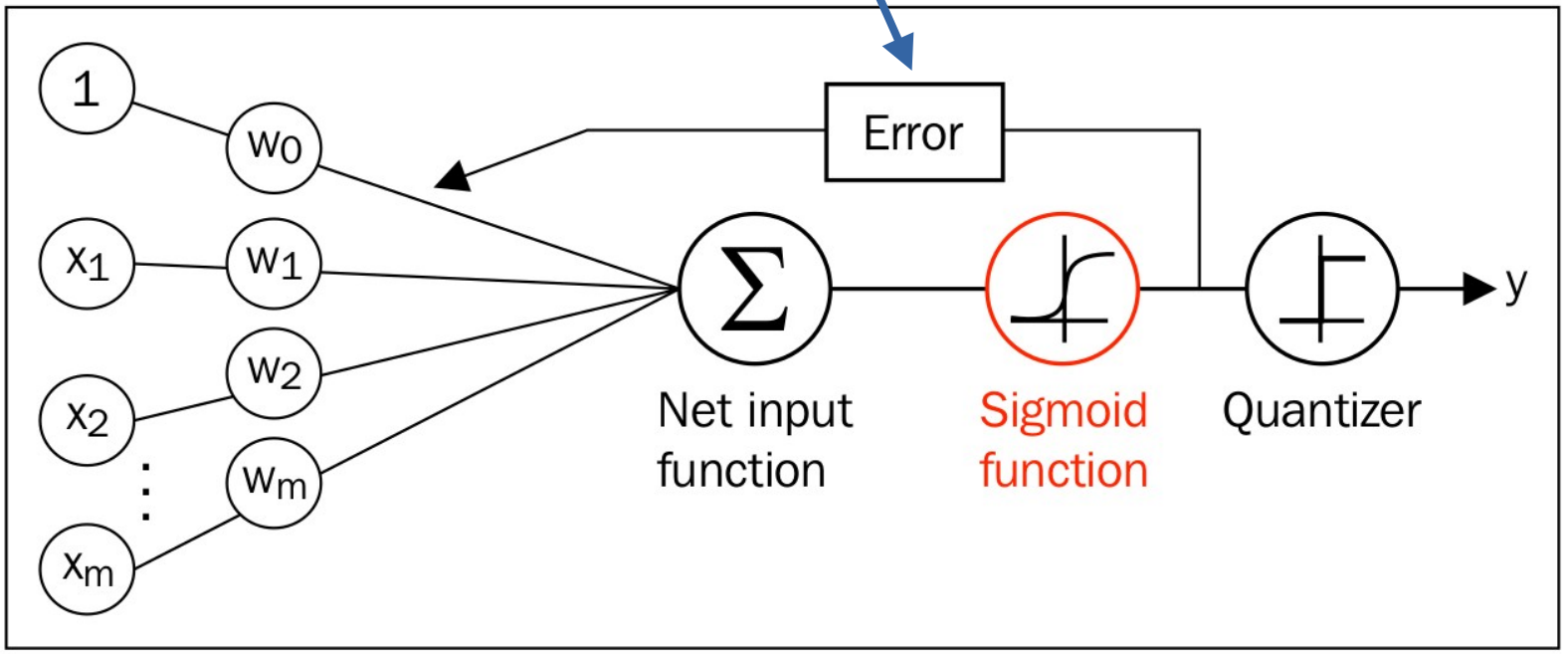


(p. 63: Raschka, 2015)

Separates flowers, while keeping errors at a minimum

What's our cost function, $J(\boldsymbol{w})$?

i.e. how do we calculate this?



Updating weights

ADALINE fit

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

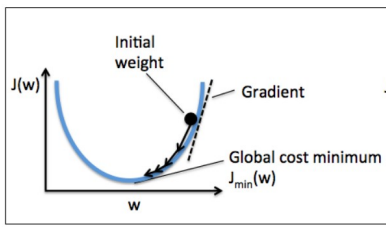
$\Delta \boldsymbol{w}$: change in $\boldsymbol{w}$

η: learning rate

$\nabla J(\boldsymbol{w})$: gradient of cost function $J(\boldsymbol{w})$

$$J(\boldsymbol{w})_{LINEAR} = \left(\sum_{i=1}^{n} (\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2\right)/2$$

```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
```

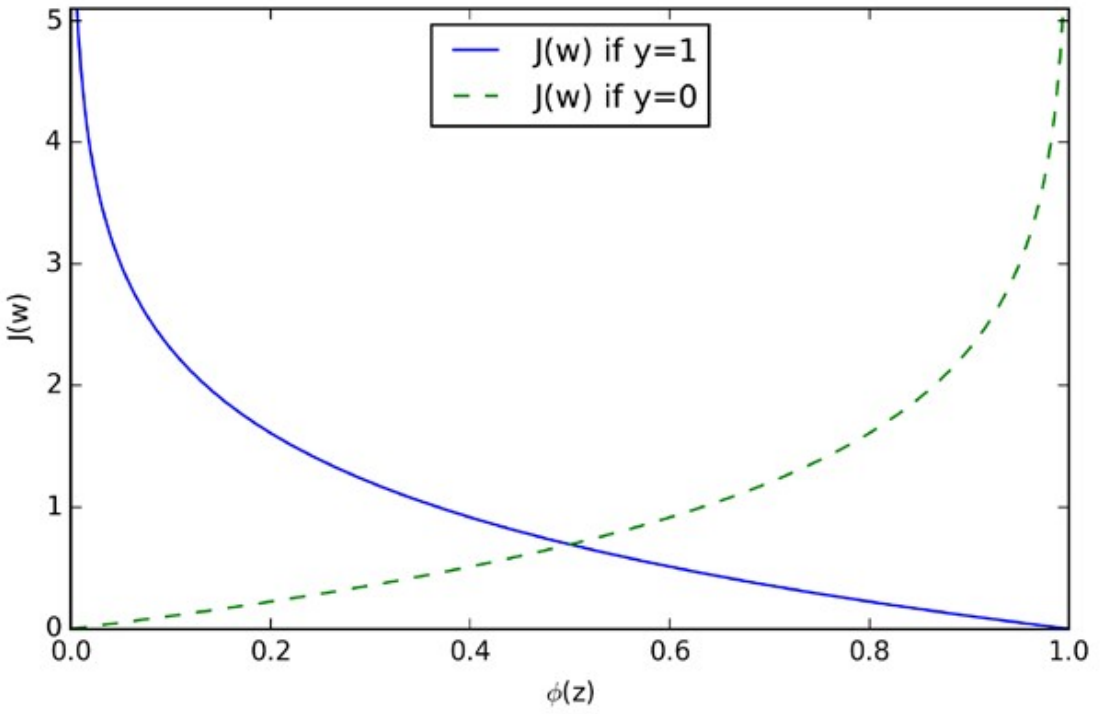


(p. 40: Raschka, 2015)

$$J(\boldsymbol{w})_{LOGISTIC} = -\sum_{i=1}^{n} \boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i) + (1 - \boldsymbol{y}_i) \log(1 - \hat{\boldsymbol{y}}_i)$$

Where have you seen this before?

Cost function for one instance



(p. 61: Raschka, 2015)

L1 and L2 regularisation

$$J(\boldsymbol{w})_{L1} = -\sum_{i=1}^{n} \boldsymbol{y}_i \log\left(\phi(\boldsymbol{z}_i)\right) + (1-\boldsymbol{y}_i)\log\left(1-\phi(\boldsymbol{z}_i)\right) + \lambda \|\boldsymbol{w}\|_1$$

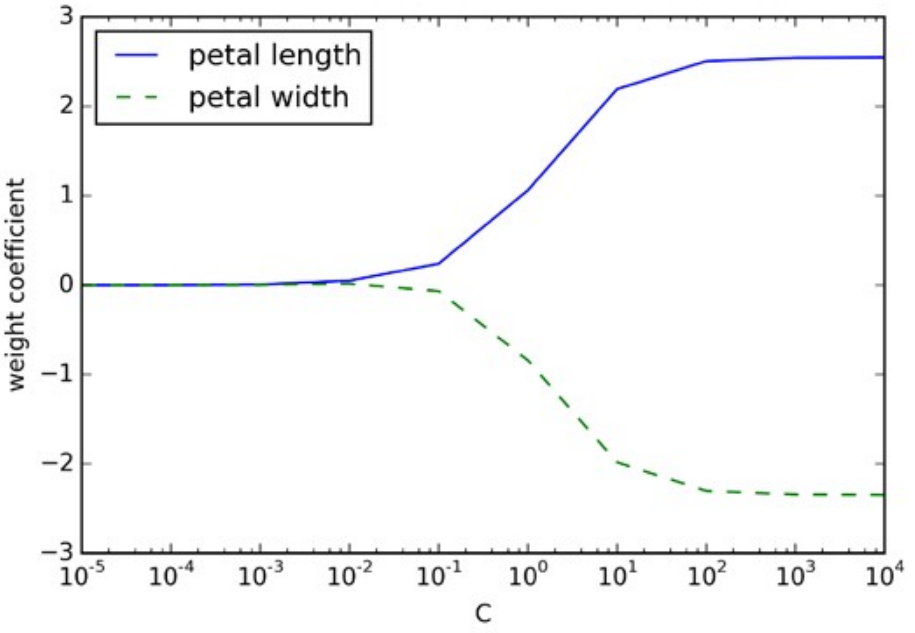
$$J(\boldsymbol{w})_{L2} = -\sum_{i=1}^{n} \boldsymbol{y}_i \log\left(\phi(\boldsymbol{z}_i)\right) + (1-\boldsymbol{y}_i)\log\left(1-\phi(\boldsymbol{z}_i)\right) + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2$$

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

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

L2 regularisation

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




```
logR_l2 = LogisticRegression(penalty='l2', C=10)
```


(p. 68: Raschka, 2015)

Updating weights – the general case

proof on p. 64

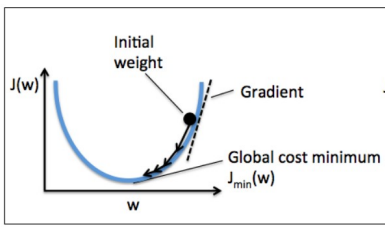
$$\Delta w_j = -\eta \frac{\delta J}{\delta w_j} = \eta \sum_{i=1}^{n} \left(y_i - \phi(z_i) \right) x_j$$

$$\phi(z)_{LINEAR} = z$$

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

```
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
```

ADALINE fit



(p. 40: Raschka, 2015)

Exercise in class

```python
import numpy as np
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def cost_function(y, yhat):
    return -np.sum(y * np.log(yhat) - (1 - y) * np.log(1 - yhat))

def gradient(X, y, yhat):
    return np.dot((y - yhat), X)

class LogisticRegressionGD:
    def __init__(self, eta=0.0001, n_iterations=1000, tol=1e-4):
        self.eta = eta
        self.n_iterations = n_iterations
        self.w_ = None
        self.cost_ = []
        self.tol = tol
        self.n_iter_ = 0

    def fit(self, X, y):

        X = np.insert(X, 0, 1, axis=1) # adding intercept term
        self.w_ = np.zeros(X.shape[1])
        # Gradient Descent
        for _ in range(self.n_iterations):
            self.n_iter_ += 1 ## go through number of iterations
            # Calculate linear combination of features and weights
            linear_model = None ## WHAT IS SUPPOSED TO BE HERE??
            # Apply sigmoid function to get probabilities
            yhat = None ## WHAT IS SUPPOSED TO BE HERE??

            self.cost_.append(cost_function(y, yhat))

            # Update weights using the gradient
            self.w_ += None## WHAT IS SUPPOSED TO BE HERE?
            ## stop when we have found a minimum
            if self.n_iter_ > 1: ## we need at least two observations
                diff = None ## WHAT IS SUPPOSED TO BE HERE??
                if diff < self.tol:
                    break
```

```python
X, y = load_iris(return_X_y=True)
X = X[y < 2, 0:3:2]  ## petal length and sepal length
y = y[y < 2]

logreg = LogisticRegressionGD(eta=0.00001, n_iterations=1000, tol=1e-4)
logreg.fit(X, y)

plt.figure()
plt.xlabel('Iteration (#)')
plt.ylabel('Cost (J(w))')
plt.title('Converged at iteration: ' + str(logreg.n_iter_))
plt.plot(logreg.cost_)
plt.show()
```

Find code on Brightspace

Interim summary

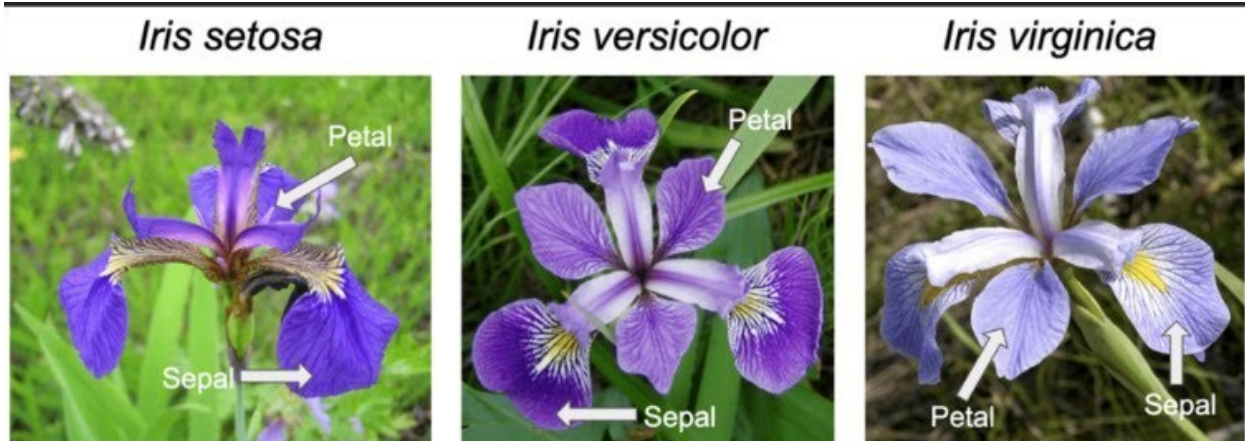
- Gradient descent can be put into a common framework
  - $\Delta w_j = -\eta \frac{\delta J}{\delta w_j} = \eta \sum_{i=1}^{n} \left(\boldsymbol{y}_i - \phi(\boldsymbol{z}_i) \right) x_j$
  - we would thus also be able to do Poisson machine learning (not covered here), and any other transformation that would be meaningful
- Now, we need to move beyond fitting and test

Steps of machine learning analysis

1. Selection of features.
2. Choosing a performance metric.
3. Choosing a classifier and optimization algorithm.
4. Evaluating the performance of the model.
5. Tuning the algorithm.

(p. 50: Raschka, 2015)

1. Selection of features, i.e. X



https://i.ytimg.com/vi/E8Nrl9QLNq8/maxresdefault.jpg

It is a good idea to standardise these

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

2. Choosing a performance metric

E.g. Accuracy of classification or Reduction of error

3. Choosing a classifier and an optimization algorithm

Logistic regression and gradient descent

Linear regression and gradient descent

4. Evaluating the performance of the model

Perform out-of-sample testing

Do this in a cross-validated manner where the data, several times, is partitioned into training and test data sets, to minimise bias.

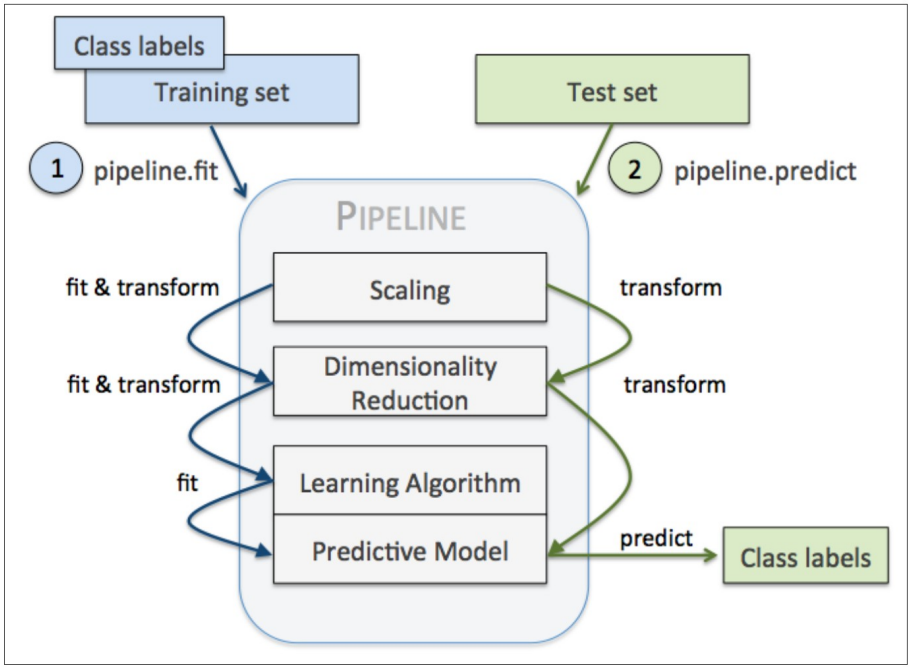
5. Tuning our model

If our model performs better on the training data than on the test data, we are likely to be overfitting. Regularisation could be a solution. We can do e.g. *grid search* or *randomised search* (next week) to find the optimal *C*-values in logistic and linear regression

We could also try to reduced the number of features in the model and only retain the ones explaining the most variance, thereby reducing the risk of overfitting, e.g. by using *Principled component analysis* (in two weeks)

General structure

(p. 172: Raschka, 2015)



Can be implemented in *scikit-learn* (in four weeks)

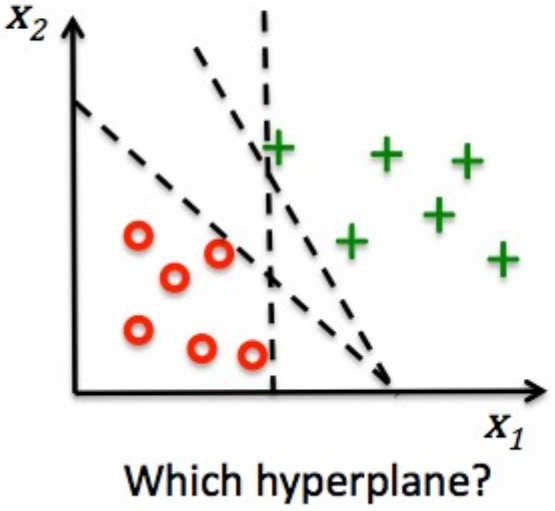
```
from sklearn.pipeline import Pipeline
pipe = Pipeline(steps=[("scaler", scaler), ("pca", pca), ("logistic", logistic)])
```


https://scikit-learn.org/stable/auto_examples/compose/plot_digits_pipe.html#sphx-glr-auto-examples-compose-plot-digits-pipe-py

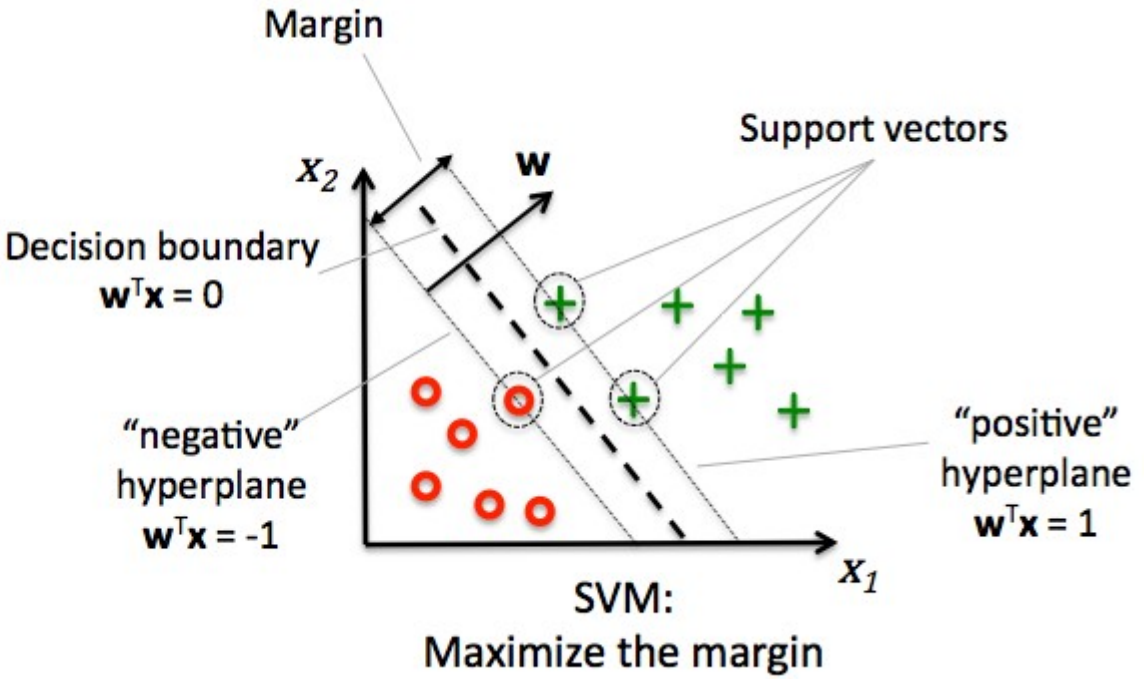
Live coding

RUNNING LOGISTIC REGRESSION

SUPPORT VECTOR MACHINES



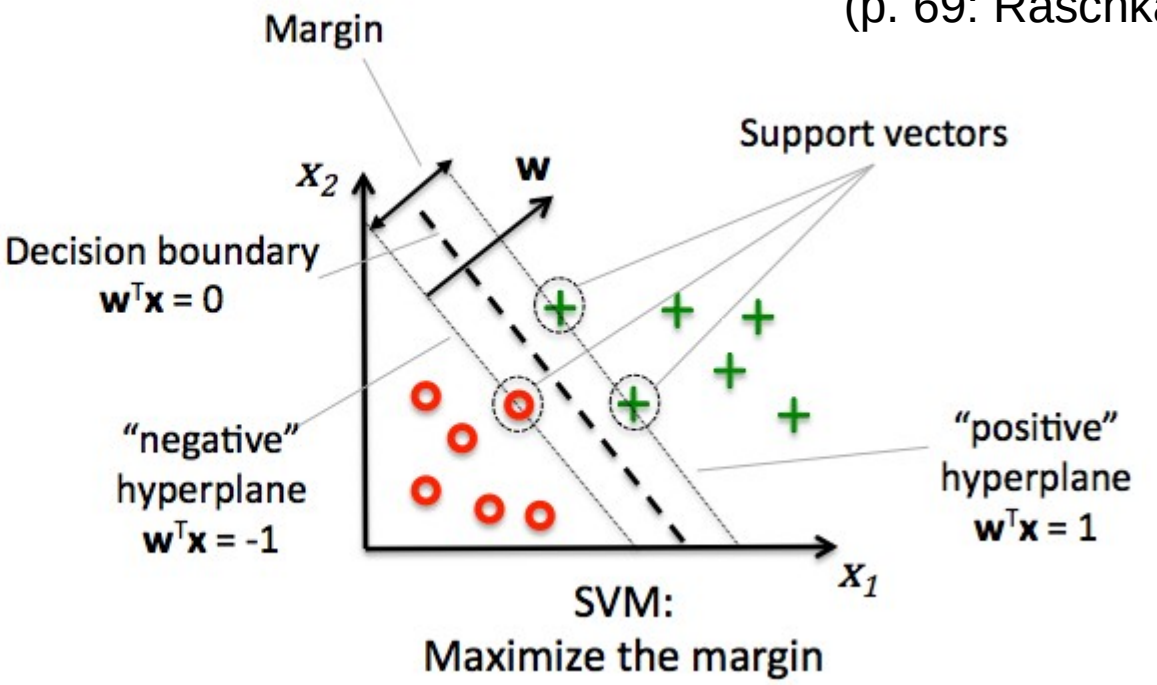
Which hyperplane?



SVM:
Maximize the margin

(p. 69: Raschka, 2015)

(p. 69: Raschka, 2015)



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

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

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

Subtraction

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

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

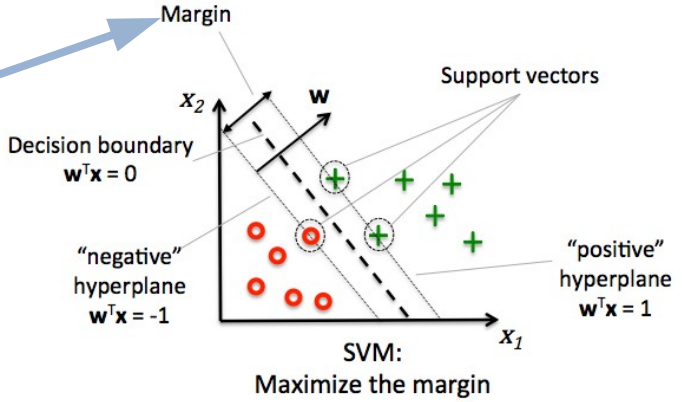
... we get: $\boldsymbol{w}^{T}\dfrac{\left(\boldsymbol{x}_{\mathbf{pos}}-\boldsymbol{x}_{\mathbf{neg}}\right)}{\|\boldsymbol{w}\|}=\dfrac{2}{\|\boldsymbol{w}\|}$

This will optimise the width of the margin

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

under the constraints:

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



(p. 69: Raschka, 2015)

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

Which?

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

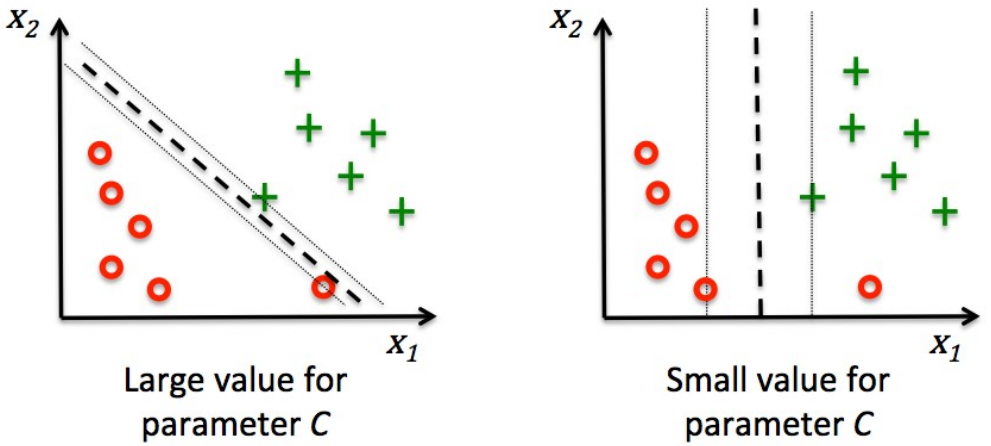
Introducing ξ

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

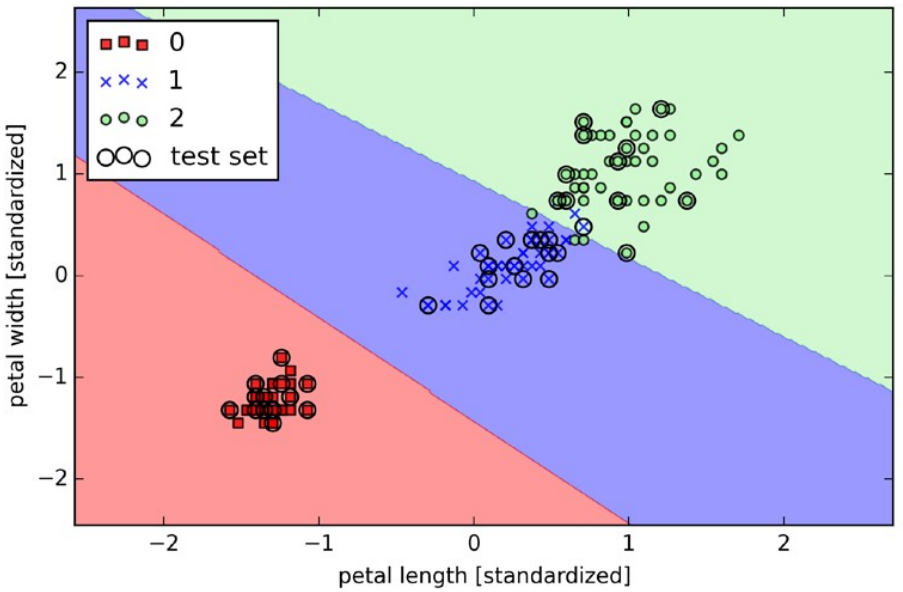
now we need to minimize: $\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i^n \xi^{(i)}\right)$ (p. 72: Raschka, 2015)

to maximise the margin

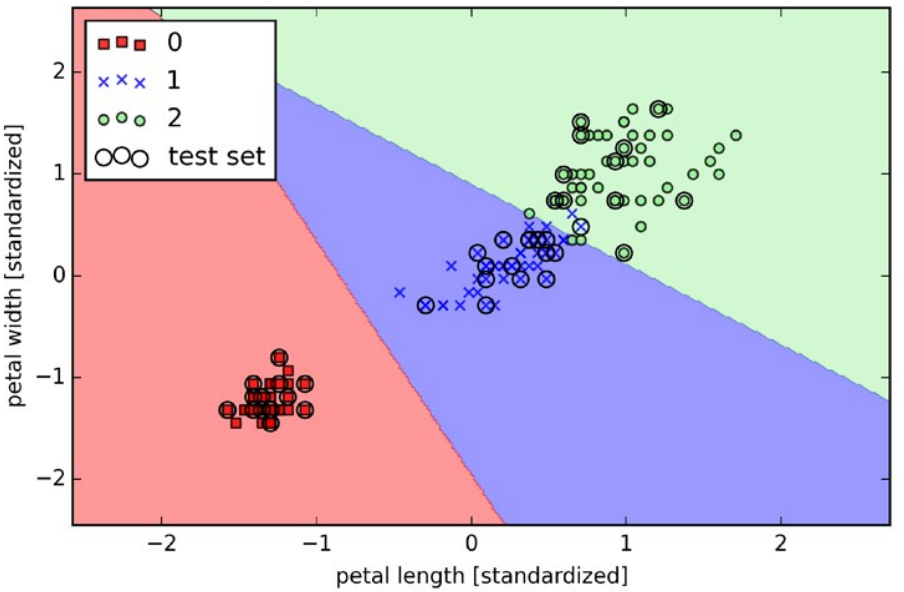
Large *C*: narrow margin – penalises errors harshly
Small *C*: wide margin – penalises errors mildly



Comparison to logistic regression



Support Vector Machine (p.73, 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)

Summary

- Our standard regression methods can be set up for prediction
- Scikit-learn can help us set the whole pipeline for classification
- Support Vector Machines are very similar to logistic regression, but may generalise better

Learning goals and outline

Linear and logistic regression revisited (machine learning)

1) Understanding how linear and logistic regression can be adapted to a classification framework
2) Understanding how gradient descent can provide a common framework across regression fits
3) Getting acquainted with Support Vector Machines

The course plan

Week 1: *Introduction*

Instructor sessions: *Setting up* R *and* Python *and recollection of the general linear model*

Week 2: *Multilevel linear regression*

Instructor sessions: *Modelling subject level effects – and how do they differ from group level effects?*

Week 3: *Link functions and fitting generalised linear multilevel models*

Instructor sessions: *What to do when the response variable is not continuous?*

Week 4: *Evaluating Generalised linear mixed models*

Instructor sessions: *How do we assess how models compare to one another?*

Week 5: *Explanation and Prediction*

Instructor sessions: *Code review*

Week 6: *Mid-way evaluation and Machine Learning Intro*

Instructor sessions: *Getting Python Running*

Week 7: *Linear and logistic regression revisited (machine learning)*

Instructor sessions: *Categorizing responses based on informed guesses*

Week 8: *Model evaluation and hyperparameter tuning*

Instructor sessions: *Code review*

Week 9: *Dimensionality Reduction, Principled Component Analysis (PCA)*

Instructor sessions: *What to do with very rich data?*

Week 10: *Outlook, unsupervised classification and neural networks*

Instructor sessions: *Data with no labels and networks*

Week 11: *Organising and preprocessing messy data*

Instructor sessions: *Code review*

Week 12: *Final evaluation and wrap-up of course*

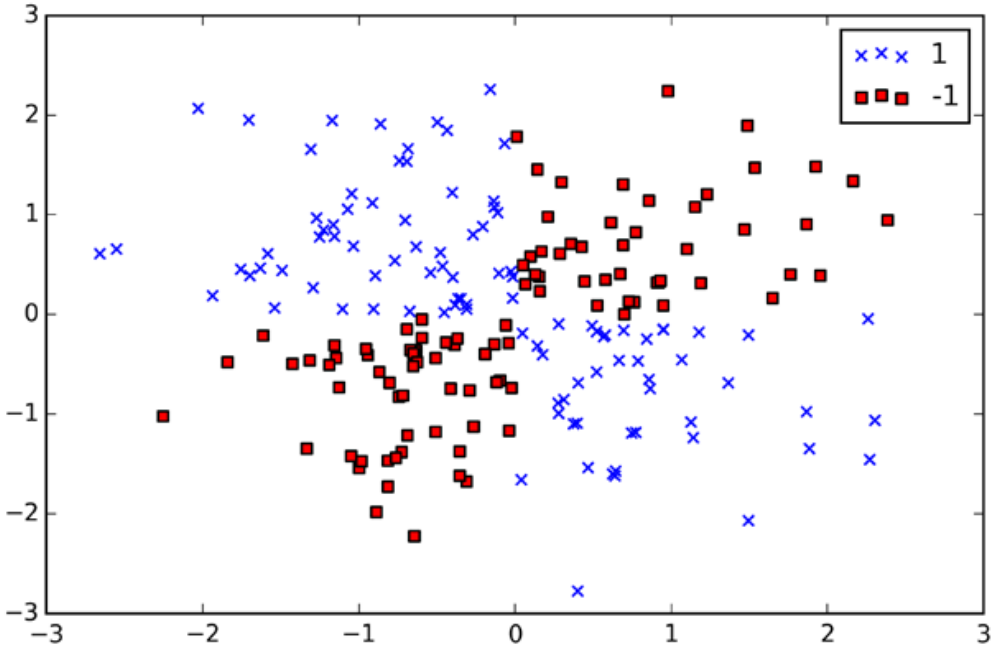
Instructor sessions: *Ask anything!*

Next time

- Doing search for the best hyperparameters
- Confusion matrices
- Validation Accuracy

Extra slides on non-linear SVM’s

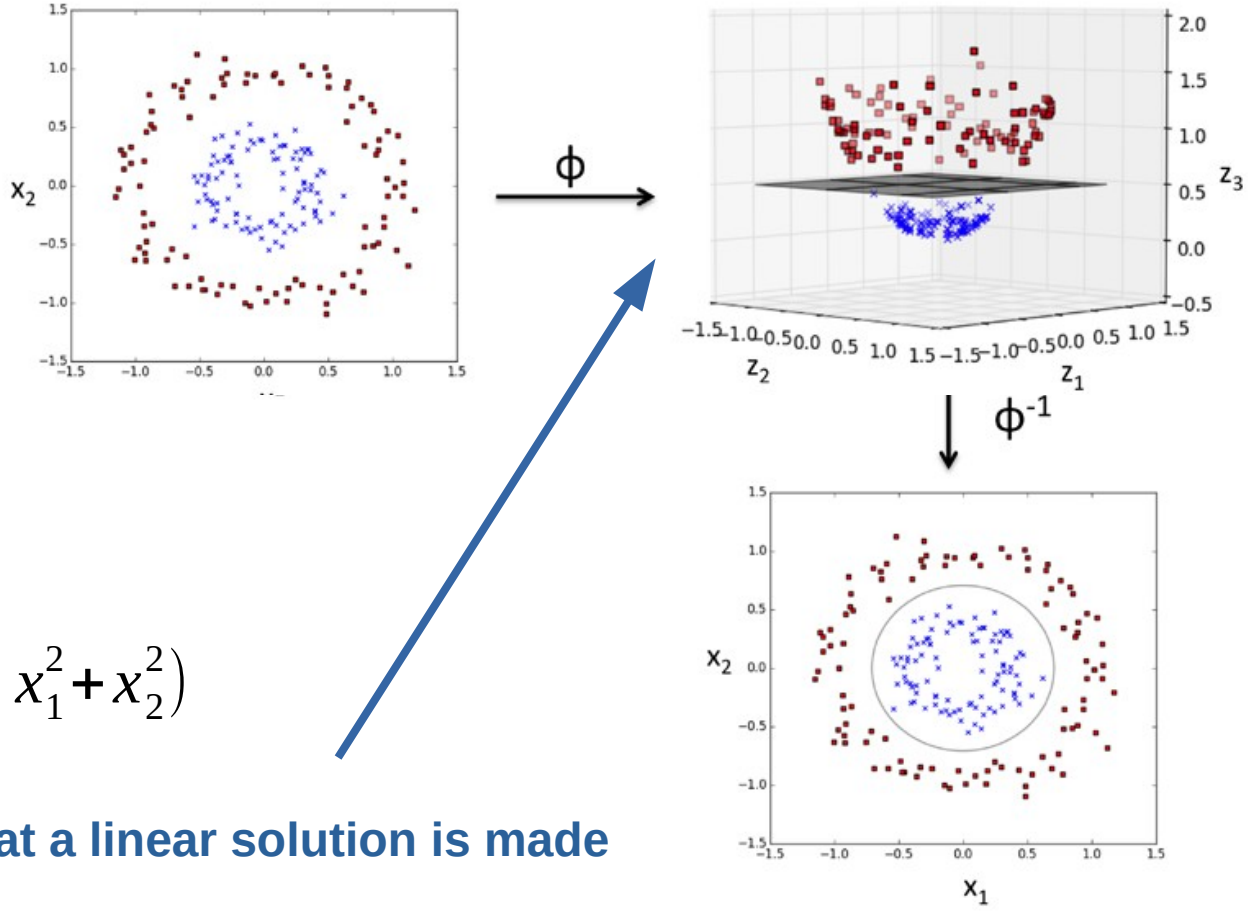
A brief look at a non-linear problem



(p. 75: Raschka, 2015)

Map onto higher-dimensional space

$$\phi(\cdot)$$



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

Note that a linear solution is made

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

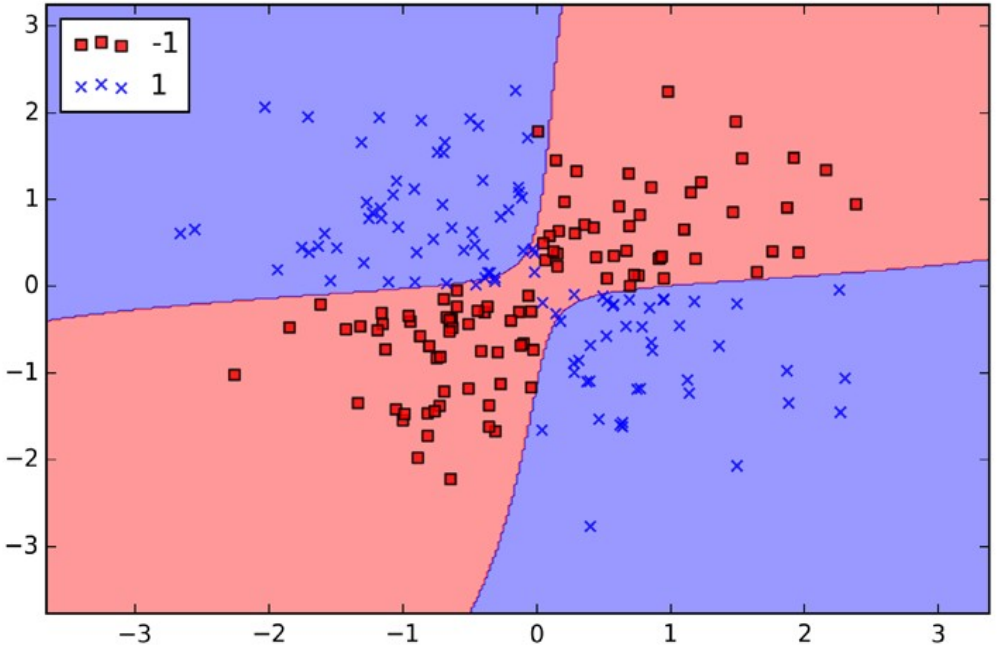
Creating the higher dimensions can be computationally expensive

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

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

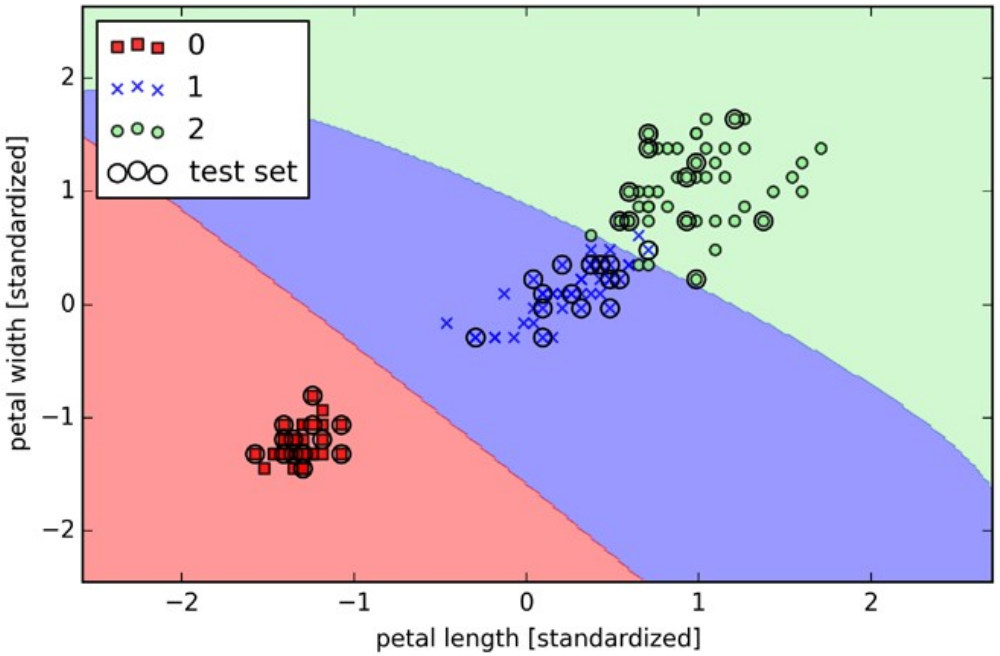
$$\left(\gamma = \frac{1}{2\sigma^2}, \text{ also called the precision} \right)$$

Non-linear decision boundaries



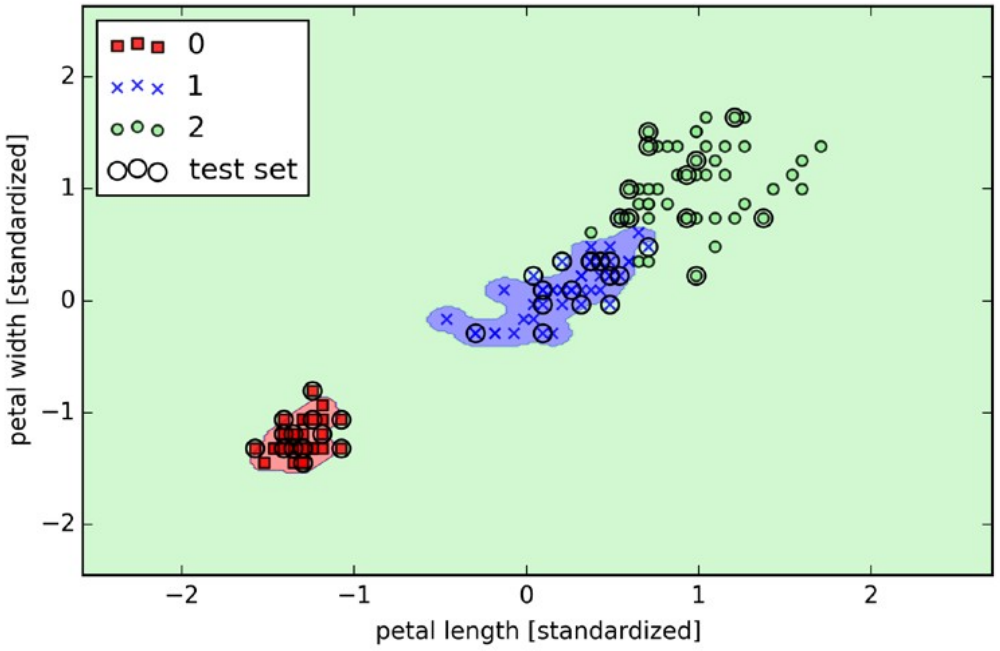
(p. 78: Raschka, 2015)

Low γ - soft boundary



(p. 79: Raschka, 2015)

High γ - tight boundary



(p. 80: Raschka, 2015)