Introduction to Machine Learning

Ramon Fuentes[1,2]

August 5, 2019

[1]Visiting Researcher, Dynamics Research Group
The University of Sheffield

[2]Research Scientist, Callsign Ltd

A little bit about me...

LEONARDO HELICOPTERS

The University Of Sheffield.

RICARDO

KONGSBERG

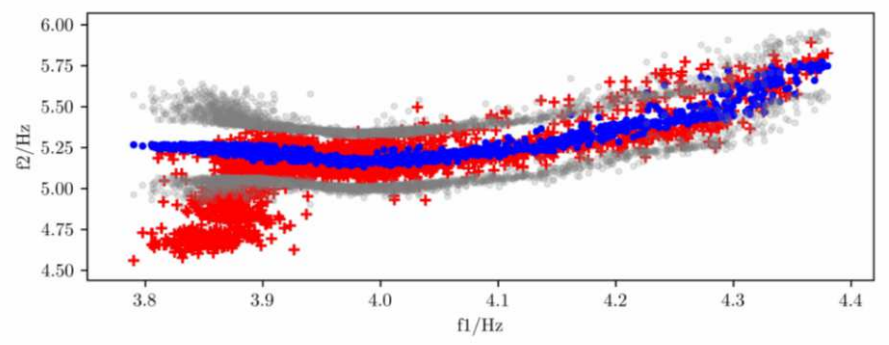
Callsign

Prenscia HBM

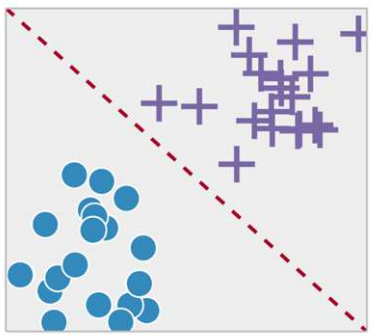
University of Strathclyde

The tools of machine learning

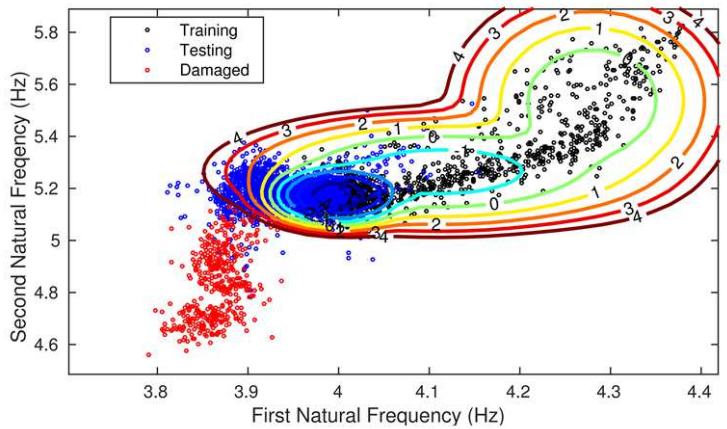
Regression



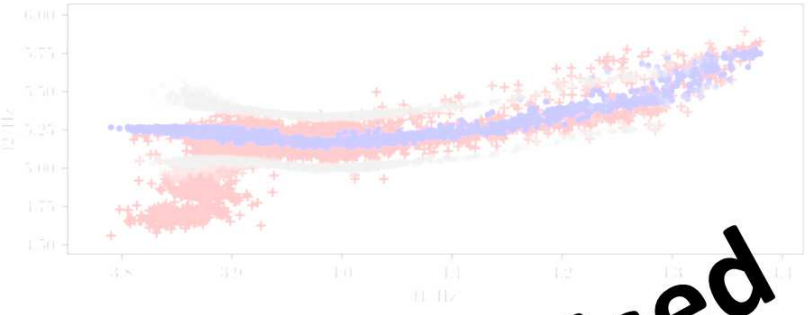
Classification



Density Estimation



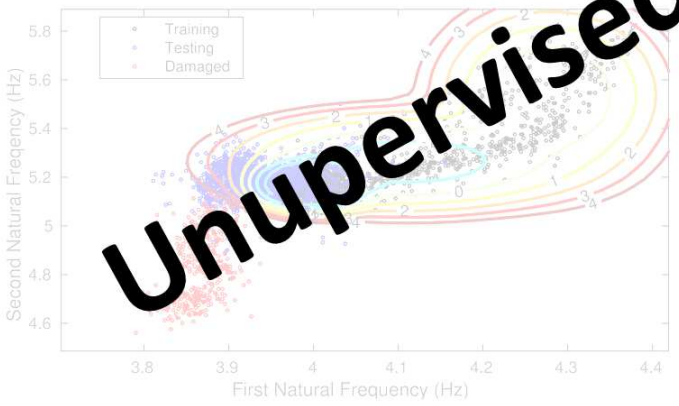
The tools of machine learning



- Classification

Supervised

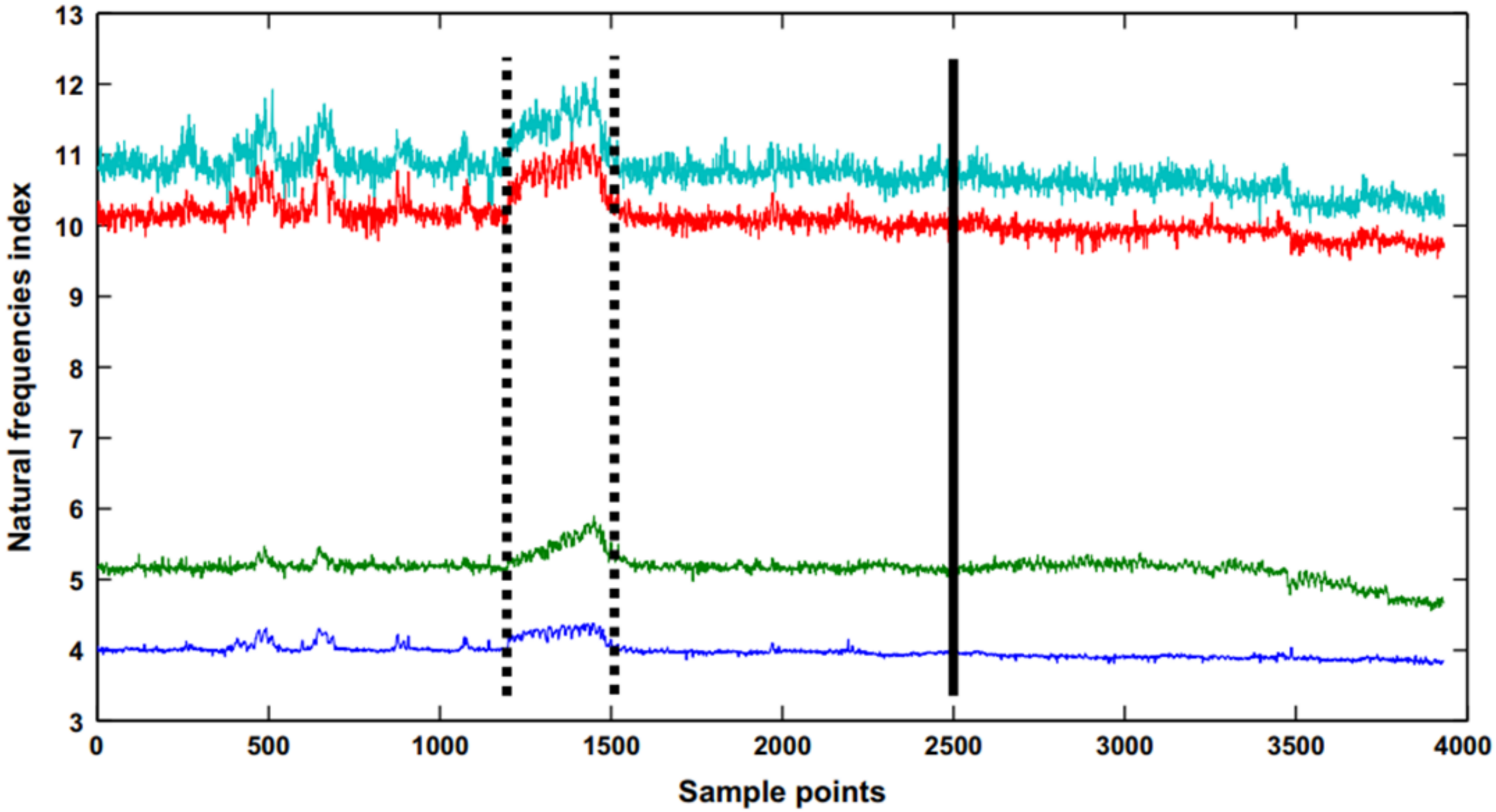
- Density Estimation



Unupervised

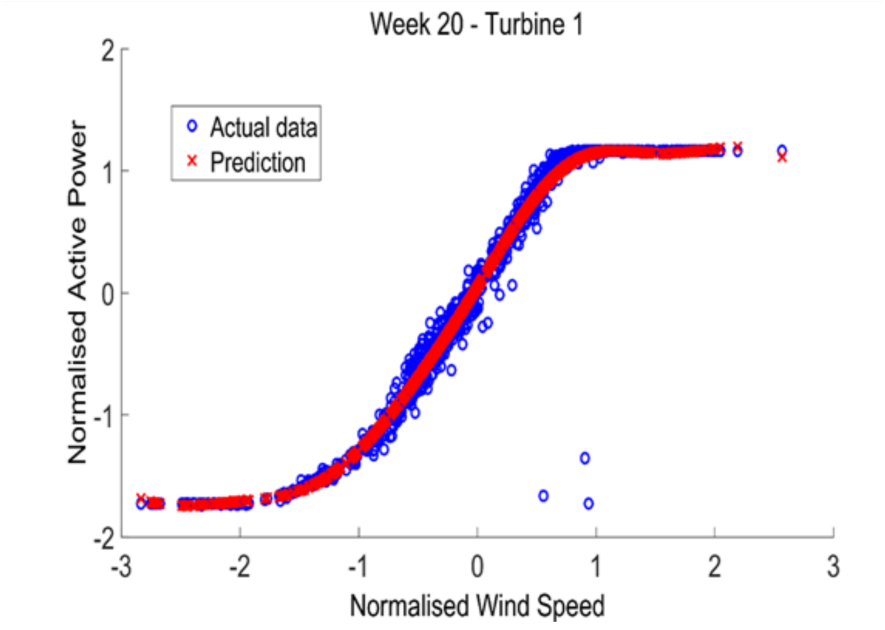
Motivating problem

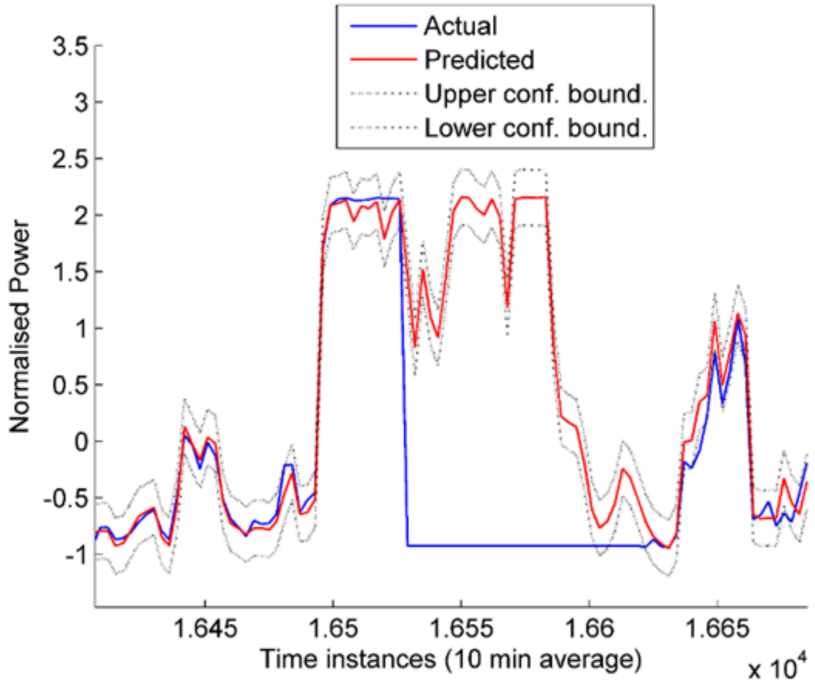
Given a set of measured natural frequencies from a bridge, can we detect damage?



Motivating problem

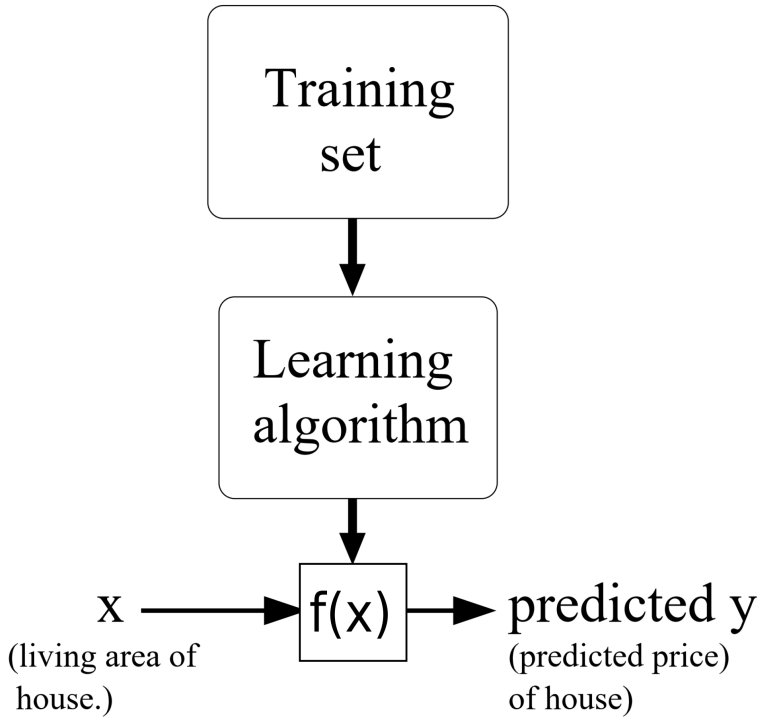
Can we diagnose problems on a wind turbine,
given measured wind and power?





Supervised learning

Supervised learning deals with the problem of modelling the relationship between a set of inputs, **x** and outputs **y**

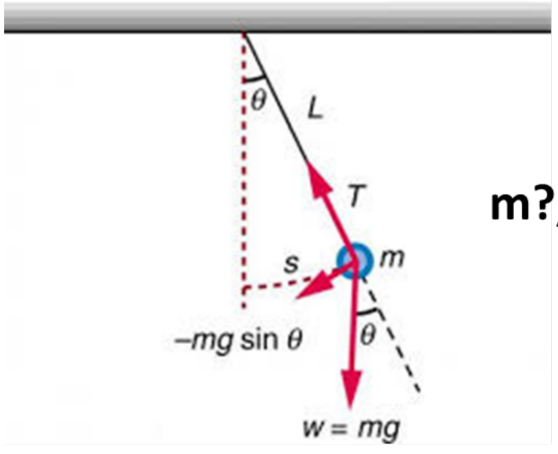


A first look at supervised learning: linear regression

Given some measured data,

there are generally two problems of interest:

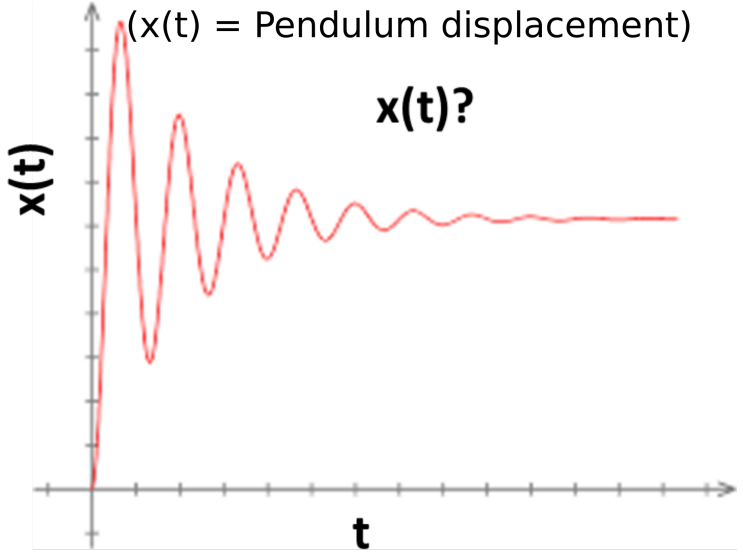
Identifying a system



m?, L?

Making predictions

(x(t) = Pendulum displacement)



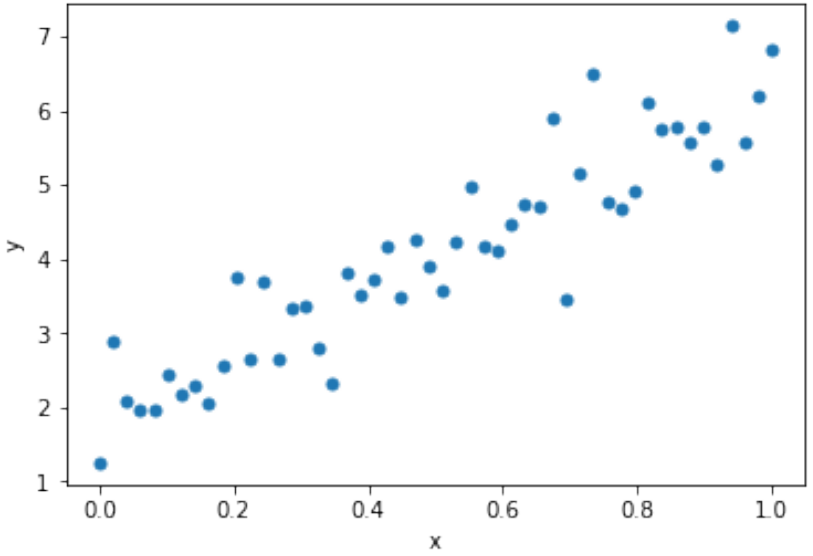
Linear Regression

Lets start with a simple toy example: a noisy $y = 5x + 1$

Can we learn the relationship between ***x*** and ***y*** from the data ?

We need to things:

1. A model
2. A loss function, that quantifies our error or predictive performance



Linear Regression

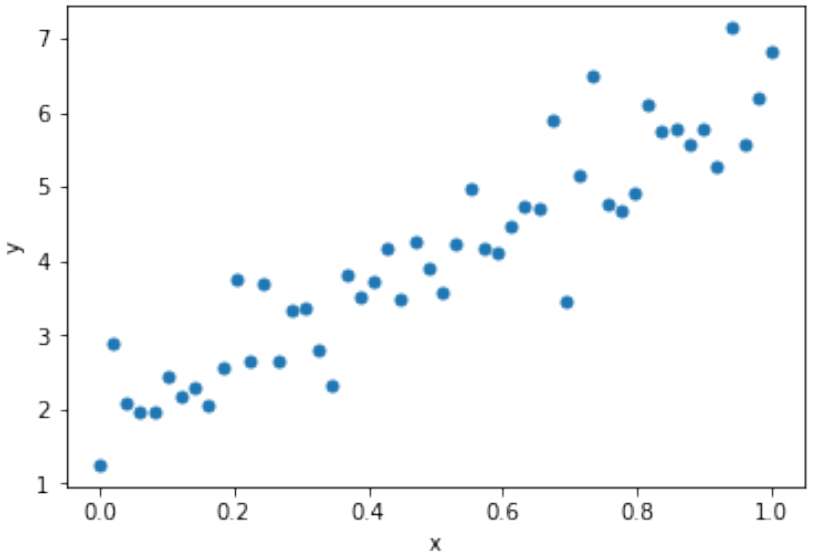
We need a model for the underlying function:

$$y = f(x)$$

Linear regression models $f(\mathbf{x})$ as:

$$y = x_0 w_0 + x_1 w_1 + x_2 w_2 + \ldots$$

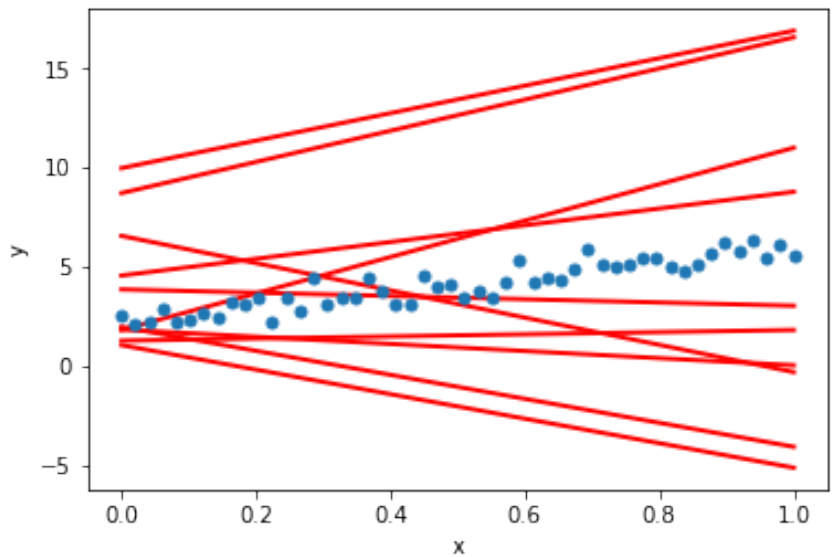
$$y = \sum_j x_j w_j$$



Linear Regression

So, we have a model, parametrised by $\mathbf{w}$, we now need to define a loss function so we can pick our weights appropriately

A bunch of models drawn at random, which fits the data best?

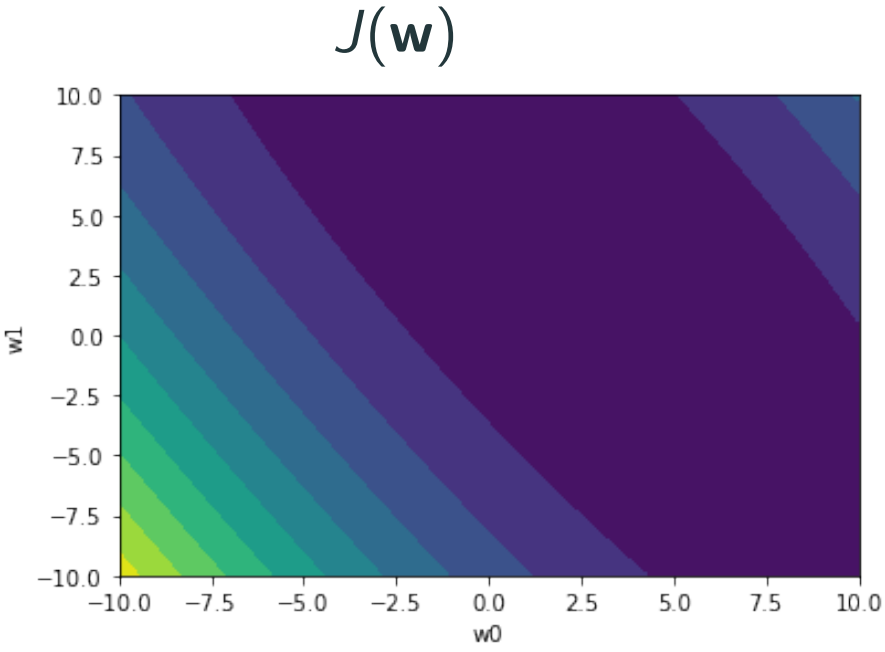


Squared error loss

One appropriate loss function is the mean of the squared prediction error:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - f(x^{(i)}))^2$$

But how should we find the value of $\mathbf{w}$ that minimises $J(\mathbf{w})$?



Gradient Descent

- There are many ways to *optimise* $\mathbf{w}$, but one efficient way of doing so is via **gradient descent**.

Gradient Descent

- There are many ways to *optimise* $\mathbf{w}$, but one efficient way of doing so is via **gradient descent**.
- The idea is that we'll start with an initial choice for $\mathbf{w}$ and improve it iteratively in a direction that decreases $J(\mathbf{w})$ - our cost function

Gradient Descent

- There are many ways to *optimise* $\mathbf{w}$, but one efficient way of doing so is via **gradient descent**.
- The idea is that we'll start with an initial choice for $\mathbf{w}$ and improve it iteratively in a direction that decreases $J(\mathbf{w})$ - our cost function
- We take a step in the direction of the gradient $\frac{\partial J}{\partial \mathbf{w}}$

Gradient Descent

This leads us to the gradient descent algorithm:

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w})$$

Gradient Descent

This leads us to the gradient descent algorithm:

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w})$$

And recall our loss function was:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

Gradient Descent

This leads us to the gradient descent algorithm:

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w})$$

And recall our loss function was:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

and remember,

$$f(\mathbf{x}^{(i)}) = \sum_{j}^{d} w_j x_j^{(i)}$$

Gradient Descent

Lets derive $\frac{\partial}{\partial w_j} J(\mathbf{w})$ for the case where we have a single training example $x_j^{(i)}$ $y^{(i)}$,

$$\begin{aligned}
\frac{\partial}{\partial w_j} J(\mathbf{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2} (y^{(i)} - f(\mathbf{x}^{(i)}))^2 \\
&= 2 \frac{1}{2} (y^{(i)} - f(\mathbf{x}^{(i)})) \frac{\partial}{\partial w_j} (y^{(i)} - f(\mathbf{x}^{(i)})) \\
&= (y^{(i)} - f(\mathbf{x}^{(i)})) \frac{\partial}{\partial w_j} \Big(\sum_{k=0}^{d} x_k^{(i)} w_k - y^{(i)} \Big) \\
&= (y^{(i)} - f(\mathbf{x}^{(i)})) x_j^{(i)}
\end{aligned}$$

Gradient Descent, learning rule

We now have an update rule, that we can apply whenever we encounter a new observation,

$$w_j = w_j + \alpha\big(y^{(i)} - f(\mathbf{x}^{(i)})\big)x_j^{(i)}$$

here, α is a *learning rate*

Batch Gradient Descent

When we have all training observations $x^{(1)}, ..., x^{(n)}$ and $y^{(i)}, ..., y^{(n)}$, we can assemble this into an algorithm

while not converged **do**
  for every j, **do**
    $w_j \leftarrow w_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - \sum_j^d x_j^{(i)} w_j\right) x_j$
  end for
end while

this is called *batch gradient descent*

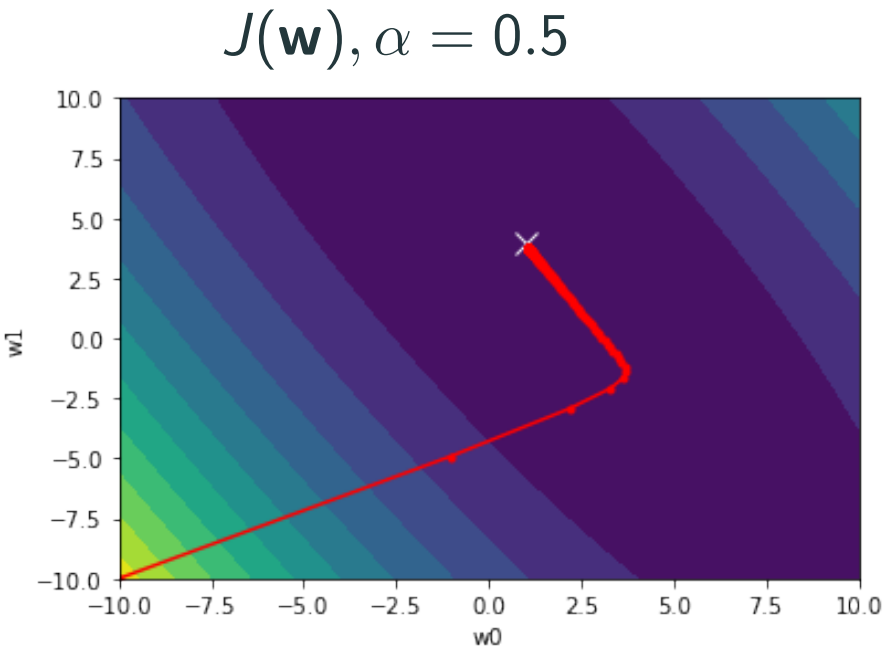
Back to our problem...

Applying 200 iterations of batch gradient descent to our toy problem, this is how our quadratic loss looks like

True function: $y = 5x + 1$

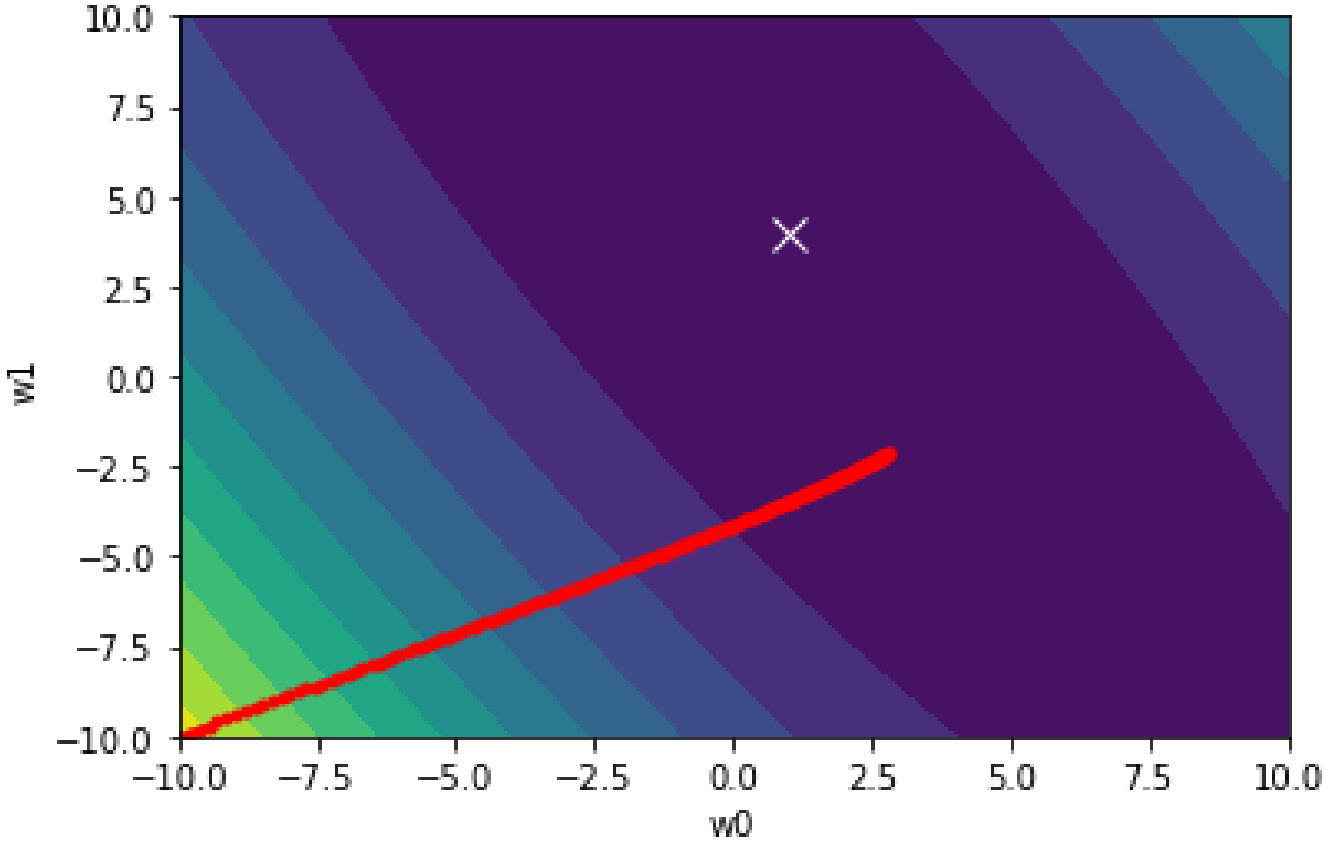
Estimated parameters:
$w_0 = 1.05, w_1 = 3.75$

close enough...



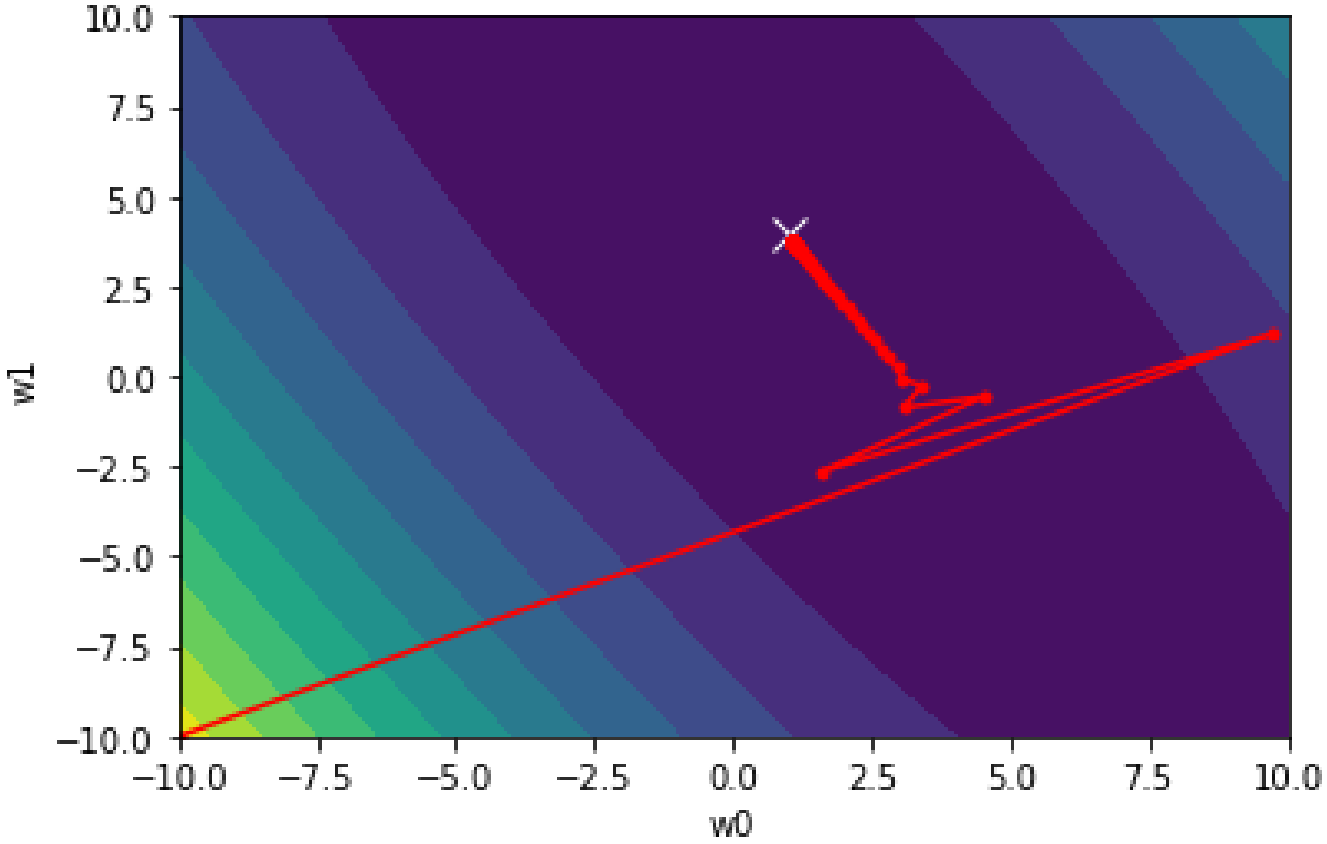
Back to our problem...

What happens if we choose a lower learning rate ($\alpha = 0.05$)?



Back to our problem...

What happens if we choose a higher learning rate($\alpha = 1.5$)?



Gradient Descent

Some notes on gradient descent:

- The learning rate has to be chosen wisely.

Gradient Descent

Some notes on gradient descent:

- The learning rate has to be chosen wisely.
- It is common to have a cooling rate - take large steps initially, and slow down as learning progresses. A bit of a heuristic

Gradient Descent

Some notes on gradient descent:

- The learning rate has to be chosen wisely.
- It is common to have a cooling rate - take large steps initially, and slow down as learning progresses. A bit of a heuristic
- It can get stuck in local optima. Not in this case though, as we have a well defined convex quadratic loss function.

Gradient Descent

Some notes on gradient descent:

- The learning rate has to be chosen wisely.
- It is common to have a cooling rate - take large steps initially, and slow down as learning progresses. A bit of a heuristic
- It can get stuck in local optima. Not in this case though, as we have a well defined convex quadratic loss function.
- For the basic linear regression problem, can we do better?

Gradient Descent

Some notes on gradient descent:

- The learning rate has to be chosen wisely.
- It is common to have a cooling rate - take large steps initially, and slow down as learning progresses. A bit of a heuristic
- It can get stuck in local optima. Not in this case though, as we have a well defined convex quadratic loss function.
- For the basic linear regression problem, can we do better?
- yes!

Analytical solution

- There is in fact a closed form analytical solution that optimises $J(\mathbf{w})$.

Analytical solution

- There is in fact a closed form analytical solution that optimises $J(\mathbf{w})$.
- But first, lets re-write our problem in matrix notation

Analytical solution

- There is in fact a closed form analytical solution that optimises $J(\mathbf{w})$.
- But first, lets re-write our problem in matrix notation
- Our output is represented by a (column) vector: $\mathbf{y} = [y_1, ...y_n]$

Analytical solution

- There is in fact a closed form analytical solution that optimises $J(\mathbf{w})$.
- But first, lets re-write our problem in matrix notation
- Our output is represented by a (column) vector: $\mathbf{y} = [y_1, ...y_n]$
- And our inputs are assembled into a matrix, with each variable represented by a column $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_m]$,

Analytical solution

- There is in fact a closed form analytical solution that optimises $J(\mathbf{w})$.
- But first, lets re-write our problem in matrix notation
- Our output is represented by a (column) vector: $\mathbf{y} = [y_1, ...y_n]$
- And our inputs are assembled into a matrix, with each variable represented by a column $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_m]$,
- where each column, $\mathbf{x}_j = [x_j^{(1)}, ..., x_j^{(n)}]$

Analytical solution

Our linear regression problem is now:

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

Analytical solution

Our linear regression problem is now:

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

Our loss is:

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \sum_{j}^{d} x_j^{(i)} w_j)^2 \\ &= \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \end{aligned}$$

Analytical solution

The gradient of the loss is

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Analytical solution

The gradient of the loss is

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} \frac{1}{2} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

which (applying a few matrix identities...) leads to,

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{X}^T \mathbf{y}$$

Analytical solution

The gradient of the loss is

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

which (applying a few matrix identities...) leads to,

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}$$

Setting $\nabla_{\mathbf{w}} J(\mathbf{w})$ to zero, leads to the normal equations,

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

Analytical solution

The gradient of the loss is

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

which (applying a few matrix identities...) leads to,

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}$$

Setting $\nabla_{\mathbf{w}} J(\mathbf{w})$ to zero, leads to the normal equations,

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

and solving for $\mathbf{w}$, gives us

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Back to our problem again

Using the normal equation,

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Back to our problem again

Using the normal equation,

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$w_0 = 1.05, w_1 = 3.75$

Back to our problem again

Using the normal equation,

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$w_0 = 1.05, w_1 = 3.75$

Similar solution as gradient descent!

Back to our problem again

Using the normal equation,

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$w_0 = 1.05, w_1 = 3.75$

Similar solution as gradient descent!

but rather easier ?

Recap

We've looked at two ways of optimising the parameters/weights in a linear regression setting:

- through gradient descent, leading to the batch gradient descent algorithm

Recap

We've looked at two ways of optimising the parameters/weights in a linear regression setting:

- through gradient descent, leading to the batch gradient descent algorithm
- analytically, through the matrix normal equations

Recap

We've looked at two ways of optimising the parameters/weights in a linear regression setting:

- through gradient descent, leading to the batch gradient descent algorithm
- analytically, through the matrix normal equations
- both of these useful on their own, and key ingredients when solving more complex problems, as we'll see...

Conclusions

What have we learned today?

- Basics of supervised vs. unsupervised learning
- Linear models for regression
- The least means squares algorithm
- Analytical solution to the unconstrained linear regression problem

So... what next?

Tomorrow, we'll learn how to deal with more complex nonlinear problems, using the tools we've worked through today ;)