Session 11:

Machine learning introduction

Andreas Bjerre-Nielsen

Agenda

1. Why machine learing
2. What is machine learning
3. Classification models
   A. the perceptron
   B. beyond the perceptron
   C. maximum margin classification

Learning ML

- During lectures copy code for see what it does - listen to me
- After lecture > understand code details
- Learn with your group - VERY IMPORTANT!

Why machine learning

Value of modelling

Why are models useful?

Models are pursued with differens aims. Suppose we have a regression model, $y = X\beta + \epsilon$.

- Social science:
  - They teach us something about the world.
  - We want to estimate $\hat{\beta}$ and distribution
- Data science:
  - To make optimal future decisions and precise predictions, i.e. $\hat{y}$.

Model fragility (1)

What is a polynomial regression?

- Fitting a curve with an *n-dimenstional polynomial*
- Can fit any "regular" curve ~ Taylor Series Approximation.

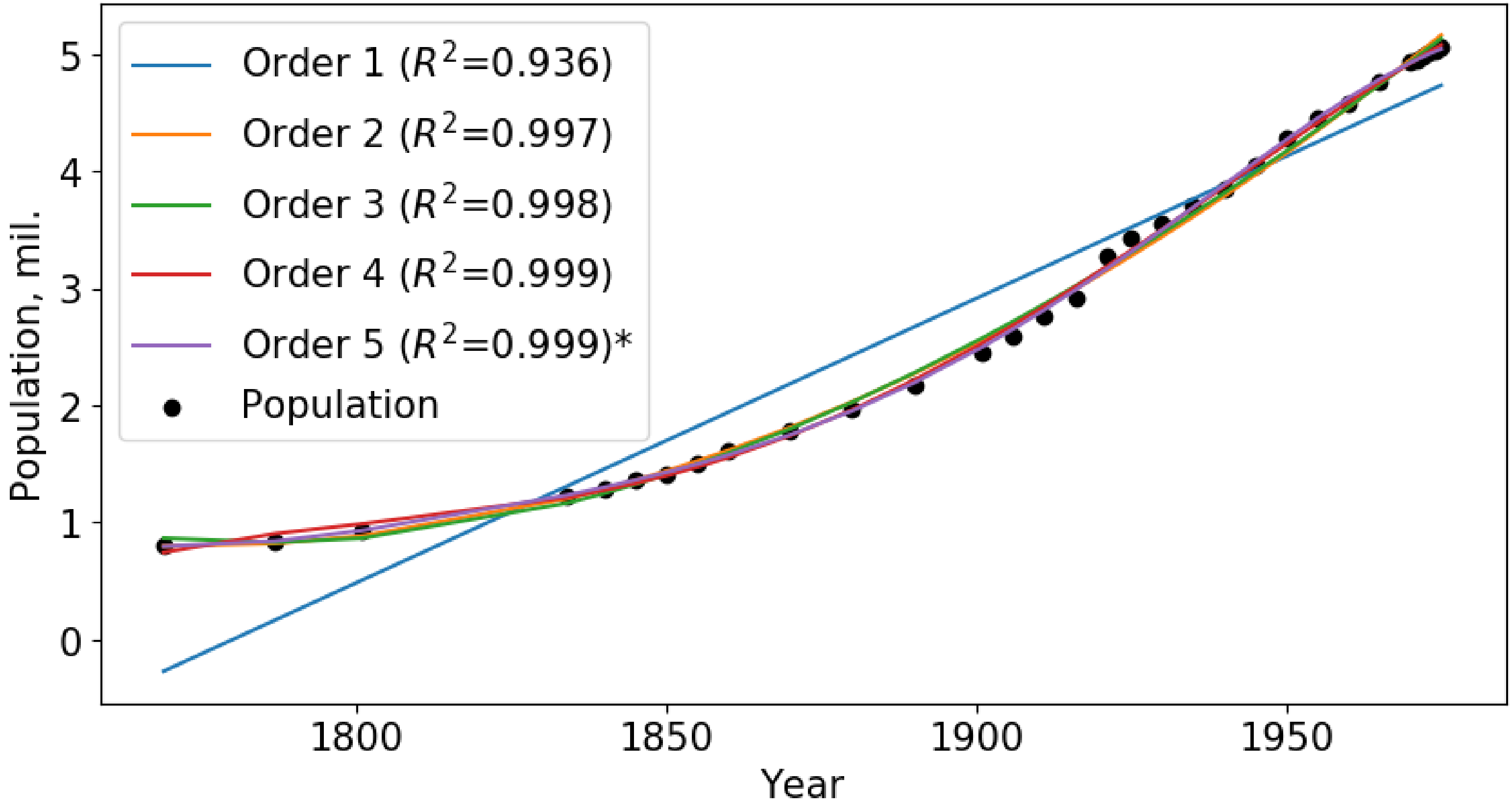
Model fragility (2)

Suppose we build models of the size of the Danish population, how do polynomial fits perform?

- We use data from the years 1769-1975.

In [18]:
```
f_pop1
```

Out[18]:

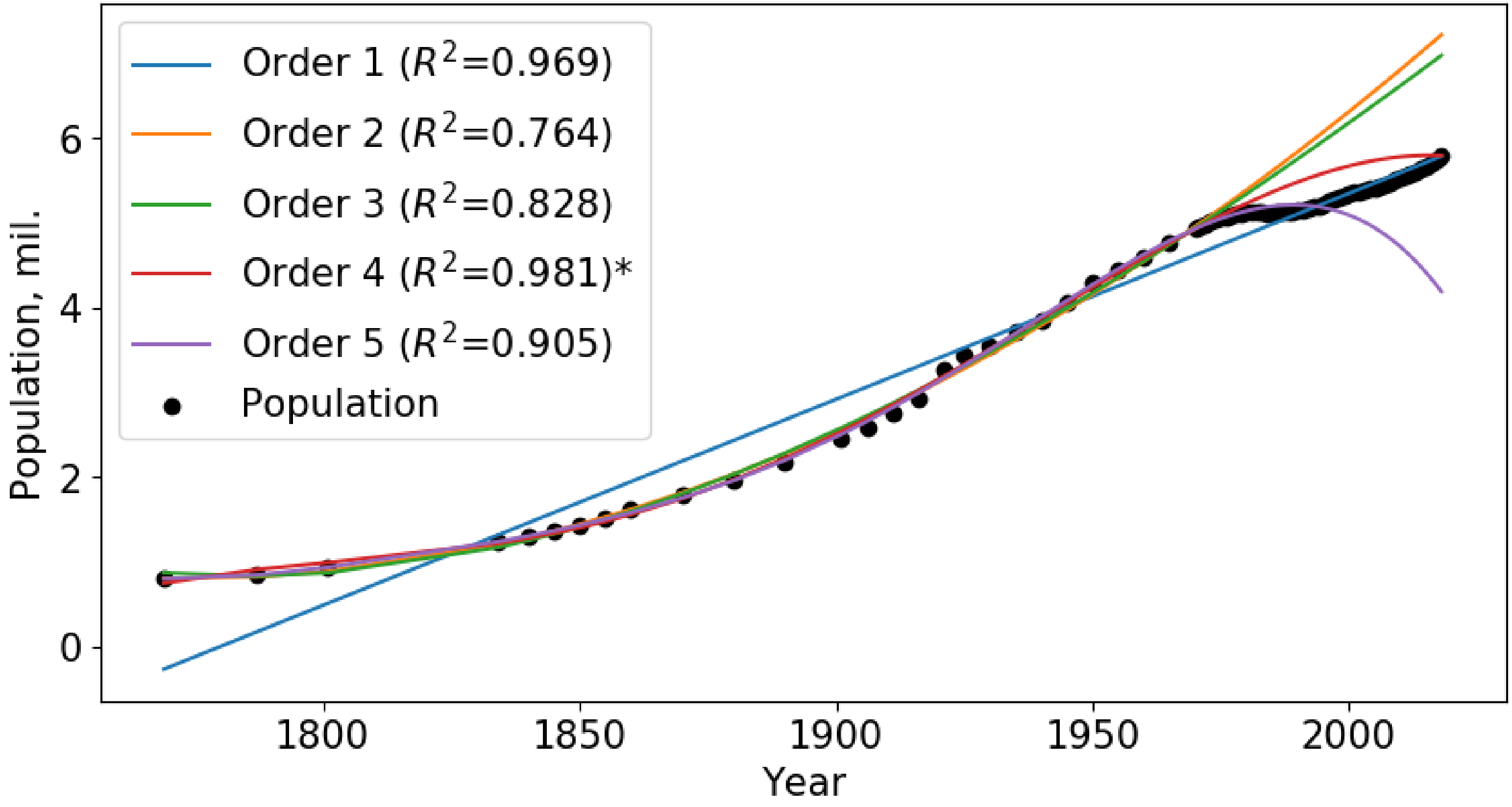


Model fragility (3)

Which model performs best when we extend the period to 1975?

In [5]:
```
f_pop2
```

Out[5]:

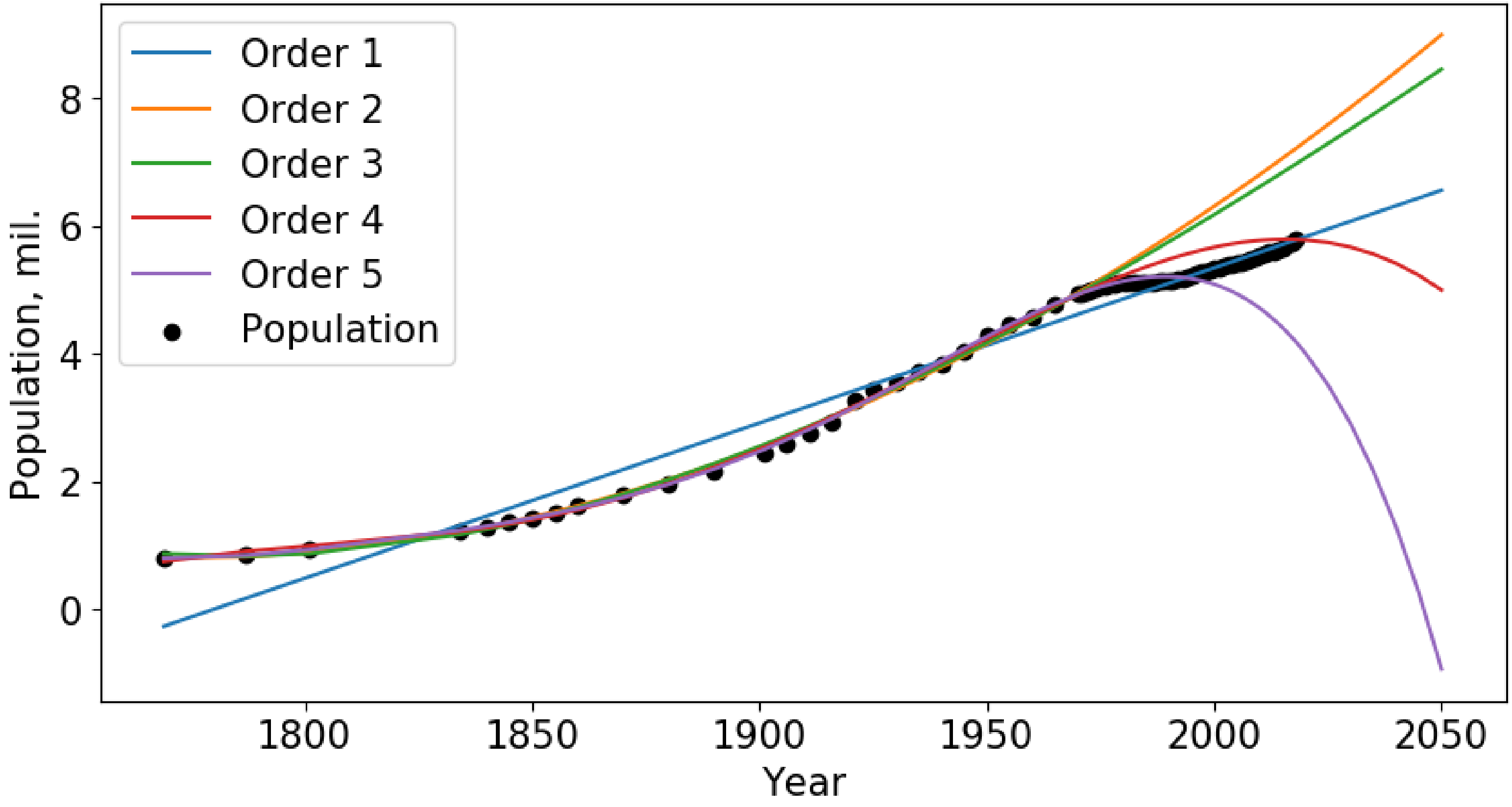


Model fragility (4)

What happens if we extend the prediction period until 2050? See the fifth order.

In [6]:
```
f_pop3
```

Out[6]:



Model fragility (5)

What trade off do we face in modelling?

- Making a model that is to simple and does not capture enough of data (`underfitting`)
- Making a model with great fit on estimation data, but poor out-of-sample prediction (`overfitting`)

The goal of machine learning is to find models that minimize these two problems simultaneously.

Machine learning overview

What is machine learning (1)

Can you define machine learning, i.e. ML?

- Supervised learning
  - Models designed to infer a relationship between input and **labeled** data.
  - Requirement: labels on data
- Unsupervised learning
  - Find patterns and relationships from **unlabeled** data.
    - This may involve clustering, dimensionality reduction and more.

What is machine learning (2)

How might this be useful for social scientists?

- Supervised:
  - Improve estimation
    - Model validation (not only theory)
    - Other methods
  - We can generate new data
  - Better predictive models
- Unsupervised:
  - We can generate new data

What is machine learning (3)

How can we categorize a supervised learning model?

Suppose we have model

$$y = w_0 + w_1 x_1 + .. + w_k x_k$$

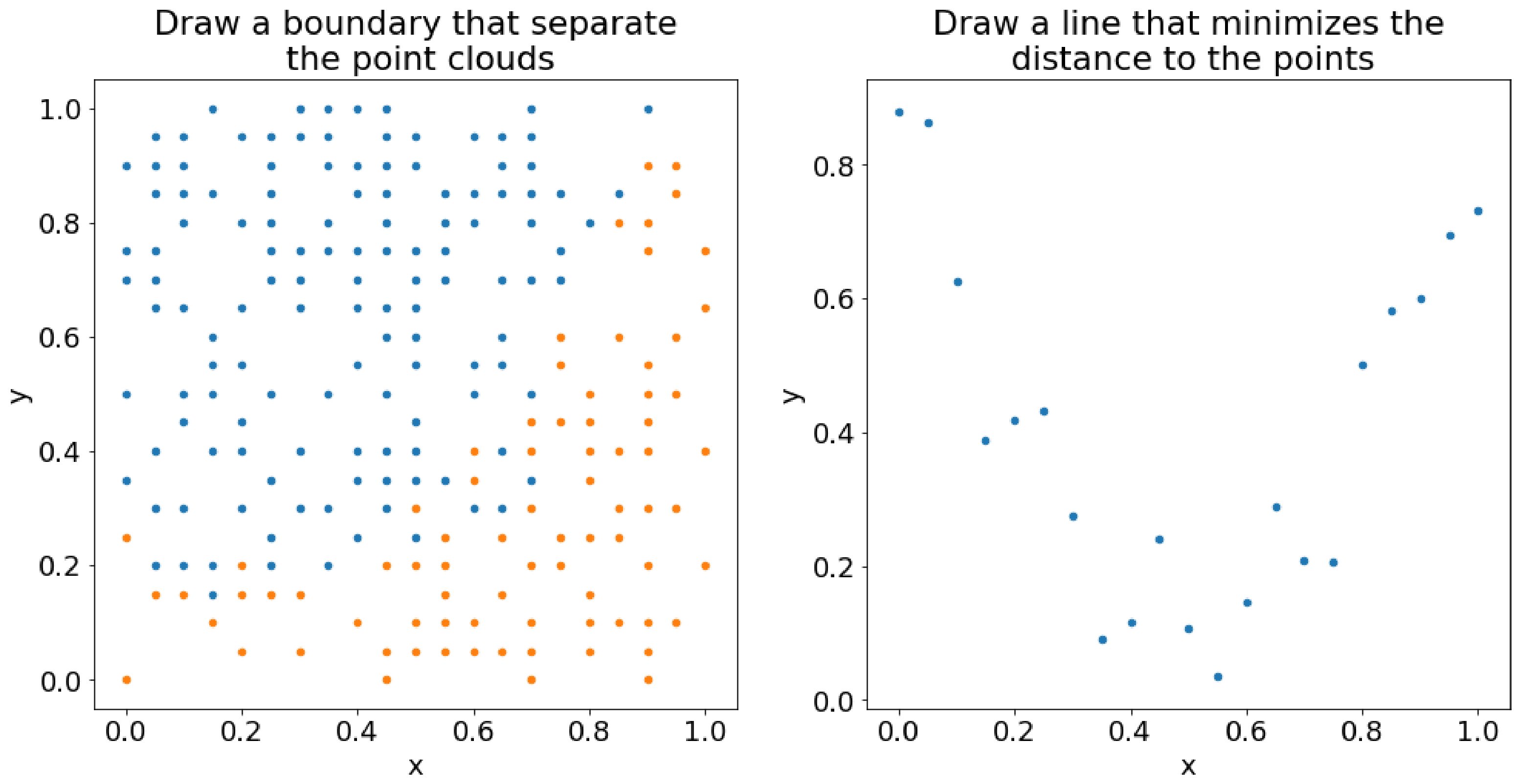
- We distinguish by type of the `target` variable y

What is machine learning (4)

Which one is classification, which one is regression?

In [7]:
```
f_identify_question
```

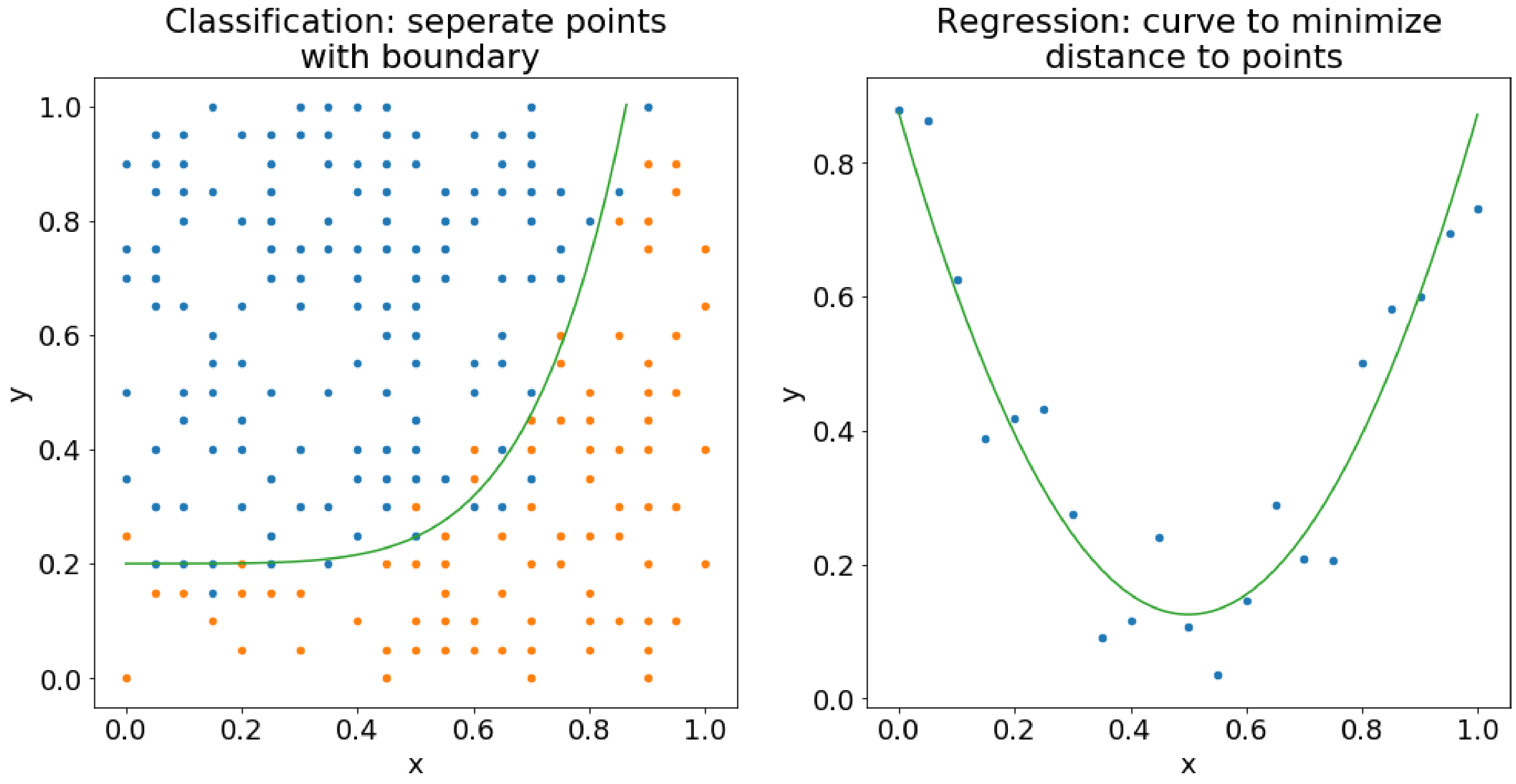
Out[7]:



What is machine learning (5)

In [8]:
```
f_identify_answer
```

Out[8]:



Regression models

What are examples of regressions models?

- Output is income, life expectancy, education length (years)

What is the underlying data of the target, y?

- target is `continuous`

Classications models

What are examples of classication models?

What is the underlying data of the target, y?

- target is `categorical`
  - sometimes known as`factor`
  - can also be `bool` or `int`

ML lingo and concepts

- `net-input`, $z_i = \underbrace{\boldsymbol{w}^T \boldsymbol{x}_i}_{vector\ form} = \underbrace{1 \cdot w_0 + w_1 x_{i,1} + \ldots + w_k x_{i,k}}_{expanded\ form}$

- `feature` vector, $\mathbf{x}_i$, i.e a row of input variables
  - ~ explanatory variables in econometrics
- `weight` vector, $\mathbf{w}$, i.e model parameters
  - ~ coefficients in econometrics where denoted β
- `bias` term, w_0, i.e. the model intercept
  - ~ the constant variable in denoted β_0

The perceptron model

The artificial neuron (1)

We are interested in making a decision rule $\phi : \mathbb{R}^p \to \{-1, 1\}$.

$$\phi(z_i) = \begin{cases} 1, & z_i > 0 \\ -1, & z_i \leq 0 \end{cases}$$

- `unit step function`, ϕ, checks if value exceeds threshold

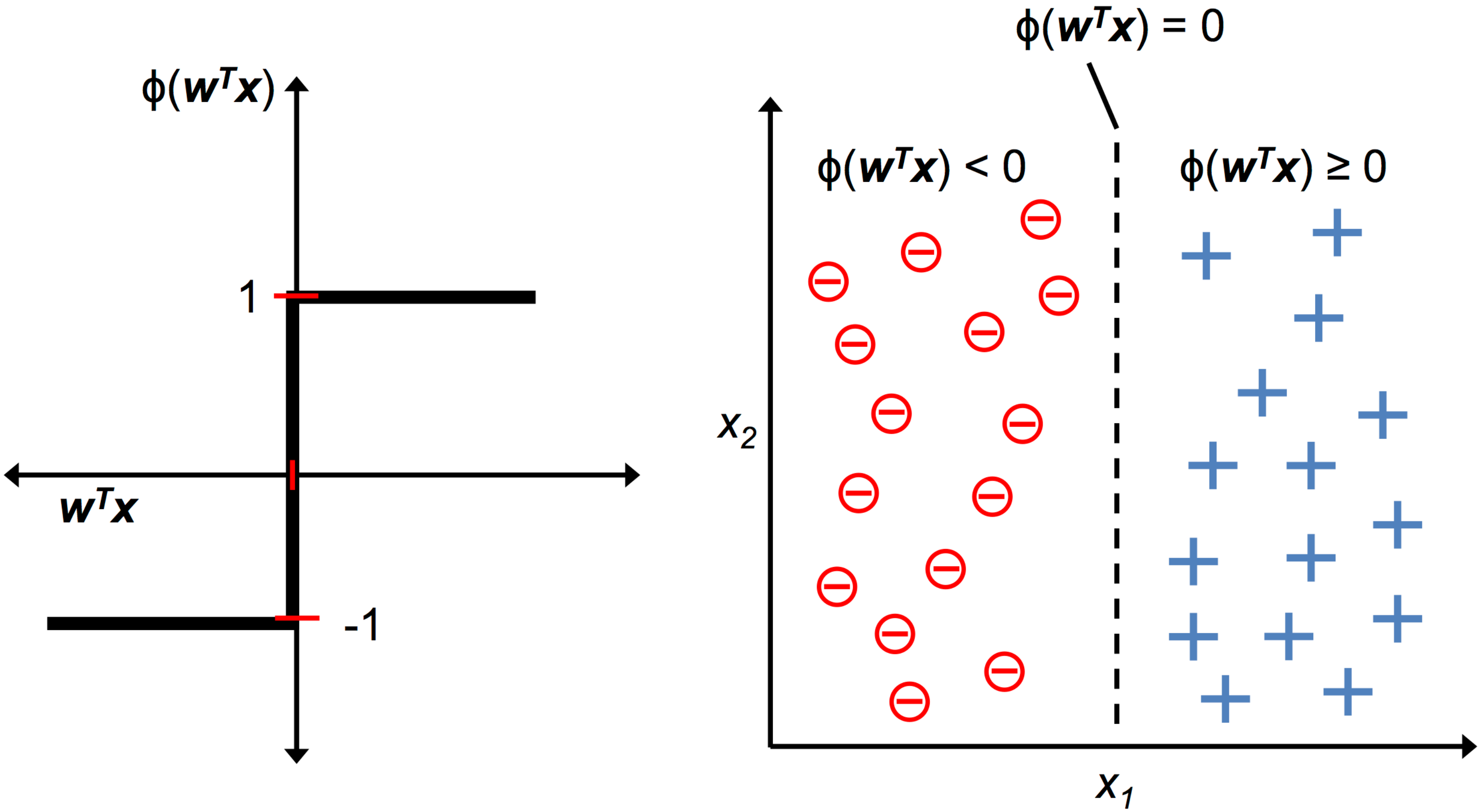
The artificial neuron (2)

Quiz: what are the input dimensions of the neuron, what is the output dimension?

- Input is the p-dimensional space, $\mathbb{R}^p$.
- Output is binary, either -1 or 1.

The artificial neuron (3)

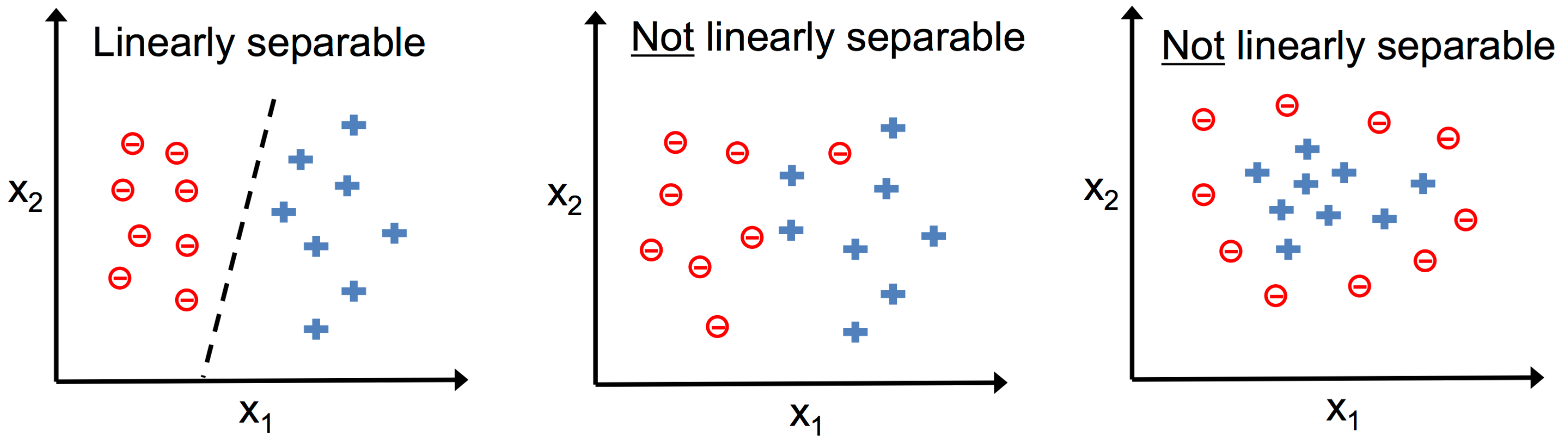
The unit step function (left) and the decision boundary (right)



The artificial neuron (4)

When does the artical neuron work?

If the two target types are linearly separable:



The perceptron learning rule (1)

How do we estimate the model parameters?

1. initialize the weight with small random number
2. for each training observation, i=1,..,n
   A. compute predicted target, $\hat{y}_i$
   B. update weights $\hat{w}$

The perceptron learning rule (2)

How do we predict the target?

Single observation:

$$\begin{aligned}
\hat{y}_i &= \phi(z_i), \quad z_i = w_0 + w_1 x_{i,1} + \ldots + w_k x_{i,k}, \qquad \text{expanded notation} \\
\hat{y}_i &= \phi(z_i), \quad z_i = \hat{\boldsymbol{w}}^T \boldsymbol{x}_i, \qquad \text{vector notation}
\end{aligned}$$

Multiple observations

$$\hat{\boldsymbol{y}} = \phi(\boldsymbol{z}), \quad \boldsymbol{z} = \boldsymbol{X}\hat{\boldsymbol{w}}, \qquad \text{matrix notation}$$

The perceptron learning rule (3)

How do we update weights?

Weights are updated as follows:

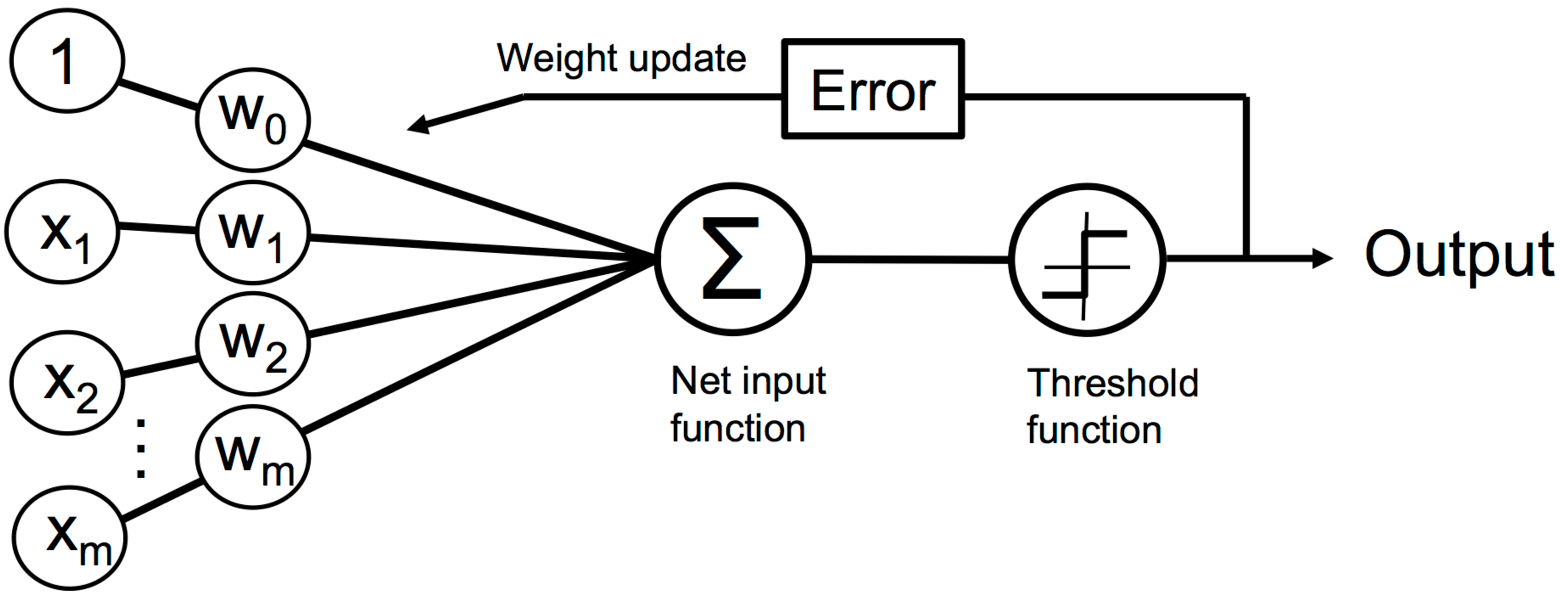
$$\hat{w} = \hat{w} + \Delta\hat{w}$$
$$\Delta\hat{w} = \eta \cdot (y_i - \phi(z_i)) \cdot \mathbf{x}_i$$

where η is the learning rate, and the first order derivative is:

$$\frac{\partial MSE}{\partial w} = \mathbf{X}^T\mathbf{e}$$

The perceptron learning rule (4)

The computation process



Implementation in Python (1)

How do we compute the net-input vectorized?

In [ ]:

```
X = np.random.normal(size=(3, 2)) # feature matrix
print('X:',X)

y = np.array([1, -1, 1]) # target vector
print('y:',y)

w = np.random.normal(size=(3)) # weight vector
print('w:',w)

# compute net-input
z = w[0] + X.dot(w[1:])
print('z:\n', z)
```

Implementation in Python (2)

How do we compute the errors vectorized?

In [ ]:

```
# compute errors
positive = z>0
y_hat = np.where(positive, 1, -1)
e = y - y_hat
SSE = e.T.dot(e)
```

Implementation in Python (3)

How do we compute the updated weights?

In [10]:

```
# learning rate
eta = 0.001

# first order derivative
fod = X.T.dot(e) / 2

# update weights
update_vars = eta*X.T.dot(e) # insert fod
update_bias = eta*e.sum()/2
```

Working with the perceptron (1)

We load the iris data.

In [32]:

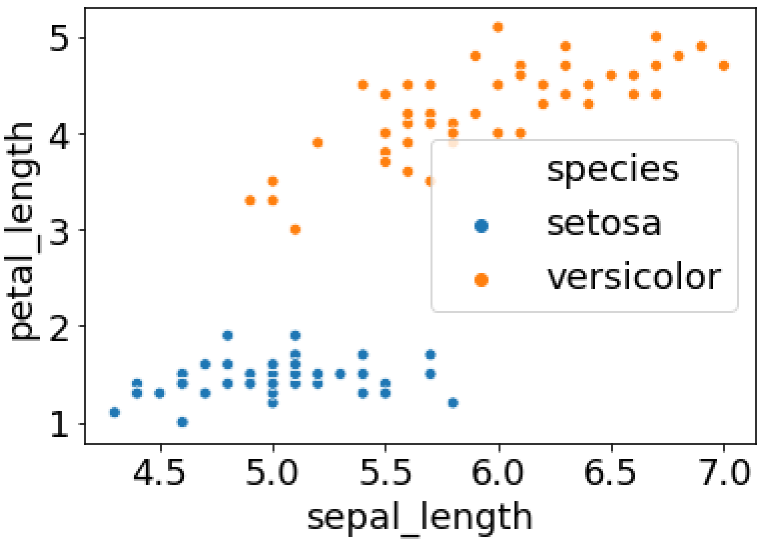
```
iris = sns.load_dataset('iris').iloc[:100] # drop virginica

X = iris.iloc[:, [0, 2]].values # keep petal_length and sepal_length
y = np.where(iris.species=='setosa', 1, -1) # convert to 1, -1

sns.scatterplot(iris.sepal_length, iris.petal_length, hue=iris.species)
# plt.scatter(iris.sepal_length, iris.petal_length)
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x185816a2c88>
```



Working with the perceptron (2)

How do we fit the perceptron model? perceptron definition

In [28]:

```
# initialize the perceptron
clf = Perceptron(n_iter=10)

# fit the perceptron
# runs 10 iterations of updating the model
clf.fit(X, y)
```

Out[28]:

```
<ch02.Perceptron at 0x1858311d0f0>
```

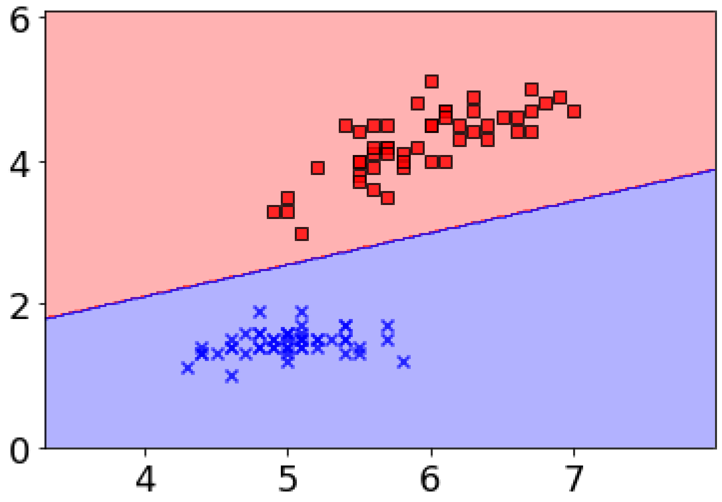
Working with the perceptron (3)

How can we evaluate the model??

In [36]:
```
print('Number of errors: %i' % sum(clf.predict(X)!=y))

# we plot the decisions
plot_decision_regions(X,y,clf)
```

```
Number of errors: 0
```

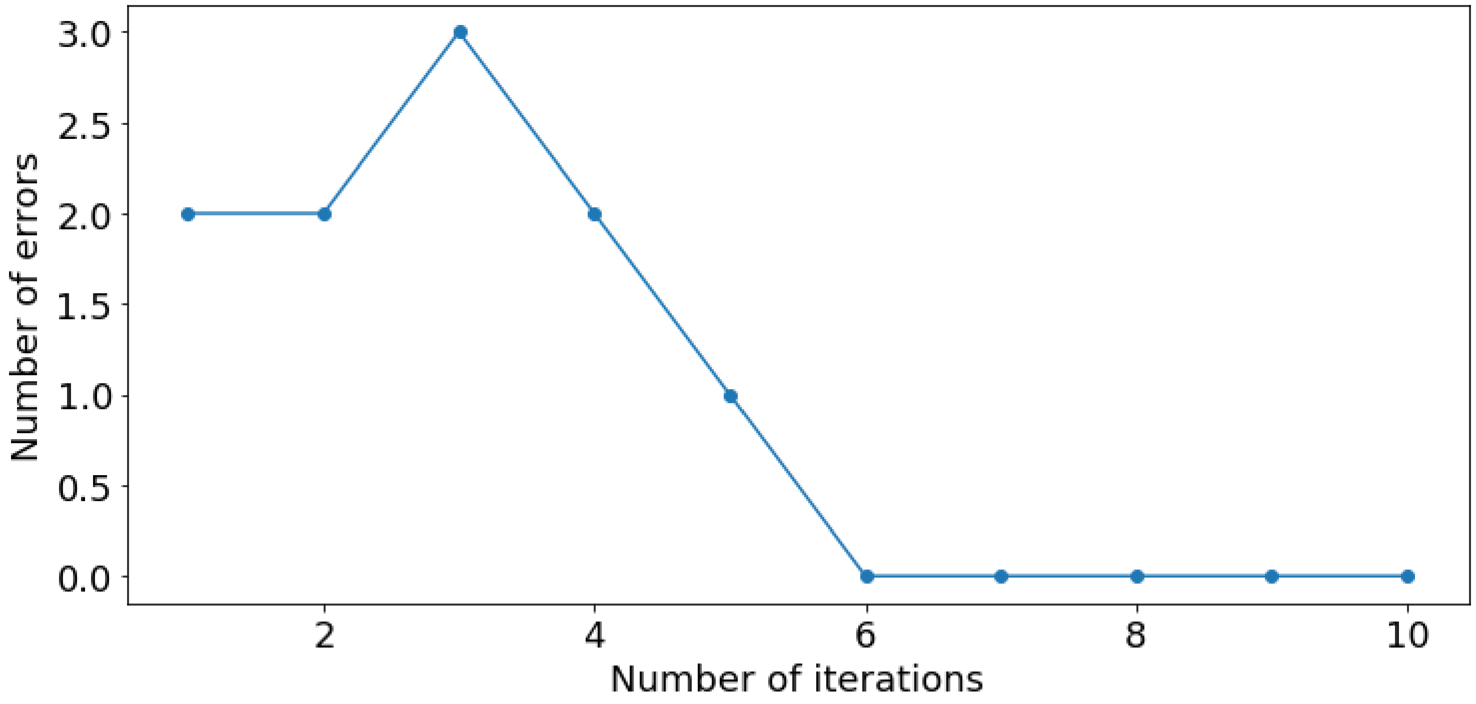


Working with the perceptron (4)

How does the model performance change??

In [39]:
```
f,ax = plt.subplots(figsize=(12, 5.5))
ax.plot(range(1, len(clf.errors_) + 1), clf.errors_, marker='o')
ax.set_xlabel('Number of iterations')
ax.set_ylabel('Number of errors')
```

Out[39]:
```
Text(0,0.5,'Number of errors')
```



Beyond the perceptron

Motivation

What might we change about the perceptron?

1. Change from updating errors that are binary to continuous
2. Use more than one observation a time for updating

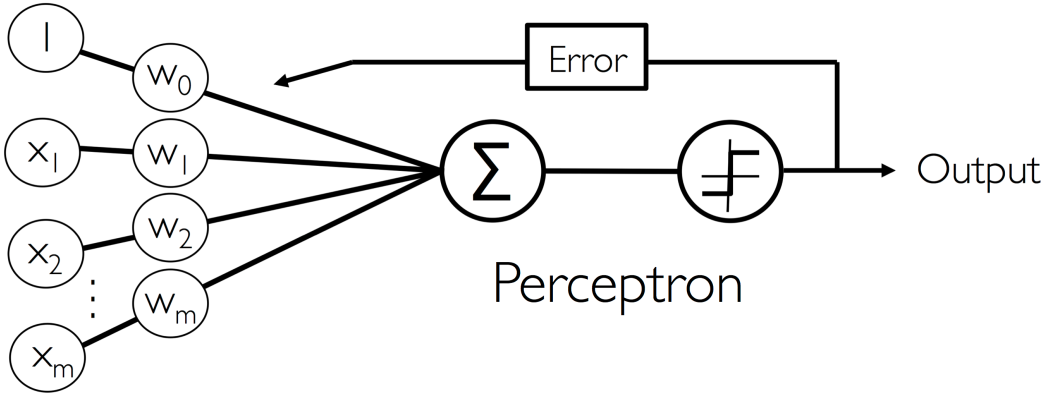
The activation function (1)

What else might we use to update errors?

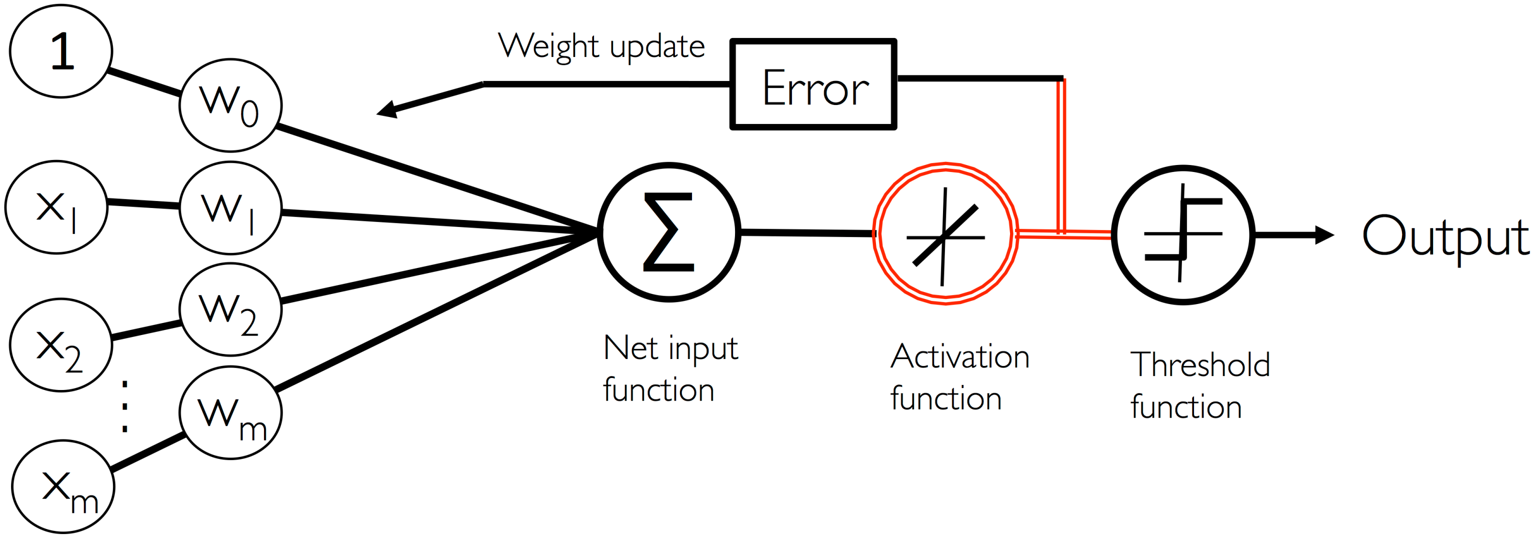
- The most simple is **no transformation** of the net-input, i.e. $\phi(z_i) = z_i$.
- When we change this from perceptron we call it Adaptive Linear Neuron (**Adaline**).

The activation function (2)

How is this different from the Perceptron?



Perceptron



Adaptive Linear Neuron (Adaline)

The activation function (3)

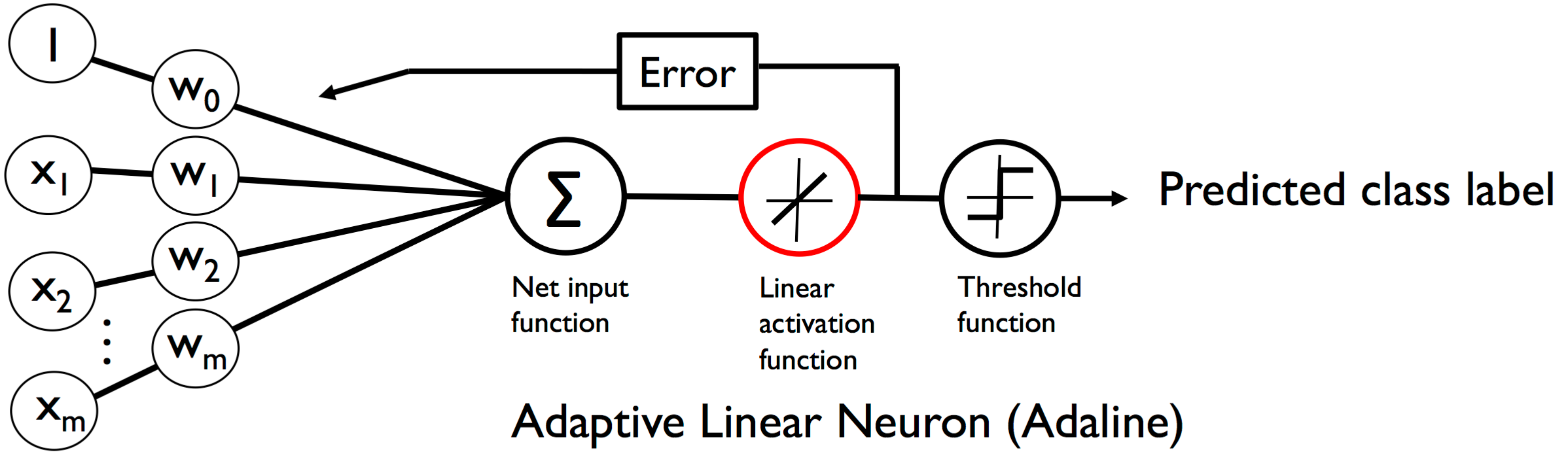
Which activation functions can be used?

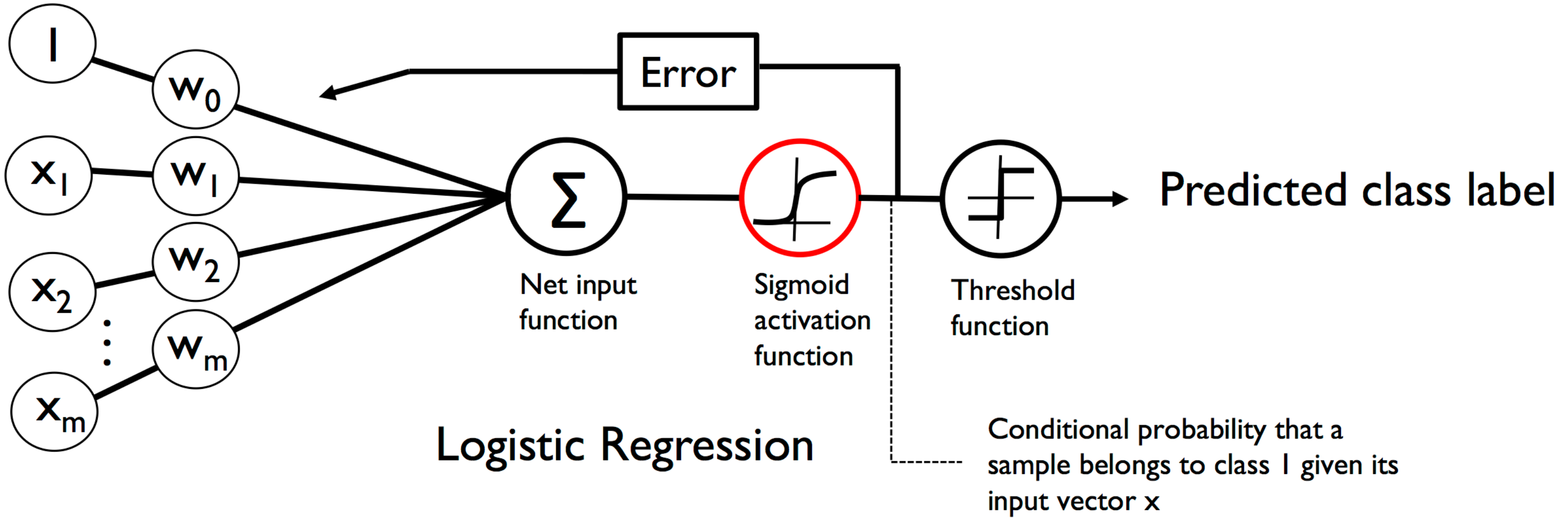
- Linear
- Logistic (Sigmoid)
- Unit step, sign

See page 450 in Python for Machine Learning.

The activation function (4)

How do Adaline and Logistic regression differ?


Error
Predicted class label
Net input function
Linear activation function
Threshold function
Adaptive Linear Neuron (Adaline)


Error
Predicted class label
Net input function
Sigmoid activation function
Threshold function
Logistic Regression
Conditional probability that a sample belongs to class 1 given its input vector x

A new objective (1)

The update rule in perceptron seems ad hoc, is there a more general way?

- Yes, we minimize the sum of squared errors (SSE). The SSE for Adaline is:

$$SSE = \boldsymbol{e}^T \boldsymbol{e} = e_1^2 + .. + e_n^2$$

$$\boldsymbol{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

Doesn't the above look strangely familiar?

- Yes, it is the same objective as OLS.
- But no, we will not solve like OLS.

A new objective (2)

So how the hell do we solve the model?

- We approximate the solution. Two options:
  - We approximate the first order derivative ~ gradient descent (GD)
  - We approximate both first and second order derivative ~ quasi Newton
- We take gradient descent - much simpler (sometimes faster)

A new objective (3)

What is the first order derivative of SSE in Adaline?

$$\frac{\partial SSE}{\partial \hat{w}} = \mathbf{X}^T\mathbf{e},$$

How do we update with GD in Adaline?

- Idea: take small steps to approximate the solution.
- $\Delta\hat{w} = \eta\mathbf{X}^T\mathbf{e} = \eta \cdot \mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}})$

A new objective (4)

The gradient descent algorithm we just learned uses the whole data.

- Often known as batch gradient descent.

What might be a smart way of changing (batch) gradient descent?

- we only use a subset of the data
- this called *stochastic gradient descent* (SGD)

Working with the logistic regression (1)

We load the titanic data and split into test and training sample

In [15]:
```
cols = ['survived','class', 'sex', 'sibsp', 'age', 'alone']

titanic = sns.load_dataset('titanic')
titanic_sub = pd.get_dummies(titanic[cols].dropna(), drop_first=True).astype(np.int64)


print(titanic_sub.head(2))

X = titanic_sub.drop('survived', axis=1)
y = titanic_sub.survived
```

```
   survived  sibsp  age  alone  class_Second  class_Third  sex_male
0         0      1   22      0             0            1         1
1         1      1   38      0             0            0         0
```

Working with the logistic regression (2)

In [16]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# we split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5, random_state=0)

# estimate model on train data, evaluate on test data
clf = LogisticRegression()
clf.fit(X_train, y_train) # model training
accuracy = (clf.predict(X_test)==y_test).mean() # model testing
print('Model accuracy is:', np.round(accuracy,3))
```

```
Model accuracy is: 0.79
```

Maximum margin classification

Motivation

Do the previous models care for how linear separation is done?

- No, as long as it classifies correctly then it is indifferent

Why is this a problem?

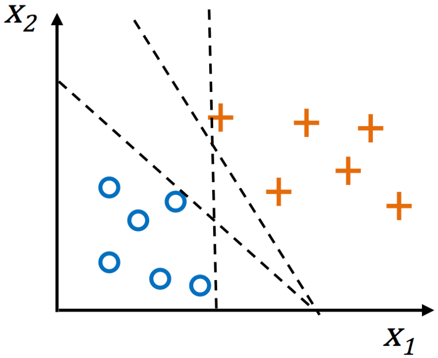
- We could optimize further on the boundary.

Maximum margin classification

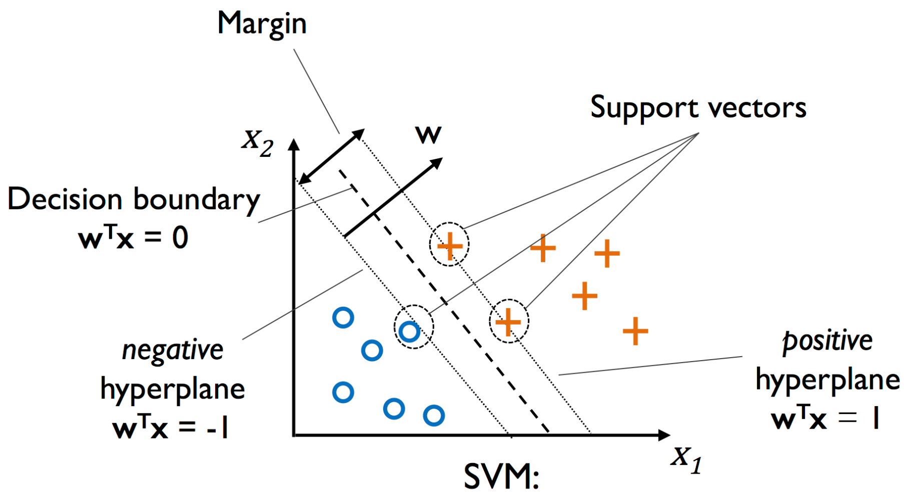
How might we improve the sepearation?

We use a Support Vector Machine (SVM) we get a solution.

SVM finds a decision boundary which maximize distance to nearest points:



Which hyperplane?

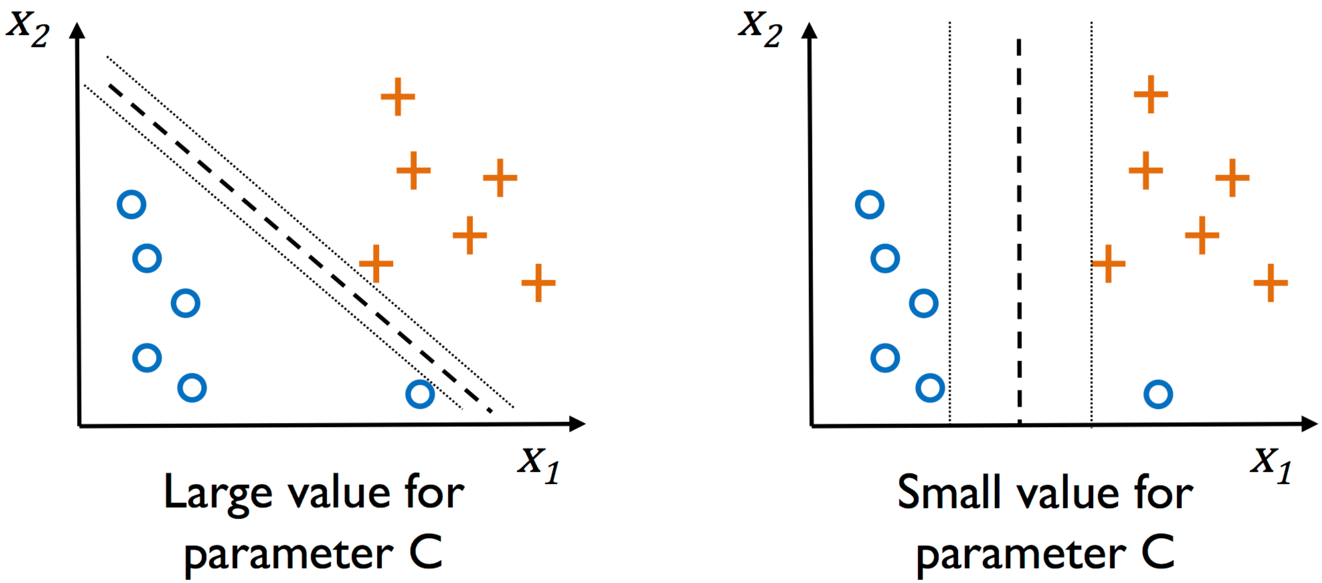


SVM:
Maximize the margin

Support vector machines

How might we improve SVM?

- We can use soft-margin classification. This extends the distance to boundary by ignoring a number of miss-classifications, likely outliers.



- SVM can also handle non-linearities using kernel methods.

The end

Return to agenda