COMP47750/COMP47990

Regression

Pádraig Cunningham
Based on slides by Aonghus Lawlor & Derek Greene

School of Computer Science

Regression - Overview

- Regression
  - Simple Linear Regression
    - Analytical Solution
- Gradient Descent
  - Multiple Linear Regression
- Logistic Regression
  - Regression for categorical features

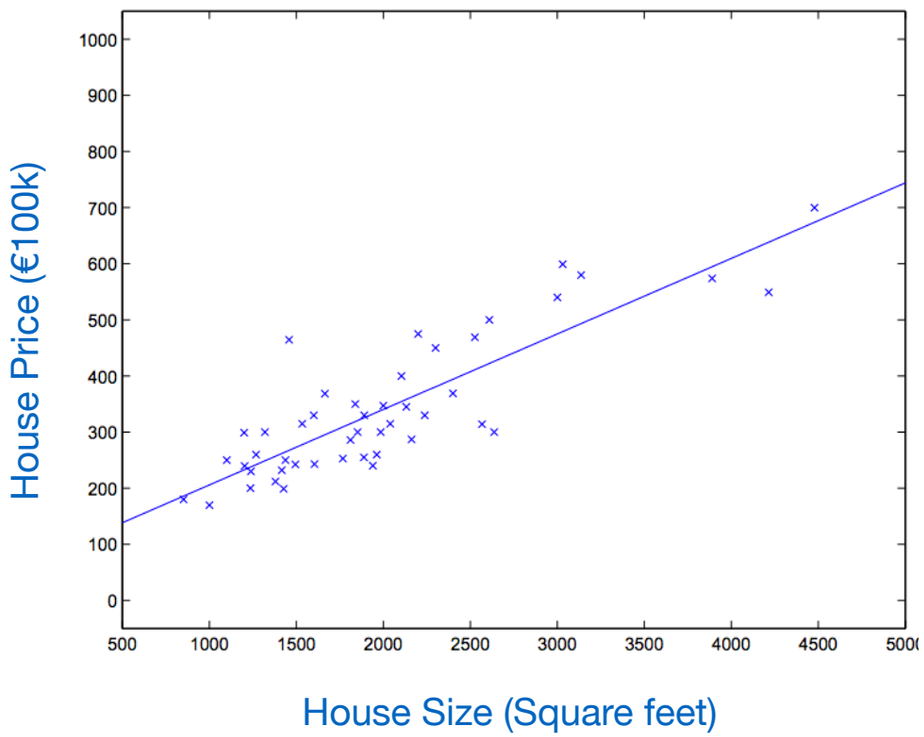
Regression

- Regression is a well-known and widely-applied method in statistics
- Linear dependence: constant rate of increase of one variable with respect to another
- Goal: minimise the error in a model of the data and make the most accurate predictions

Classification: predict a value belonging to a class

Regression: predict a value belonging to a continuous set

Predict the price of a house, given the size of the house.

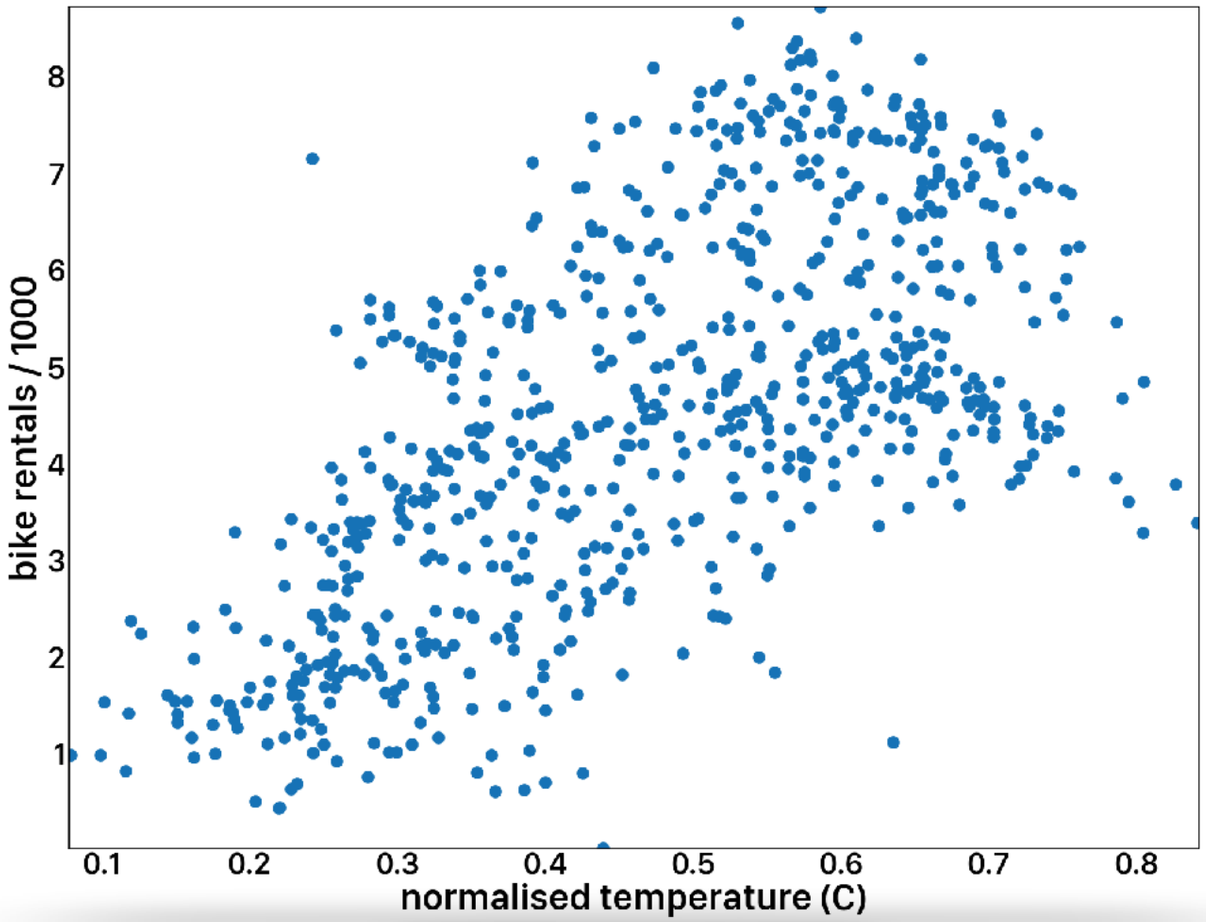


Regression Terminology

- Regression has a long history and there are many overlapping terms and names in common use.
- Linear regression: We assume the output variable Y models a linear relationship between the input variables X_1, X_2, ...
- Simple Linear Regression: There is just one input variable X. Changes is Y are caused by changes in X.
- Multiple Linear Regression: There is more than one input variable.
- One of the simplest ways to train the model is with Ordinary Least Squares (OLS), commonly called Least Squares Regression.

Regression Example

- Example: Bike Sharing Data
- We would like to know how the number of bike rentals per day depends on the temperature
- From the data it looks like there are more rentals on warmer days (perhaps expected?)
- But what is the relationship between the temperature and rentals?



Regression Example

- The input variable X is the temperature
- The output variable Y is the number of bike rentals
- Our linear model is:

bike rentals = 945 + 7501 x temperature

General form

$$Y = \beta_0 + \beta_1 \times X$$

➡ For every 1 unit increase in the temperature, we would expect the bike rentals to increase by 7501

remember X is normalised

Regression Notation

X independent variable(s)

Y dependent variable(s)

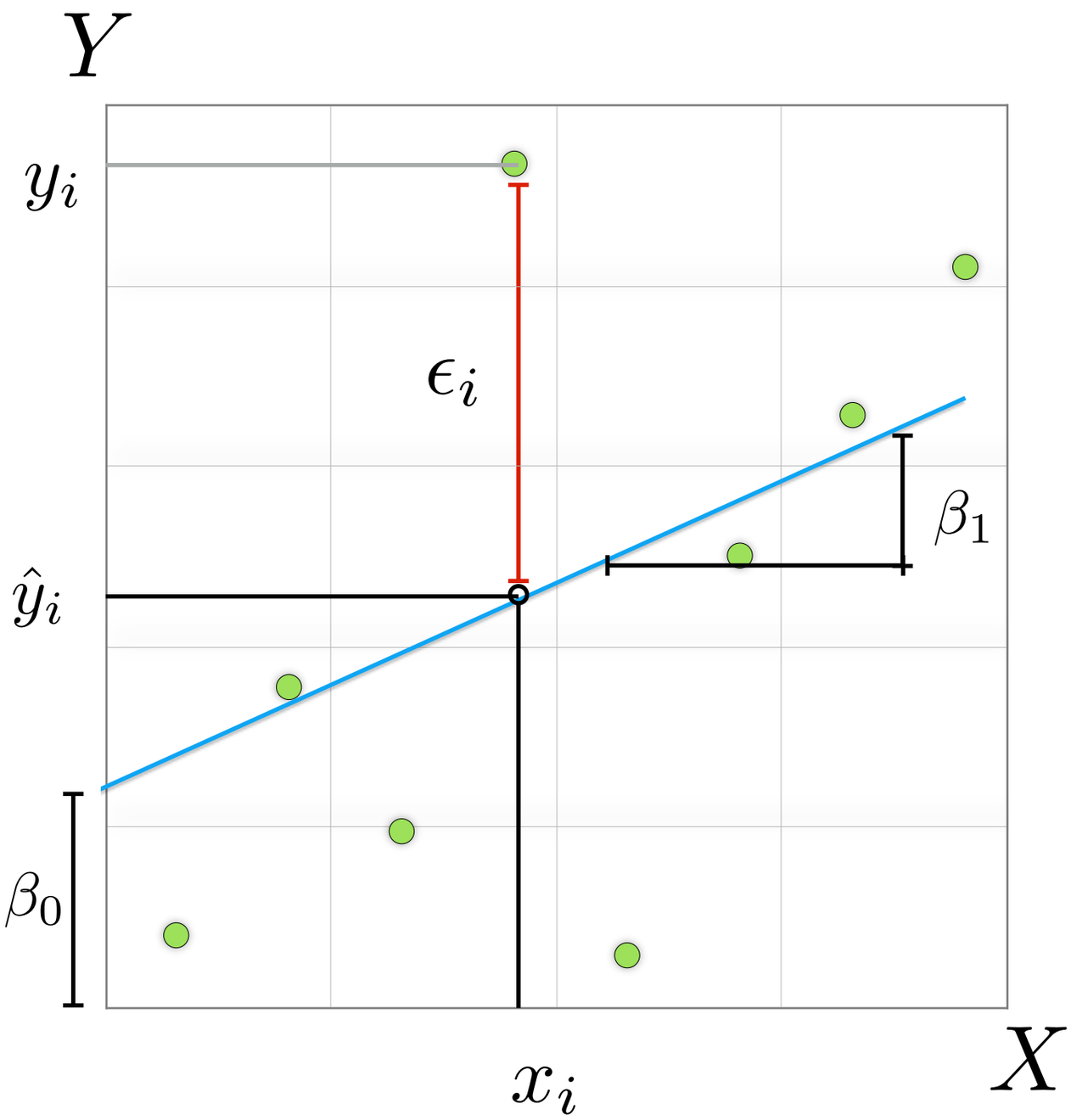
y_i observed value of Y

$\hat{y}_i$ predicted value of Y

β_0 intercept

β_1 slope

ϵ_i error



Simple Linear Regression

- **Key idea:** We assume that the dependent variable Y is a linear function of X.
- The parameters of the model are β_0, β_1:
  - β_0: estimated average value of Y when X is 0 (i.e. intercept).
  - β_1: estimated change in the average value of Y, if we make a unit change in X (i.e. slope).

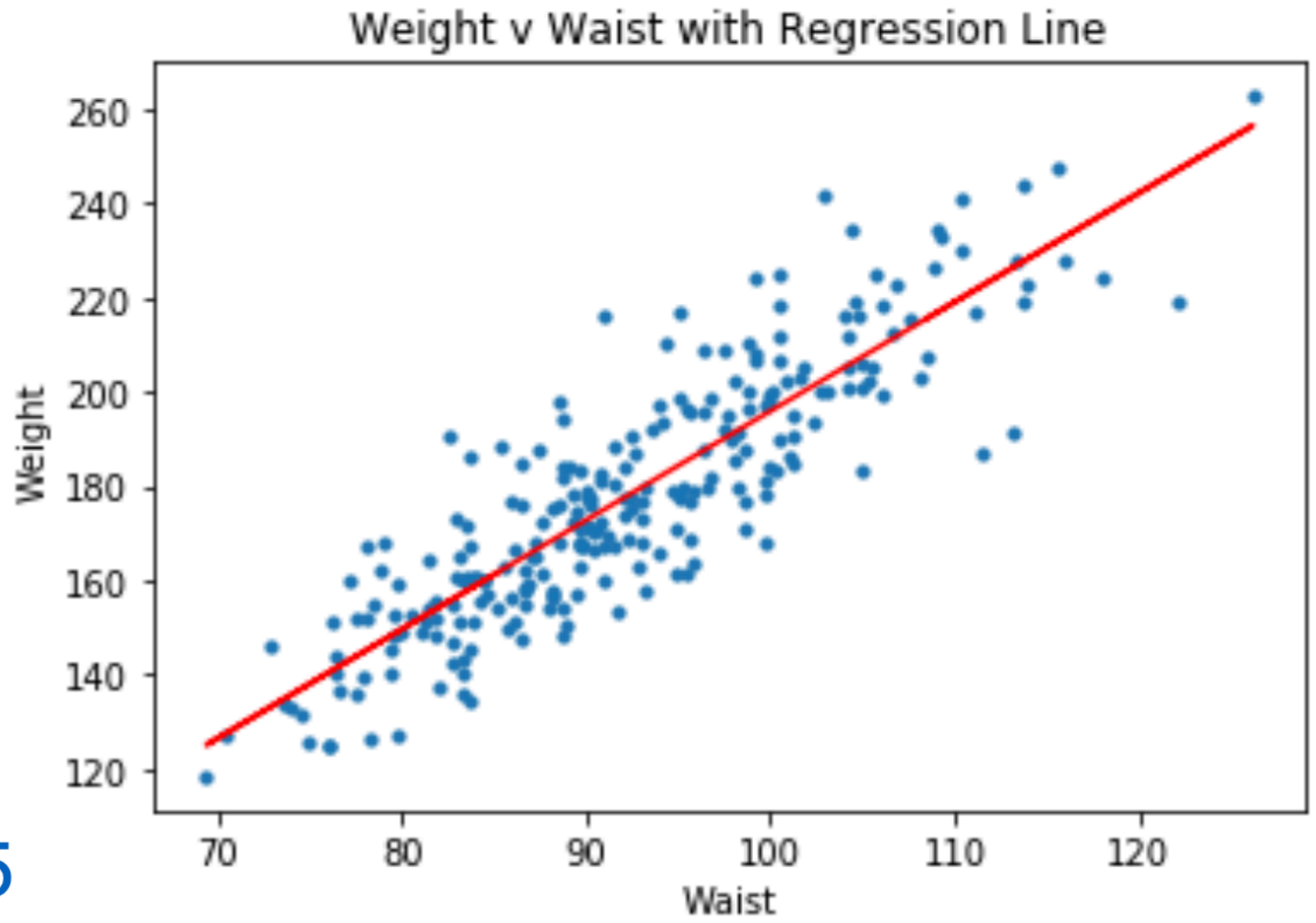
Simple linear regression model

$$Y = \beta_0 + \beta_1 \times X$$

Python Code notebook 12 Regression

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(X, y)
print(' R squared statistic: {:.2f}'.format(reg.score(X, y)))
print(' Slope: {:.2f}'.format(reg.coef_[0]))
print(' Intercept: {:.2f}'.format(reg.intercept_))
```

```
R squared statistic: 0.76
Slope: 2.31
Intercept: -35.45
```



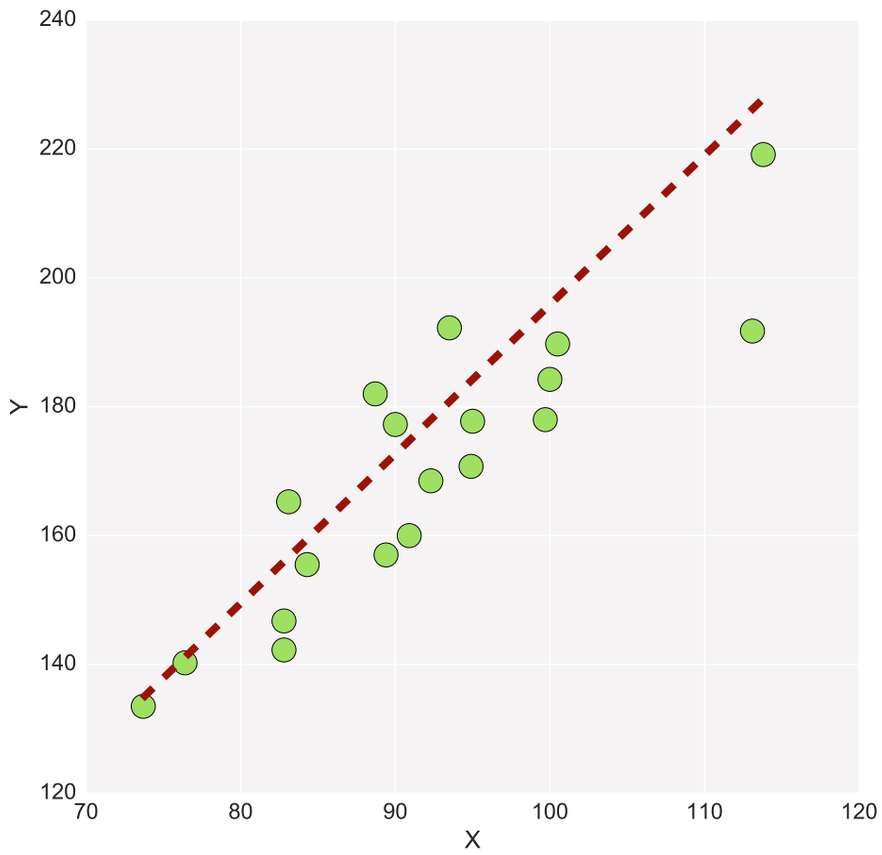
So the linear model is:

Weight = 2.31 x Waist - 35.45

Simple Linear Regression

- We want to estimate the parameters β_0, β_1
- We want to find the best fit or representation of the points
- But how do we find the best possible representation?

Key idea:
Draw a line through the data and use a measure which gives us information on how well that line represents the data.



Simple Linear Regression

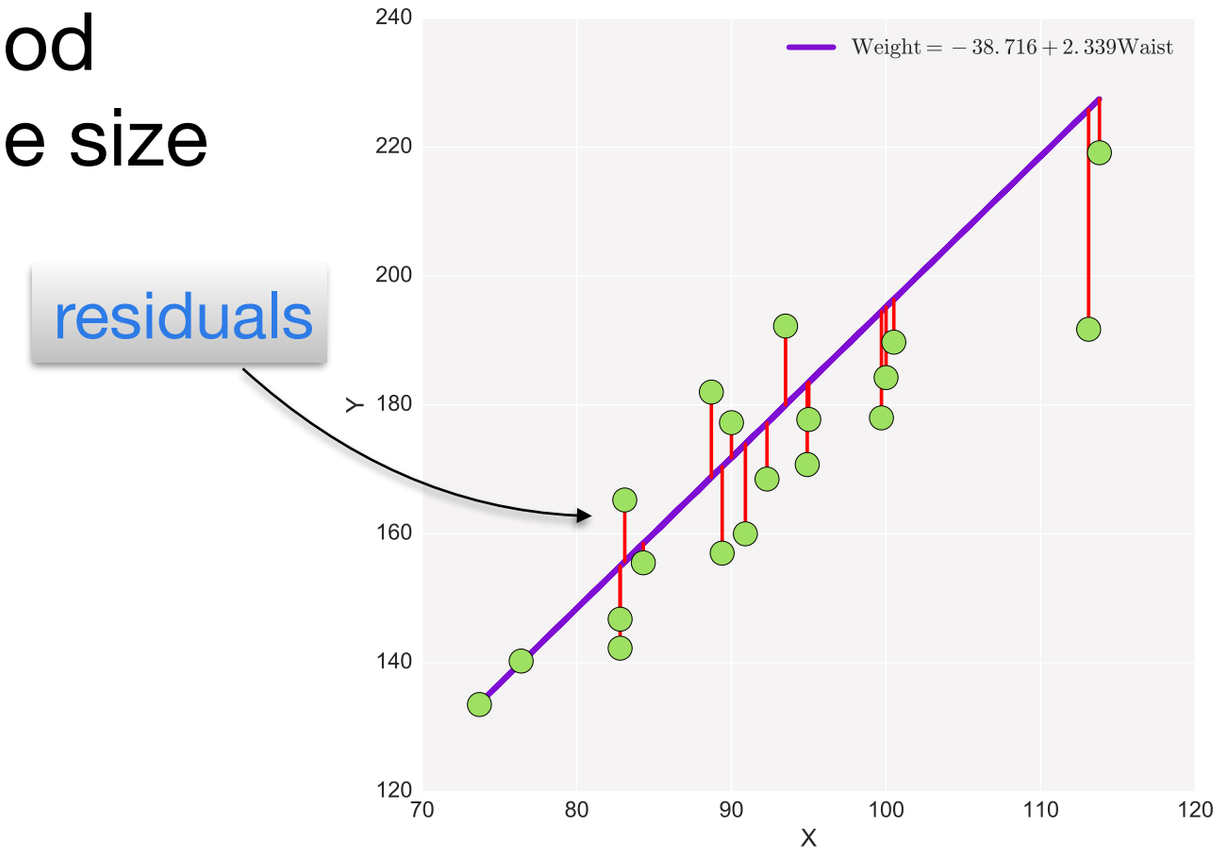
- Some lines are a better fit than others. The difference between the data and the line is the residual.
- We adjust the slope of the line until the residual above the line are equal to residuals below the line
- Use the least squares method to minimise the error (i.e. the size of the residuals).

$$\epsilon_i = y_i - \beta_0 - \beta_1 x_i$$

Residual for the *i*-th example

$$\sum_i^n \epsilon_i^2 = \sum_i^n (y_i - \beta_0 - \beta_1 x_i)^2$$

Sum of all squared residuals for *n* examples



sum(squared error) = 2995.6991

Simple Linear Regression

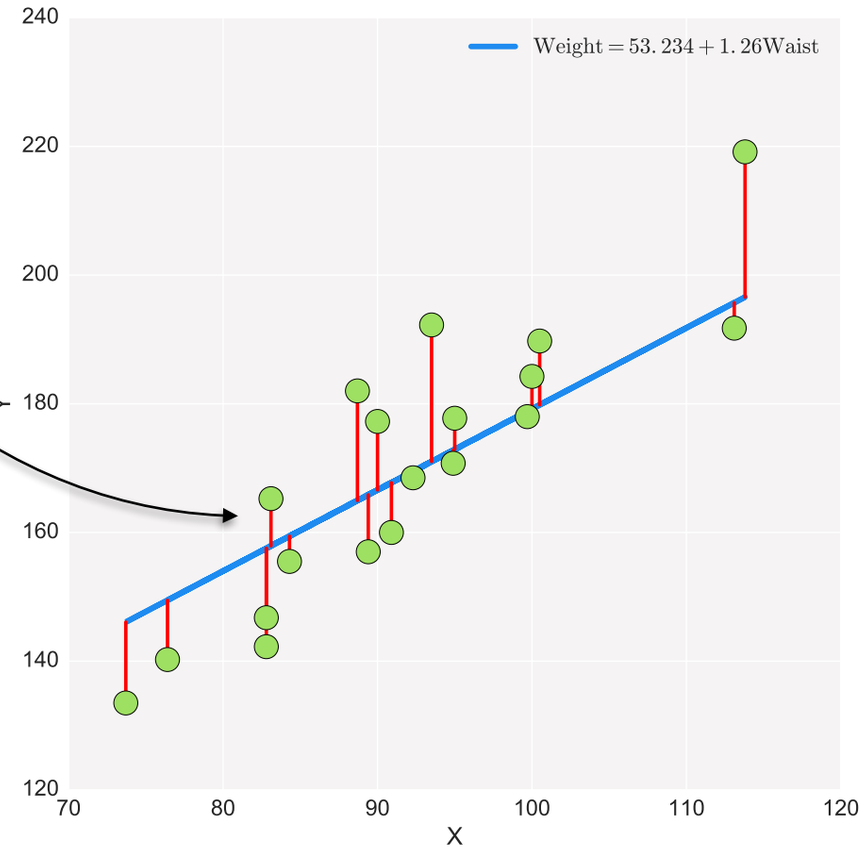
- Some lines are a better fit than others. The difference between the data and the line is the residual.
- We adjust the slope of the line until the residual above the line are equal to residuals below the line
- Use the least squares method to minimise the error (i.e. the size of the residuals).

$$\epsilon_i = y_i - \beta_0 - \beta_1 x_i$$

Residual for the *i*-th example

$$\sum_i^n \epsilon_i^2 = \sum_i^n (y_i - \beta_0 - \beta_1 x_i)^2$$

Sum of all squared residuals for *n* examples



sum(squared error) = 2333.0325

Regression Coefficients

- The best fit values for the parameters are found by minimising the sum of squared errors and solving for β_0, β_1.
- First compute:
  - the variance of x (S_{xx}) and the covariance of x and y (S_{xy})

$$S_{xx} = \sum_i^n (x_i - \bar{x})^2 \qquad S_{xy} = \sum_i^n (y_i - \bar{y})(x_i - \bar{x})$$

$$= \sum_i^n x_i y_i - \frac{(\sum_i^n x_i)(\sum_i^n y_i)}{n}$$

- The best fit parameters are:

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}}$$

Fitted linear regression model

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$$

Regression Coefficients

- In practice, we calculate:

$$\hat{\beta}_1 = \frac{\sum X_i Y_i - \frac{(\sum X_i)(\sum Y_i)}{n}}{\sum X_i^2 - \frac{(\sum X_i)^2}{n}}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$$

- **Example:**

temperature	count
0.47583	6606
0.18696	1550
0.33083	3747
0.42583	6041
0.55000	7538
0.71667	7264
0.13478	1605
0.37333	2209
0.73167	7499
0.72250	5743

$$\hat{\beta}_1 = \frac{27274.1012 - 23149.9915}{2.5831 - 2.1607} = 9763.8694$$

$$\hat{\beta}_0 = 4980 - \hat{\beta}_1 \times 0.4648 = 441.5571$$

$$\hat{Y} = 441.5571 + 9763.8694 \times X$$

For temperature=0.5, we predict the count to be:

441.5571 + 9763.8694 x 0.5 = 5323.492

Sources of Error

- What are the sources of error in a regression model?

$$SST = \sum_{i=1}^{n} (Y_i - \bar{Y})^2$$

total sum of squared deviations in Y from its mean

Total sum of squares

$$SSR = \sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2$$

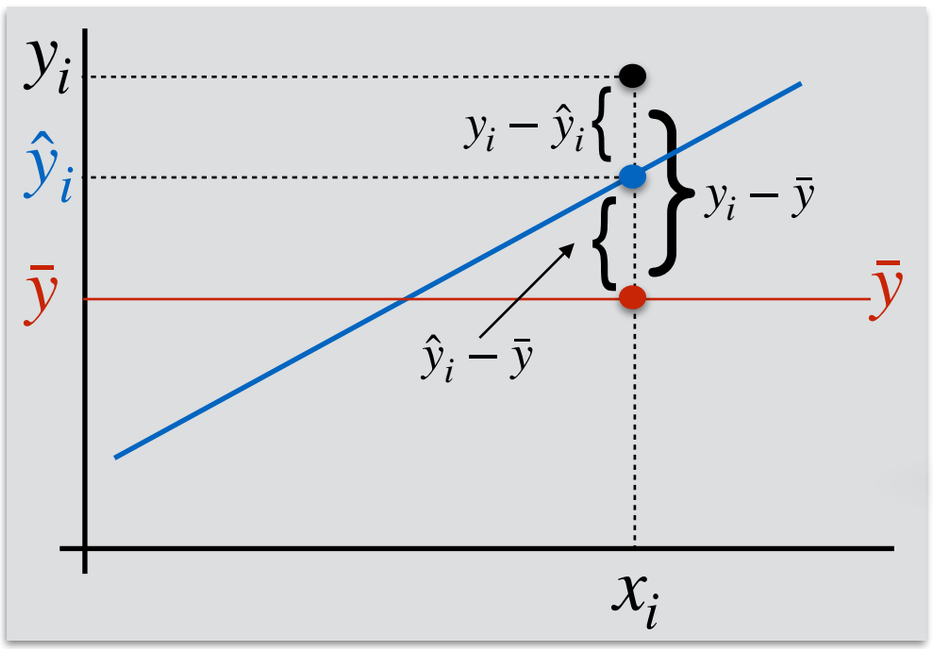
total sum of squared deviation of the best fit from the mean

Variation in Y explained by the regression line

$$SSE = \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2$$

total sum of squared residuals (difference between fit and the observed data)

Variation in Y that is left unexplained



$$SST = SSR + SSE$$

overall variability in Y = regression model + error

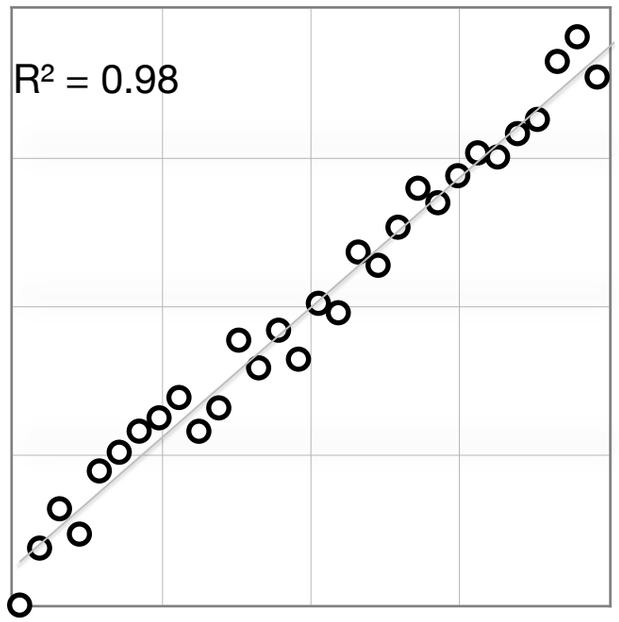
explained by model

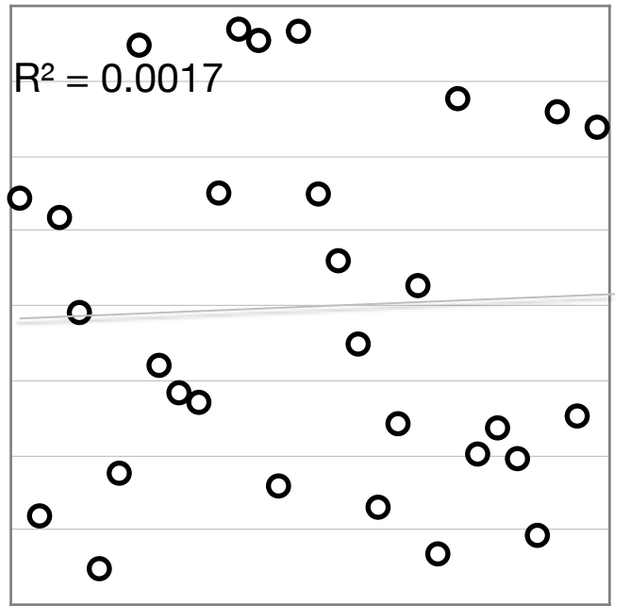
remains unexplained

Correlation

- When we want to measure the strength of the linear relationship, we use the coefficient of determination R^2.
- R^2 is the fraction of the variation which is explained by the linear regression model.
- It measures the explanatory power of the model, and tells us how well our regression line matches the real data. The closer R^2 is to 1, the better the fit.

$$R^2 = \frac{SSR}{SST}$$
$$= 1 - \frac{SSE}{SST}$$





Does not tell us if X is the cause of changes in Y

Error measures for Regression

- R^2: Variance 'explained' by the model

$$R^2(y, \hat{y}) = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

- MAE: mean absolute error

$$MAE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n} |y_i - \hat{y}_i|$$

- MAPE: mean absolute percentage error

$$MAPE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n} \frac{|y_i - \hat{y}_i|}{\max(\epsilon, |y_i|)}$$

ϵ prevents div by 0

- MSE / RMSE: (root) mean squared error

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n} (y_i - \hat{y}_i)^2 \qquad RMSE(y, \hat{y}) = \sqrt{MSE(y, \hat{y})}$$

COMP47750/COMP47990

Regression

Part II
Gradient Descent
Regularisation
Regularisation

Pádraig Cunningham
Based on slides by Aonghus Lawlor & Derek Greene

School of Computer Science

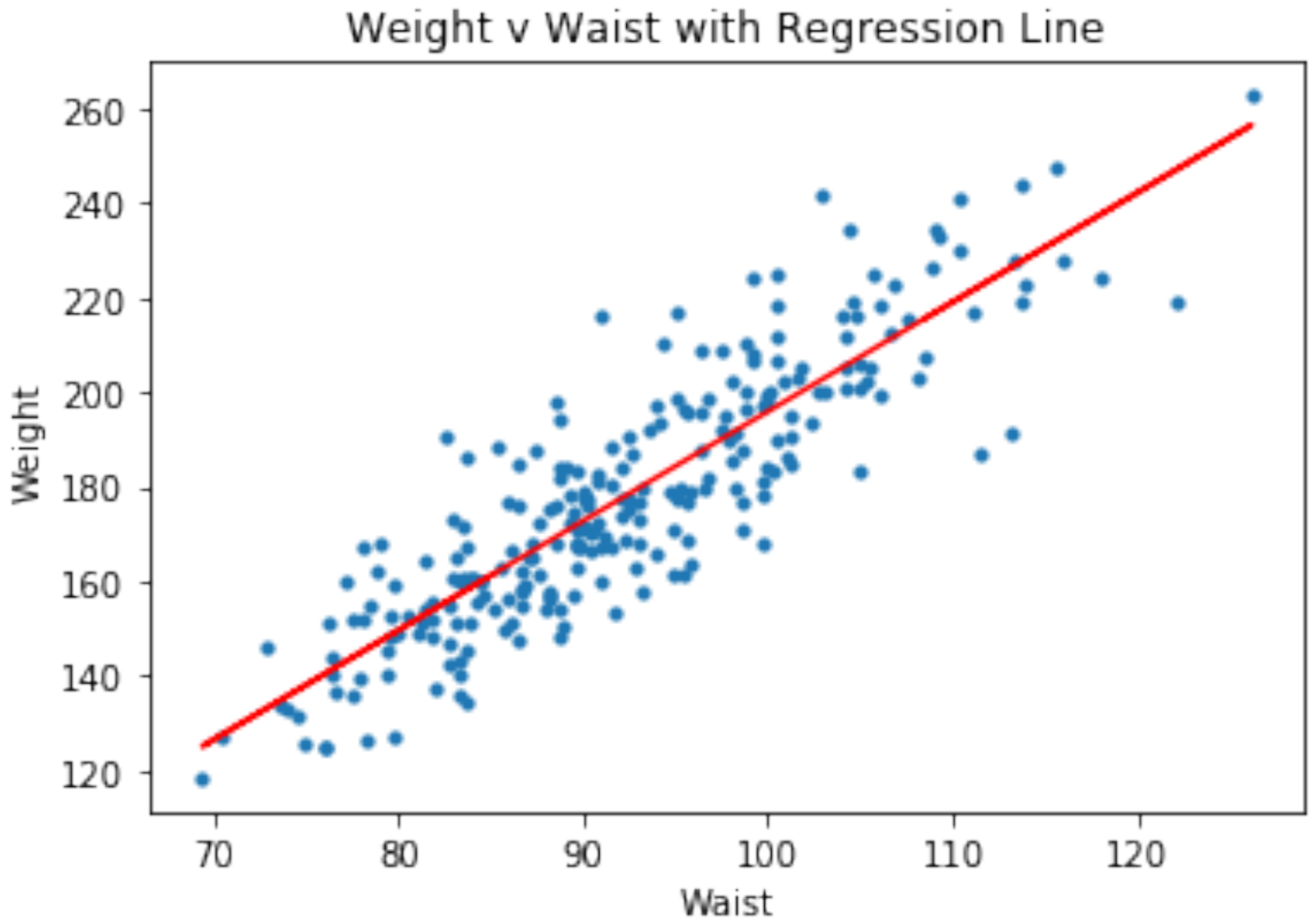
Regression - Overview

- Regression
  - Simple Linear Regression
    - Analytical Solution
- Gradient Descent
  - Multiple Linear Regression
- Logistic Regression
  - Regression for categorical features

Python code notebook 12 Regression

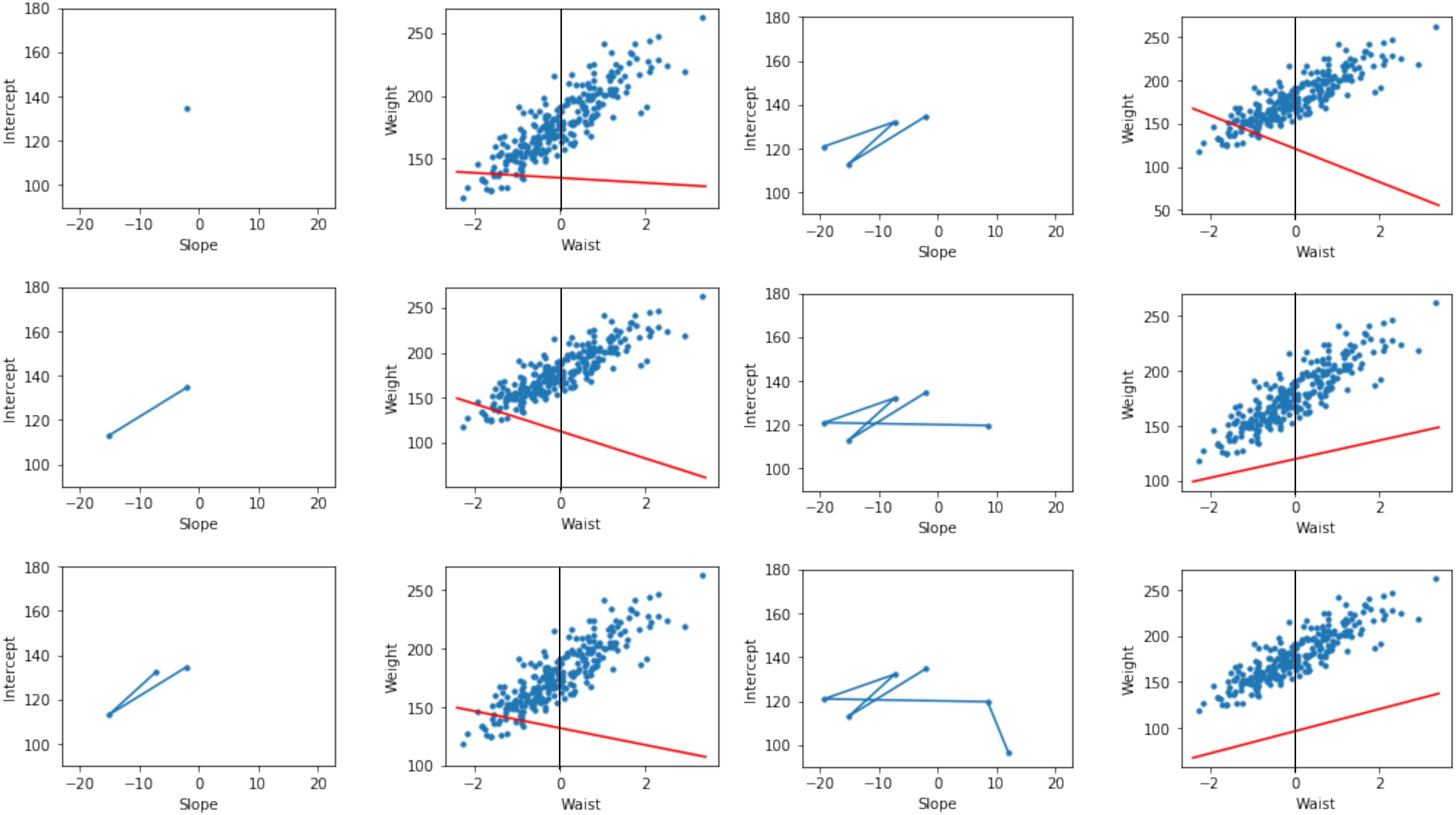
```
Slope: 2.31
Intercept: -35.45
```

Parameters for data before scaling



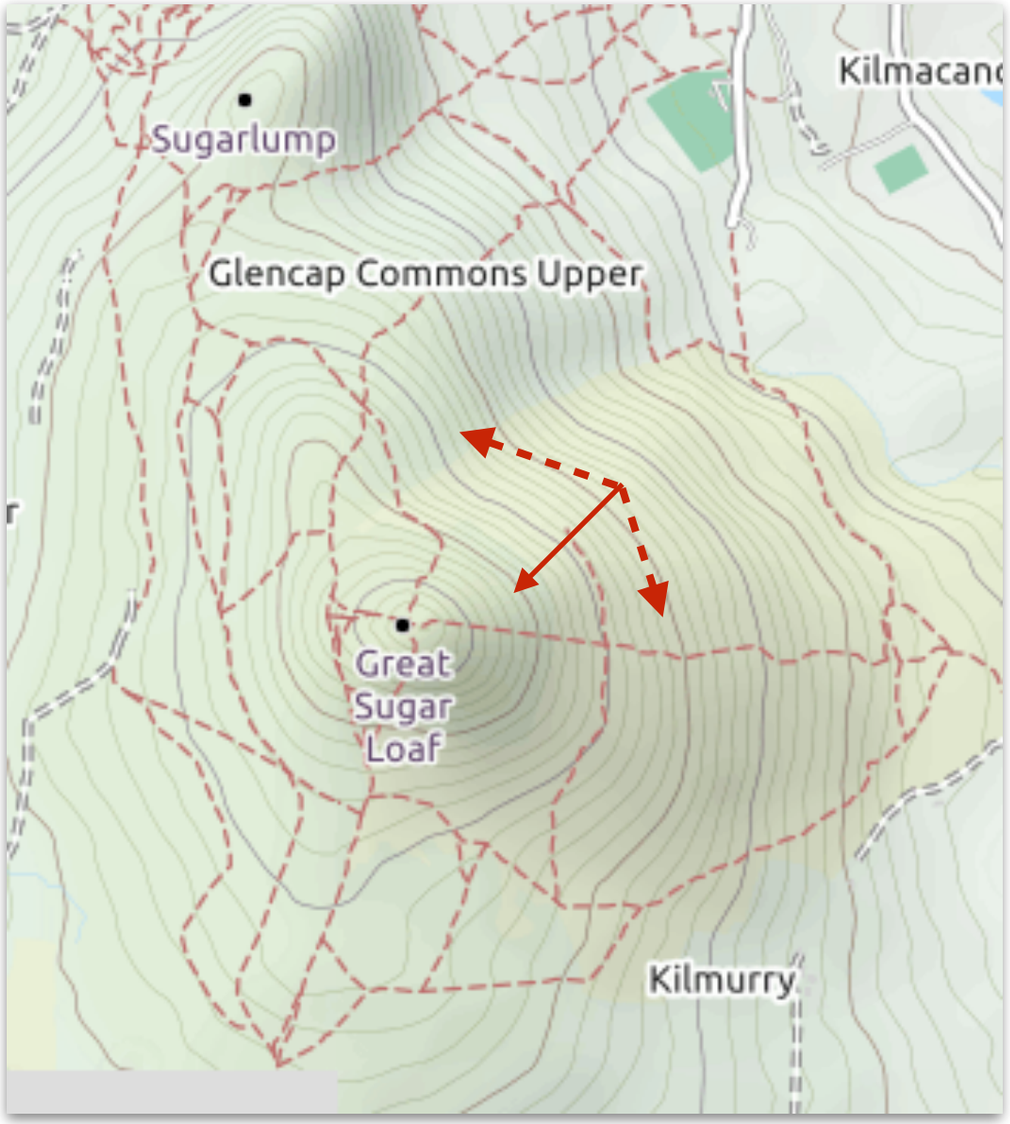
- Model has two parameters
- Therefore 2D solution space

Solving by Random Guessing



Gradient Ascent

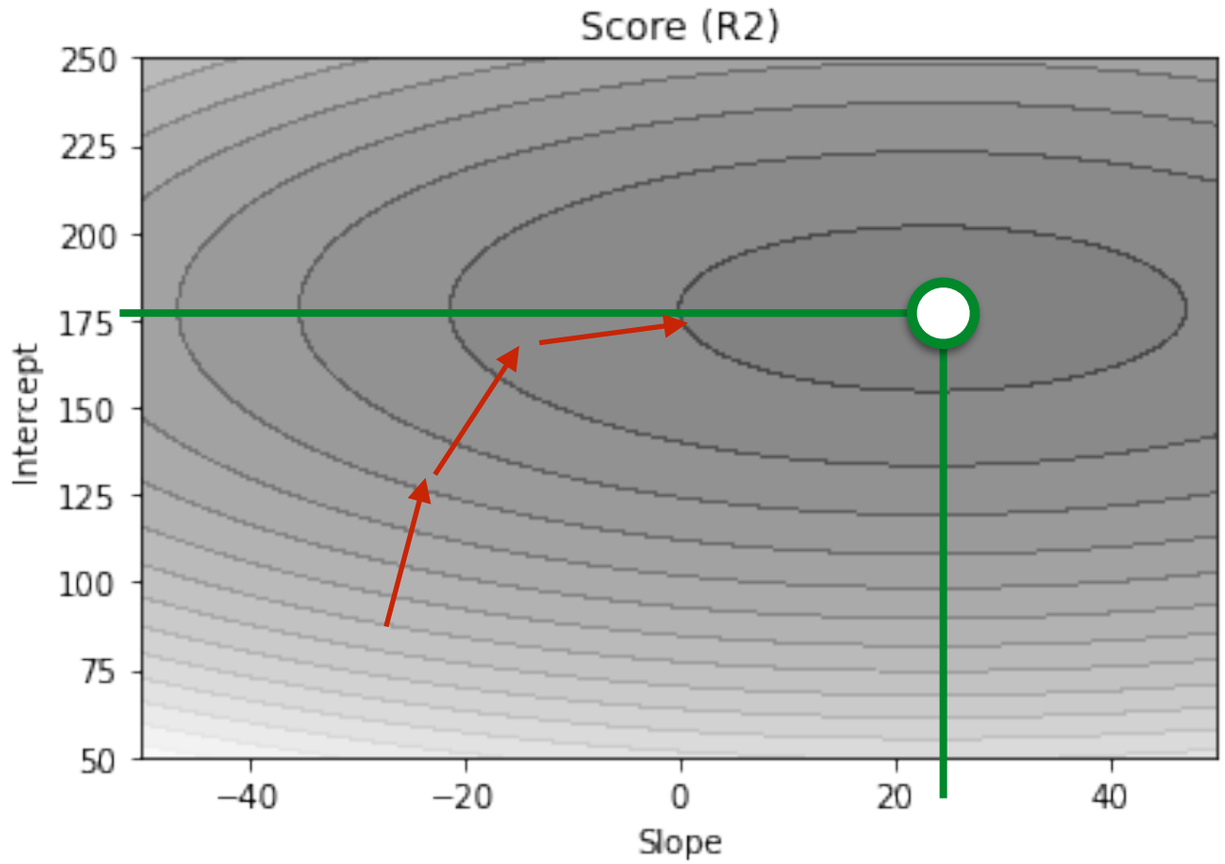
- Name this mountain?



Shortest route to top (typically) straight up - steepest gradient

Gradient Ascent/Descent in Regression

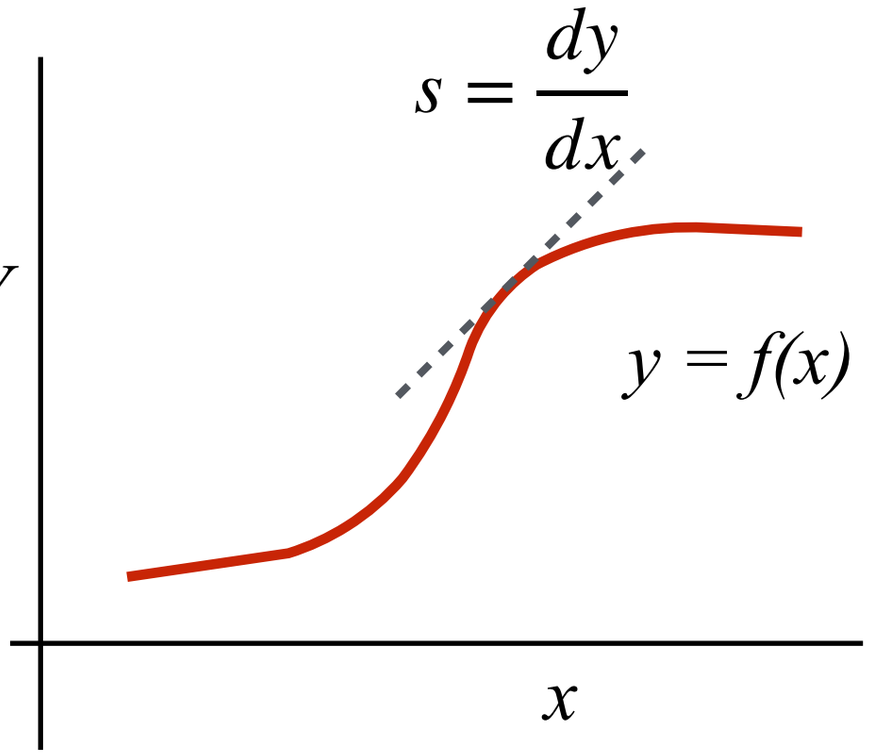
- Contour plot:
  - R^2 score based on Slope and Intercept.
  - R^2 score: bigger is better so gradient ascent
- 'Best model'
  - SGDRegressor
  - Slope: ~24
  - Intercept: ~178

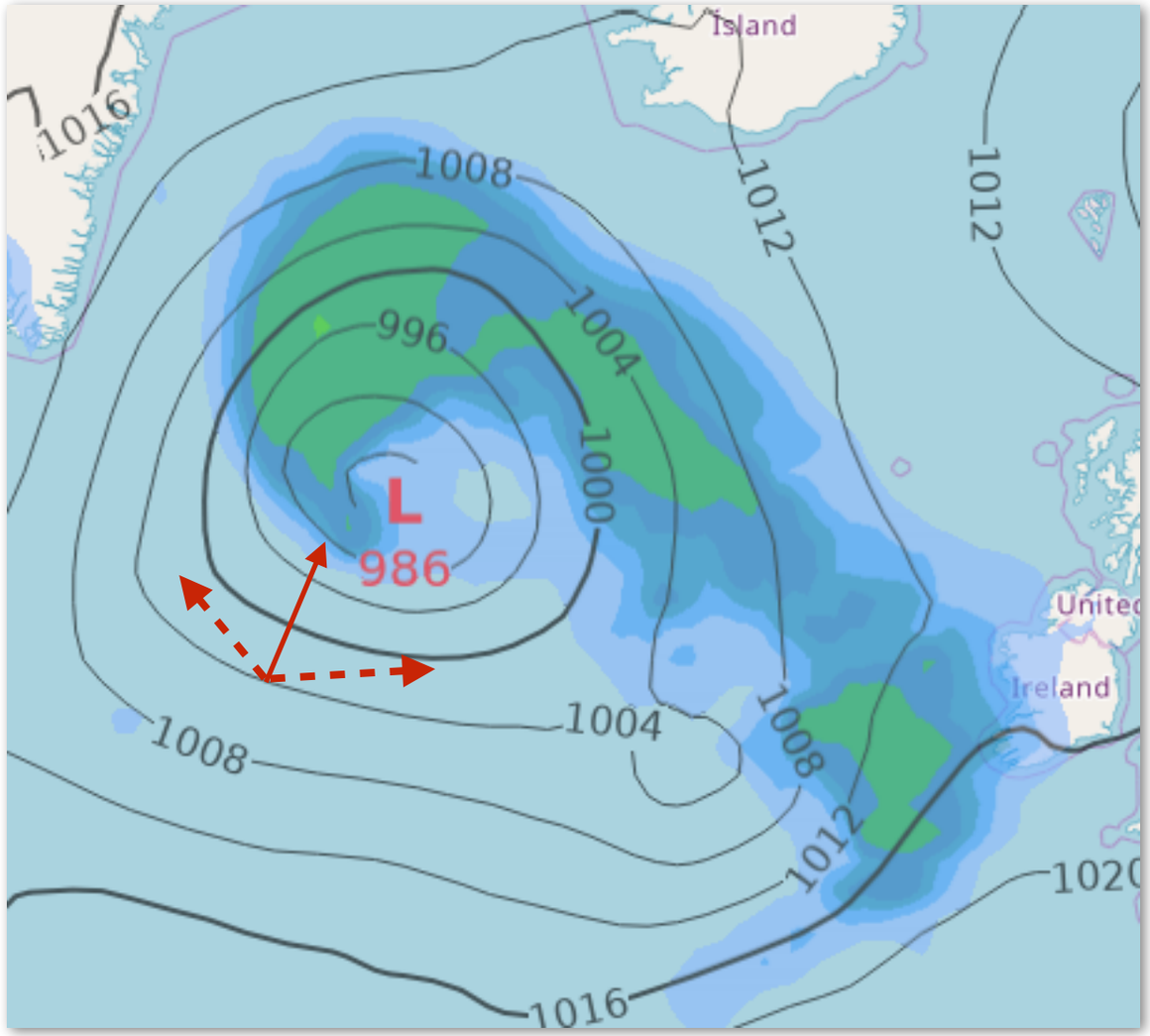


From notebook 12 Gradient Descent
1,200 models uniformly distributed in the parameter space.
Score reflects performance of a model on all the data.

Gadient Descent

- Shortest route to centre of Low pressure
  - Direction with steepest pressure gradient
  - Gradient is slope

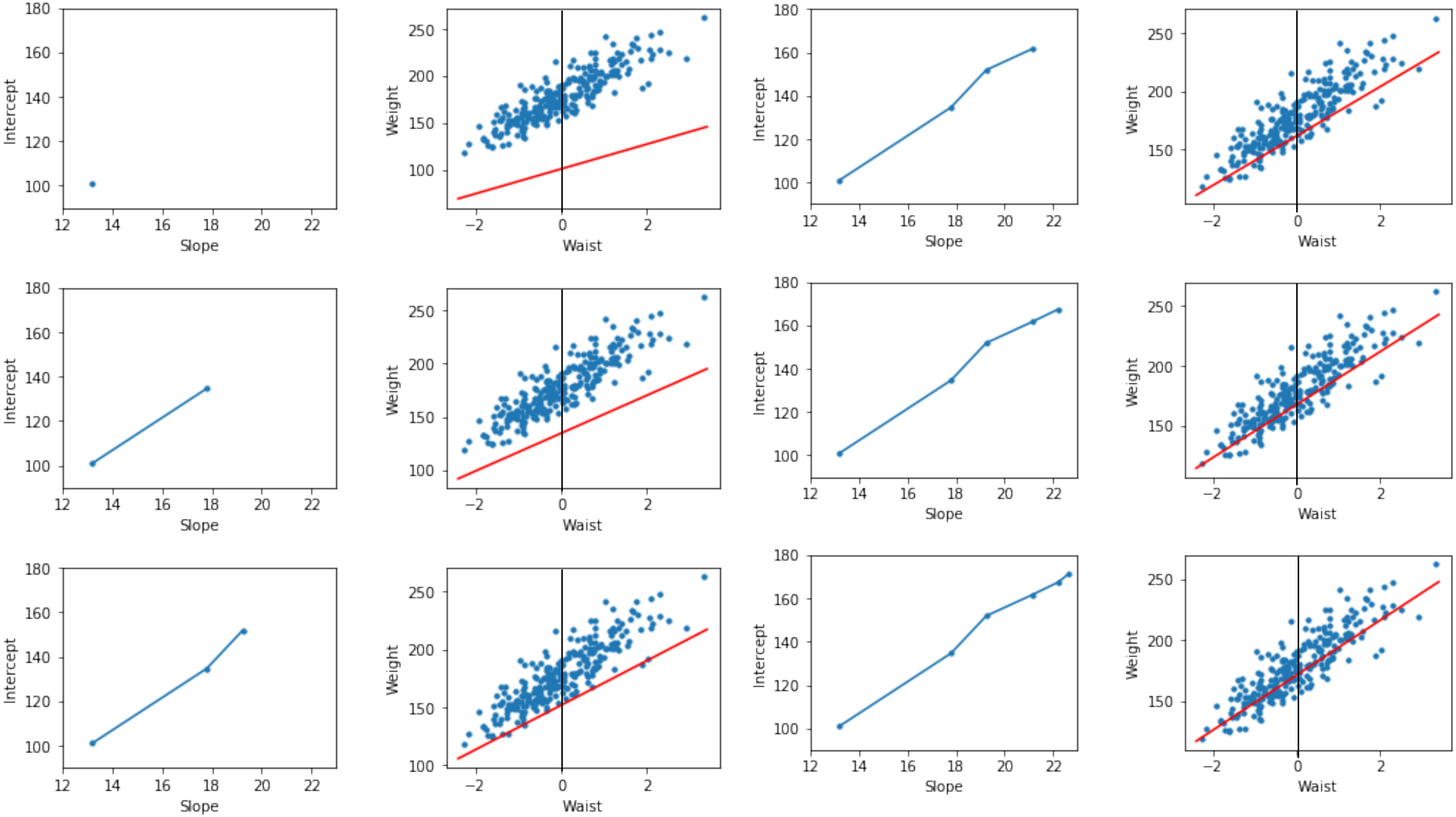




Multi-dimensional space:

select direction with steepest gradient

Solving by Gradient Descent



What have we learned?

LEARNING

Representation: choosing the functions that can be learned, the set of hypotheses

$$f(X) = \hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots$$

$$f^{\beta}(X) = \hat{y}^{\beta} = \sum_j \beta_j x_j$$

assume x_0=1

Evaluation: *loss function* for penalising errors:

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

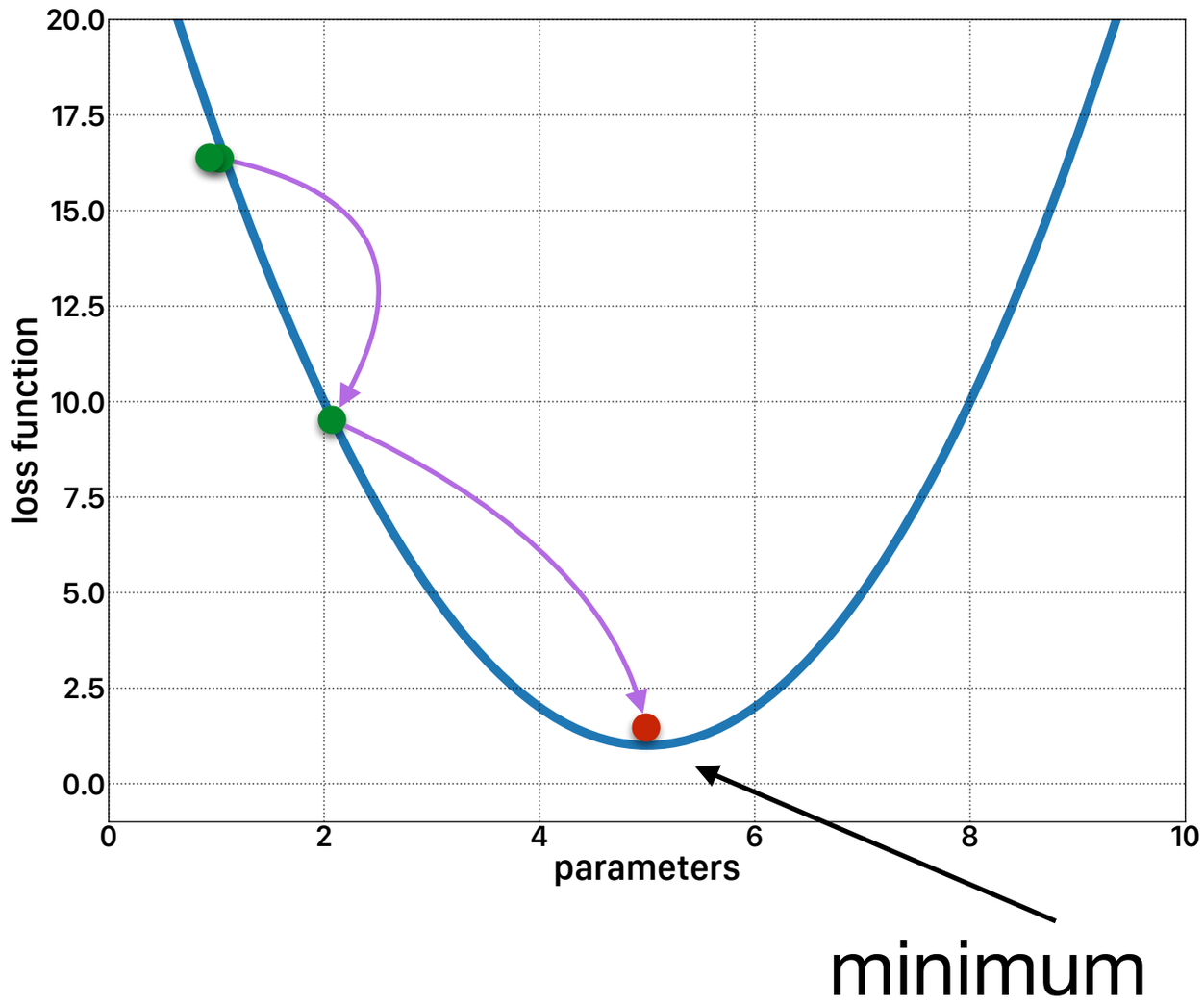
Optimisation:

$$\min_{\beta} J(\beta)$$

In the case of linear regression, we can compute the loss function (least squares minimisation) exactly. We won't be able to do this in general though...

Gradient Descent

- Gradient descent is an algorithm that makes small steps along a function to find a local minimum.
- We start at some point and find the gradient (slope)
- We take a step in the opposite direction to the gradient (ie. downhill)
- The size of the step is controlled by an adjustable parameter
- This algorithm gets us closer and closer to the local minimum.



In a 3D space, it would be like rolling a ball down a hill to find the lowest point

Gradient Descent

- How to do the update: $$\beta_j^{t+1} = \beta_j^t - \eta \frac{\delta}{\delta \beta_j} J(\beta^t)$$

- We start with some initial value of the parameters (could be randomly chosen or a reasonable first guess)
- Then we compute the gradient of the loss function with respect to that parameter
- Then adjust the parameter by a small amount (controlled by η), the opposite direction to the gradient (minus sign).
- By adjusting η, we can change how quickly we converge to the minimum. Large η -> risk of overshooting the minimum, small η -> might not converge on the local minimum.

Gradient Descent

- **Linear Regression:** for each parameter, we can compute the gradient and we find the update step is given by:

$$\beta_0 = \beta_0 - \eta \frac{\delta}{\delta \beta_0} J(\beta)$$

$$= \beta_0 - \eta \frac{\delta}{\delta \beta_0} \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i^{\beta} - y_i)^2$$

$$= \beta_0 - \frac{\eta}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i - y_i)$$

$$= \beta_0 - \frac{\eta}{n} \sum_{i=1}^{n} (\hat{y}_i^{\beta} - y_i)$$

$$\beta_1 = \beta_1 - \eta \frac{\delta}{\delta \beta_1} J(\beta)$$

$$= \beta_1 - \eta \frac{\delta}{\delta \beta_1} \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i^{\beta} - y_i)^2$$

$$= \beta_1 - \frac{\eta}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i - y_i) x_i$$

$$= \beta_1 - \frac{\eta}{n} \sum_{i=1}^{n} (\hat{y}_i^{\beta} - y_i) x_i$$

remember $\hat{y}_i = \beta_0 + \beta_1 x_i$

factor of x here!

Makes derivative calculations easier

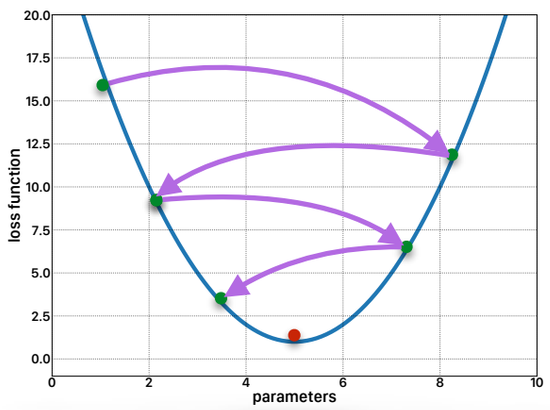
The update step for logistic regression is found by similar steps - using the derivative of the logistic function:

$$\frac{\partial}{\partial P} Logistic(P) = Logistic(P)(1 - Logistic(P))$$

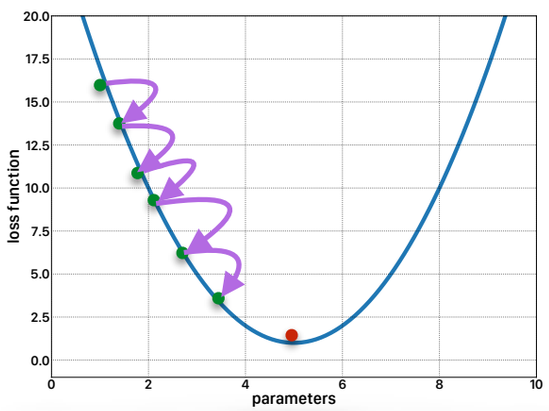
Gradient Descent

- The learning rate, η, determines the size of the adjustment made to each weight at each step in the process.
- What is the best value of η to choose?
    - There are many strategies for choosing the best value of η.
    - Most of the time we use rules of thumb, and also trial and error.
    - A typical range for η is [0.00001, 10]
- Similarly for the weights, we might start with initial values $[\beta_0, \beta_1] = [-0.2, 0.2]$

steps too large



steps too small



Gradient Descent

Notebook 12
Gradient Descent

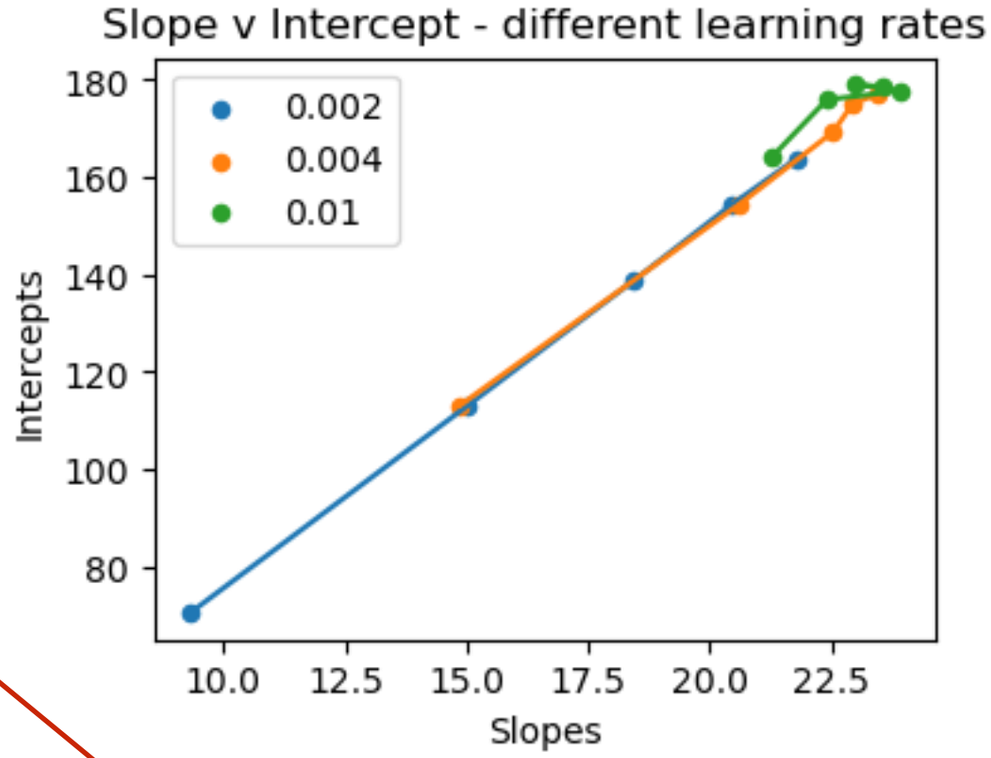
- Random initialisation for β_0 and β_1
- Update formulae:

$$\beta_0^{t+1} = \beta_0^t - \frac{\eta}{n}\sum_{i=1}^{n}(\hat{y}_i^{\beta^t} - y_i)$$

Learning rate

$$\beta_1^{t+1} = \beta_1^t - \frac{\eta}{n}\sum_{i=1}^{n}(\hat{y}_i^{\beta^t} - y_i)x_i$$

- Stop:
  - No improvements
  - Or after *n* iterations
- Learning rate: 
  - 0.002: too slow
  - 0.01: too fast
  - 0.004: 



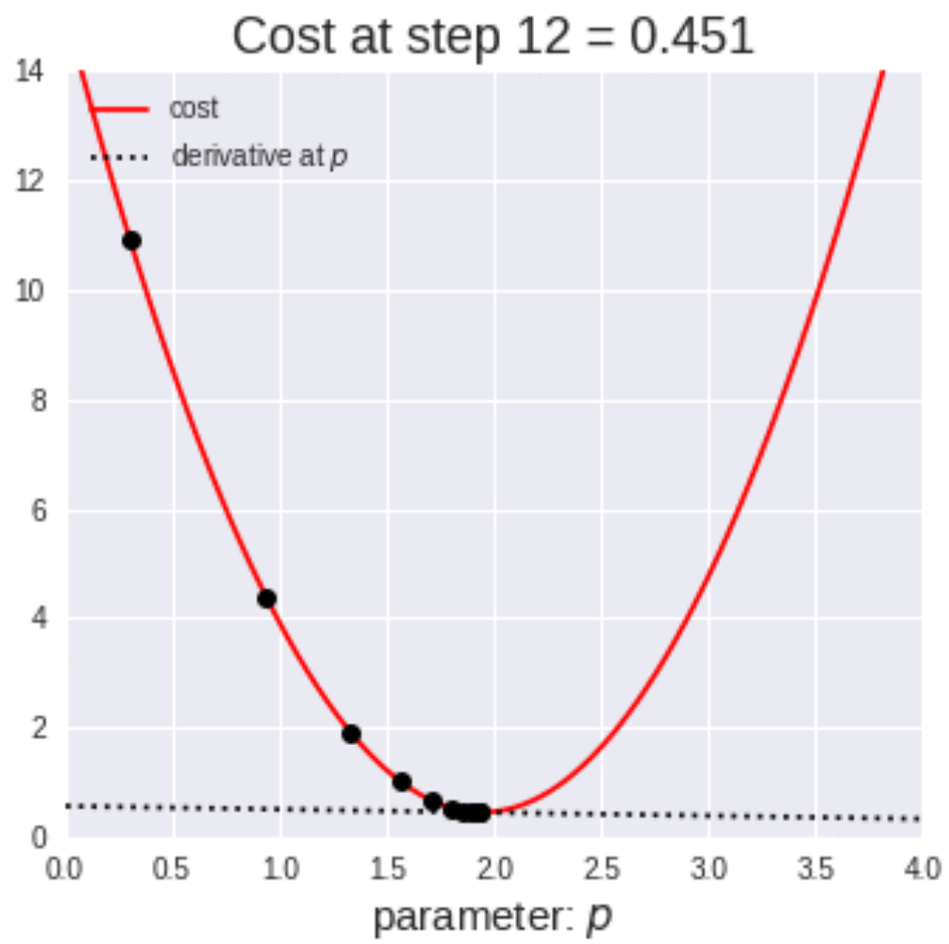
Learning rate = 0.004

	Slope β_1	Intcpt β_0	R^2
1	14.9	113.1	-5.15
2	20.6	154.5	-0.02
3	22.5	169.3	0.66
4	22.9	175.1	0.75
5	23.5	177.1	0.76

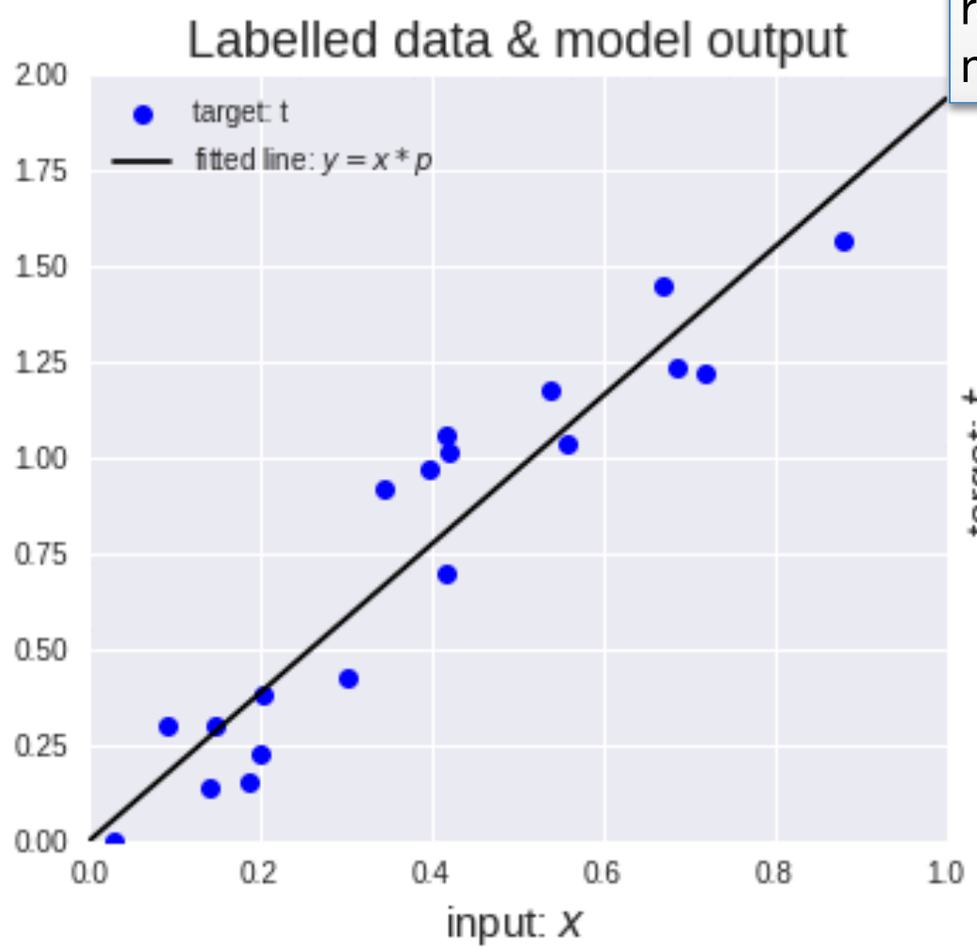
Gradient Descent in action

cost function

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



regression model



At the first step we have a poor guess for the parameters and our fit is bad.

As we perform more iterations of gradient descent, the fit improves (quite quickly)

When to stop the iterations?

- a suitable condition might be when the change in the loss function is below some (small) threshold value

Batch Gradient Descent

- In the examples so far we have computed the total loss function as the average loss for each training example:

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

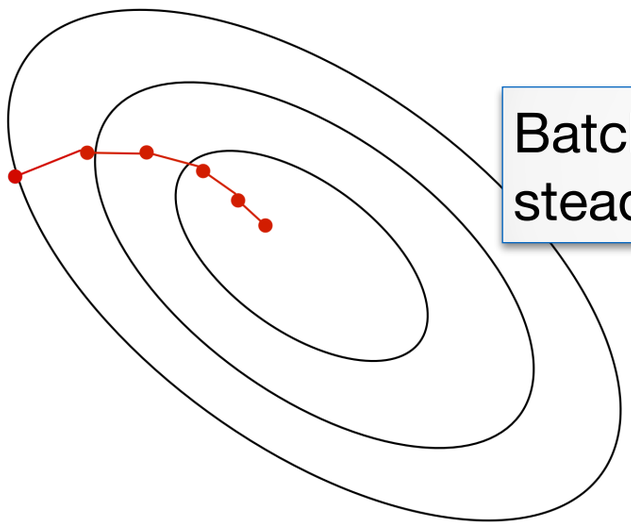
- This is called **Batch** Gradient Descent, because we need to sum over all training examples before we can make the updates.
- The problem with Batch GD is that it is very slow for large datasets!

Stochastic Gradient Descent

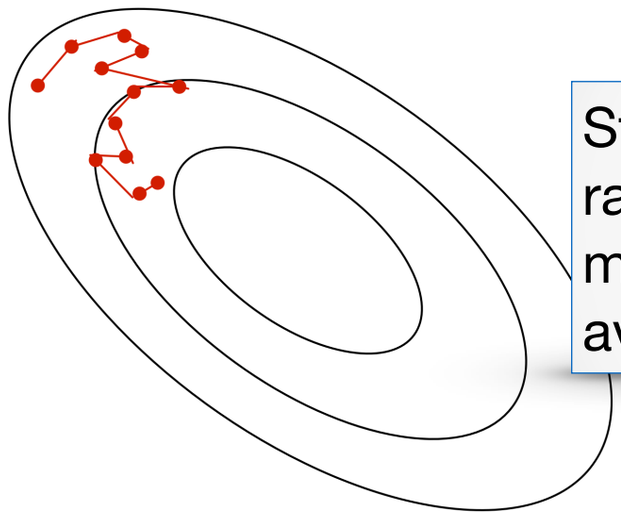
- in Stochastic GD we choose a single training example (perhaps at random), compute the loss and do the update directly:

$$\beta_j^{t+1} = \beta_j^t - \eta \frac{\delta}{\delta \beta_j} J(\beta^t)$$

- stochastic GD can surprisingly good progress with even a very small number of training examples.
- should converge closely to batch GD.
- But one at a time is still too slow, and doesn't benefit from parallelisation. So we often batch the training examples together and update the parameters for this group: **mini-batch** gradient descent.
- The batch size is a parameter we must choose carefully: too small and we get no benefit, too large and we are back to batch gradient descent



Batch GD moves steadily downhill



Stochastic GD takes random steps, but moves downhill on average

LinearRegression & SGDRegressor

- ‘count’ predicted by ‘atemp’

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
X_scal = StandardScaler().fit_transform(X)

reg = LinearRegression().fit(X_scal, y)
print(' R squared statistic: {:.3f}'.format(reg.score(X_scal, y)))
print(' Slope: {:.2f}'.format(reg.coef_[0]))
print(' Intercept: {:.2f}'.format(reg.intercept_))
```

```
R squared statistic: 0.398
Slope: 1221.67
Intercept: 4504.35
```

```
In [10]:
from sklearn.linear_model import SGDRegressor
SGD = SGDRegressor(max_iter=50, tol=1e-3).fit(X_scal, y)
print(' R squared statistic: {:.3f}'.format(SGD.score(X_scal, y)))
print(' Slope: {:.2f}'.format(SGD.coef_[0]))
print(' Intercept: {:.2f}'.format(SGD.intercept_[0]))
```

```
R squared statistic: 0.398
Slope: 1223.33
Intercept: 4505.43
```

Bike Sharing Exercise

- Assess the performance of Linear Regression models that use:
  - ‘atemp’ as the only input variable

```
X = bikes_df[['atemp']].values
```

  - ‘hum’ as the only input
  - all features except ‘casual’, ‘registered', ‘instant’ and ‘dteday’
    - as set up in notebook 12 Regression Part 2

- Use all the data for training and test
- You may use **`LinearRegression`** or **`SGDRegressor`**
- Score performance using R^2, MAPE and MAE

TL;DR Gradient Descent

1. A loss (error) function can be applied to an ML model
2. If that loss function is differentiable you can use GD
    1. Derivative of loss function directs parameter update

$$\beta_j^{t+1} = \beta_j^t - \eta \frac{\delta}{\delta \beta_j} J(\beta^t)$$

3. GD will allow you to search through the parameter space to find 'optimal' parameters
4. The GD search can be managed in a few different ways
    1. Learning rate (big or small jumps) 🦘
    2. Batch update
    3. Stochastic update
5. GD best used when the parameters cannot be calculated analytically
6. GD is one of the key algorithms in Machine Learning

Regularisation

- So far we train models to minimise the loss:
  - find weights that minimise squared error

$$\min_{\beta} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{\beta} - y_i)^2$$

- Prevent overfitting by constraining the capacity of the model
  - limit the size of the weights

$$\min_{\beta} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{\beta} - y_i)^2 + \alpha||\beta||$$

Penalty term
Norm of the weight matrix/vector

remember
pruning
decision trees

L1 and L2 Norms

- A norm is a quantity that describes the *size* of an vector or matrix, e.g. the length of a vector.
  - ML: weight matrix or weight vector (β in regression)

- Family of norms for different values of p $\quad ||\beta||_p$

- $p = 1$: L1 norm: $||\beta||_1 = |\beta_1| + |\beta_2| + \ldots + |\beta_n|$
  - (related to Manhattan distance)

- $p = 2$: L2 norm: $||\beta||_2 = \sqrt{\beta_1^2 + \beta_2^2 + \ldots + \beta_n^2}$
  - (related to Euclidean distance)

perhaps no square root in actual implementation

Regularisation Regression

- **Lasso:** L1 Norm

$$\min_{\beta} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{\beta} - y_i)^2 + \alpha||\beta||_1$$

  - Causes some of the weights to go to 0
  - Good for feature selection

- **Ridge Regression:** L2 Norm

$$\min_{\beta} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{\beta} - y_i)^2 + \alpha||\beta||_2$$

  - Weights scale in proportion
  - Good for preventing overfitting

- **Elastic Net:**
  - Combination of L1 & L2 Regularisation

Lasso example

Notebook 12
Regression

$$\min_{\beta} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{\beta} - y_i)^2 + \alpha||\beta||_1$$

controls regularisation

```
alp = 0.1
LassoX = Lasso(max_iter=5000, tol=1e-3, alpha = alp).fit(X_scal, y)
print('Alpha : {:3.1f}'.format(alp))
print('R squared statistic : {:.3f}'.format(LassoX.score(X_scal, y)))
for i,j in zip(LassoX.coef_, bikes_df.columns):
    print('Weight: {:>7.2f} {:s}'.format(i,j))
```

```
Alpha : 0.1
R squared statistic : 0.800
Weight:  565.33 season
Weight: 1020.25 yr
Weight: -133.90 mnth
Weight:  -86.70 holiday
Weight:  138.17 weekday
Weight:   55.86 workingday
Weight: -332.74 weathersit
Weight:  384.36 temp
Weight:  568.69 atemp
Weight: -144.80 hum
Weight: -198.26 windspeed
```

```
Alpha : 150
R squared statistic : 0.763
Weight:  333.62 season
Weight:  892.70 yr
Weight:    0.00 mnth
Weight:   -0.00 holiday
Weight:    0.59 weekday
Weight:    0.00 workingday
Weight: -280.50 weathersit
Weight:  261.83 temp
Weight:  612.78 atemp
Weight:   -0.00 hum
Weight:  -52.13 windspeed
```

less accurate
underfitted

COMP47750/COMP47990

Regression

Part III
Logistic Regression

Pádraig Cunningham
Based on slides by Aonghus Lawlor & Derek Greene

School of Computer Science

Regression - Overview

- Regression
  - Simple Linear Regression
    - Analytical Solution
- Gradient Descent
  - Multiple Linear Regression
- Logistic Regression
  - Regression for categorical features

Regression with Categorical Features

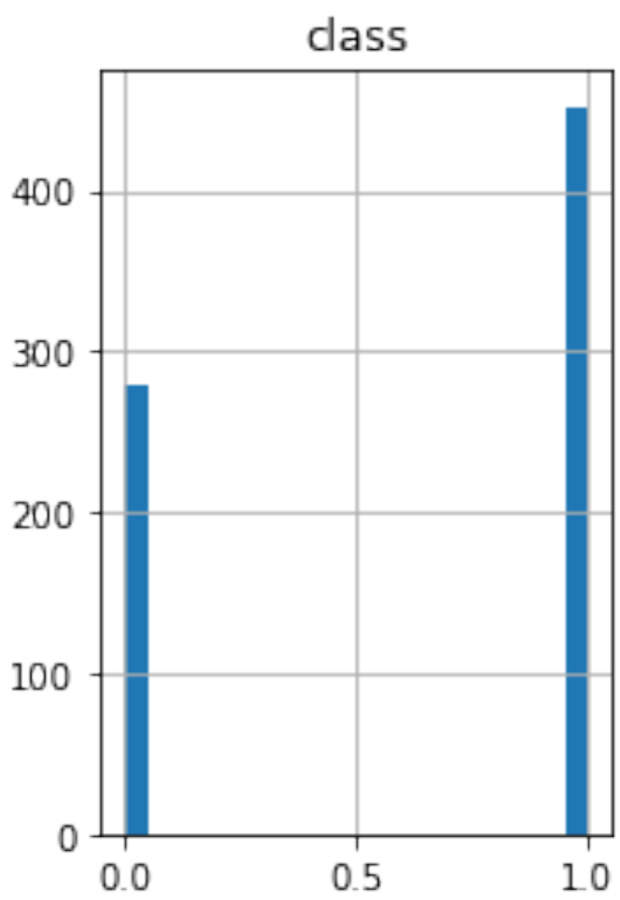
- Now consider a problem where we would like to predict an output *y* which takes the values {0, 1}.
- Example: The input variables could be age, gender, cholesterol level. As an output, we want to predict whether a patient has heart disease ⇒ {0: yes, 1: no}.

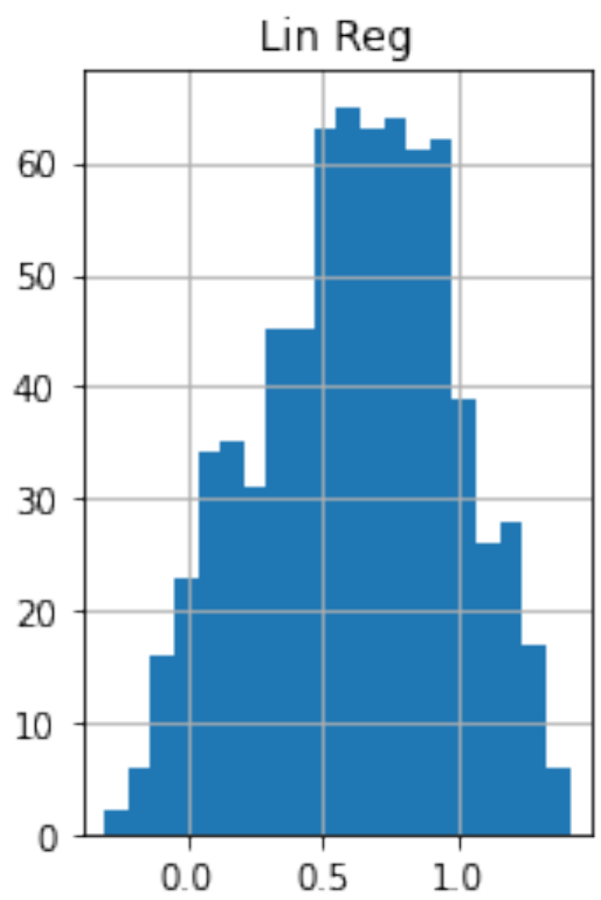
- Linear regression does not work well for this problem, since:
  - It is based on a linear model of the parameters.
  - It does not work well outside the variable range (e.g. predicting probabilities outside the range 0 to 1).
  - It does not map predictions to categorial values.

Why not use Linear Regression?

Notebook 12
Logistic Regression

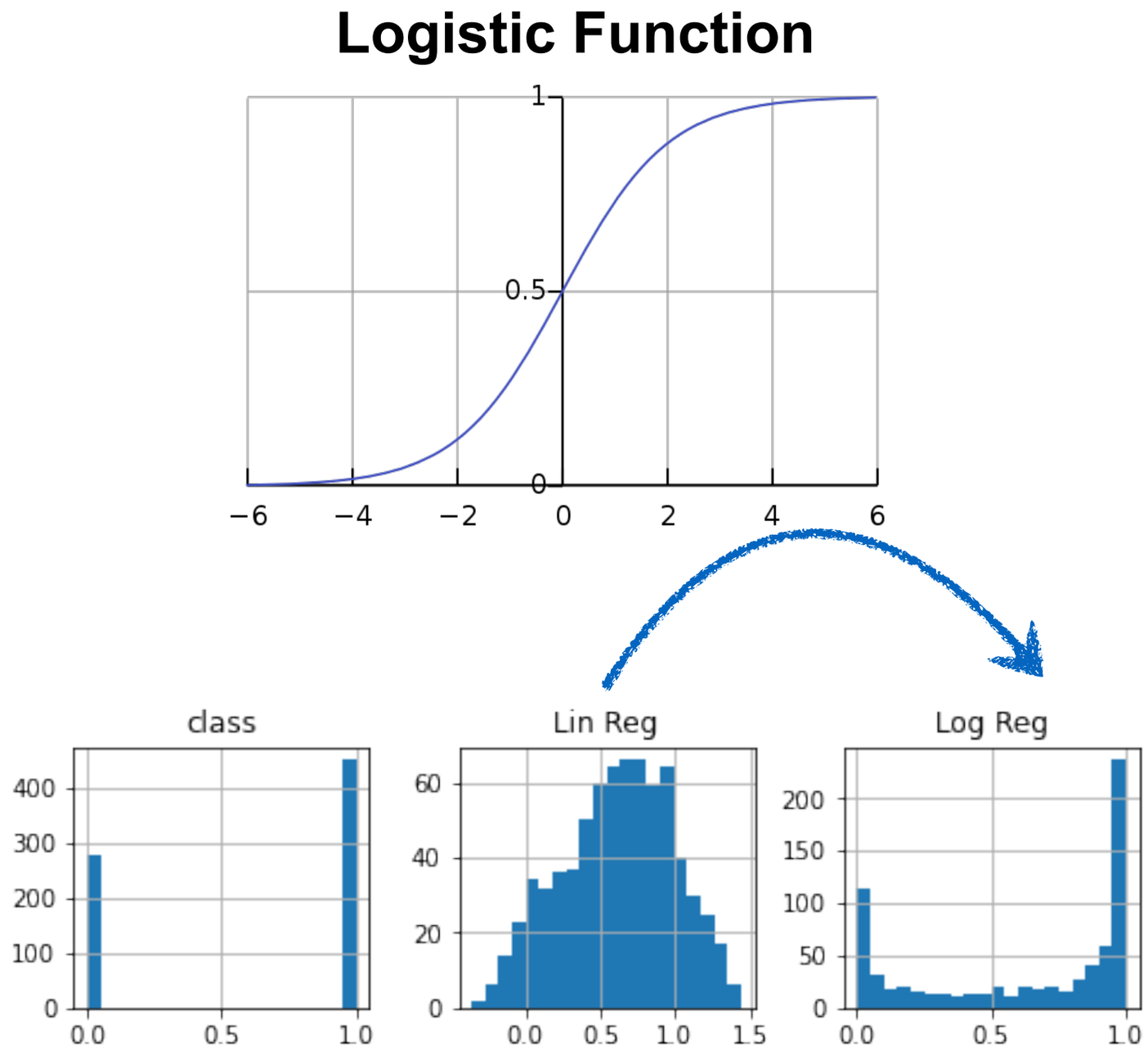
- Bike Sharing as a classification task
  - Low: < 4,000
  - High: > 4,000
- Linear Regression Model: 0,1 labels
  - ‘Probabilities’ outside [0,1] range





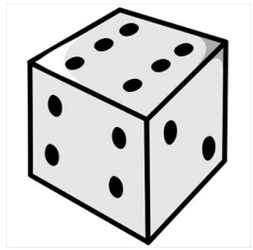
Why not use Linear Regression

- We need to squash the outputs into [0,1] range



Aside: Odds and Probabilities

- What is the probability of throwing a 6?
  - 1/6 = 0.16667
- What are the odds of throwing a 6?
  - 1:5
  - 1 to 5
  - 1/5 = 0.2
  - 0.2 is the monetary stake required to win 1 unit in a wager with fair odds.



$$odds = \frac{prob}{1 - prob}$$

$$prob = \frac{odds}{1 + odds}$$

Logistic Regression

- In linear regression the dependent variable is a numeric value

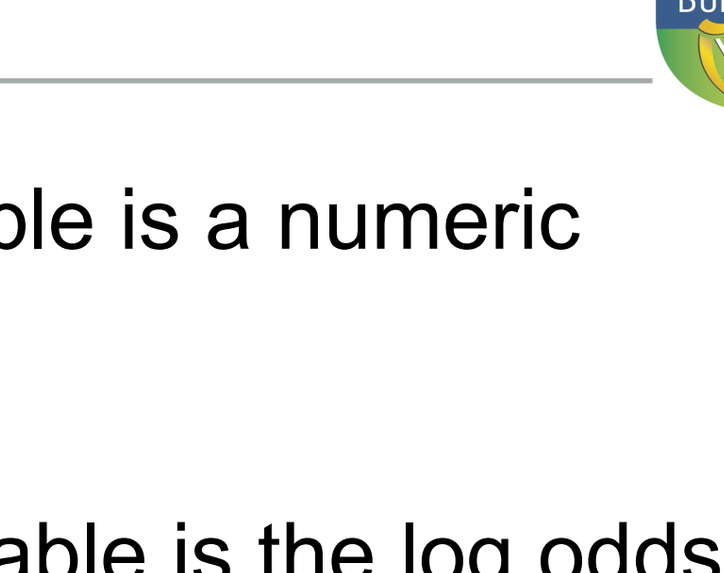
$$y = \beta_0 + \beta_1 x$$

- In logistic regression the dependent variable is the log odds that an outcome variable is 1.

$$ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x$$

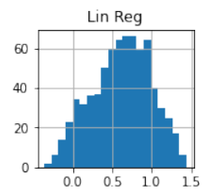
- p always in range [0,1]

- log odds $ln\left(\frac{p}{1-p}\right)$ in range [-∞ : +∞]



Logistic Regression

- log odds is the dependent variable

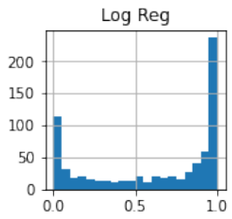


$$ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x$$

$$\frac{p}{1-p} = odds = e^{(\beta_0+\beta_1 x)}$$

$$p = \frac{e^{(\beta_0+\beta_1 x)}}{1+e^{(\beta_0+\beta_1 x)}} \quad \textbf{given} \quad prob = \frac{odds}{1+odds}$$

$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

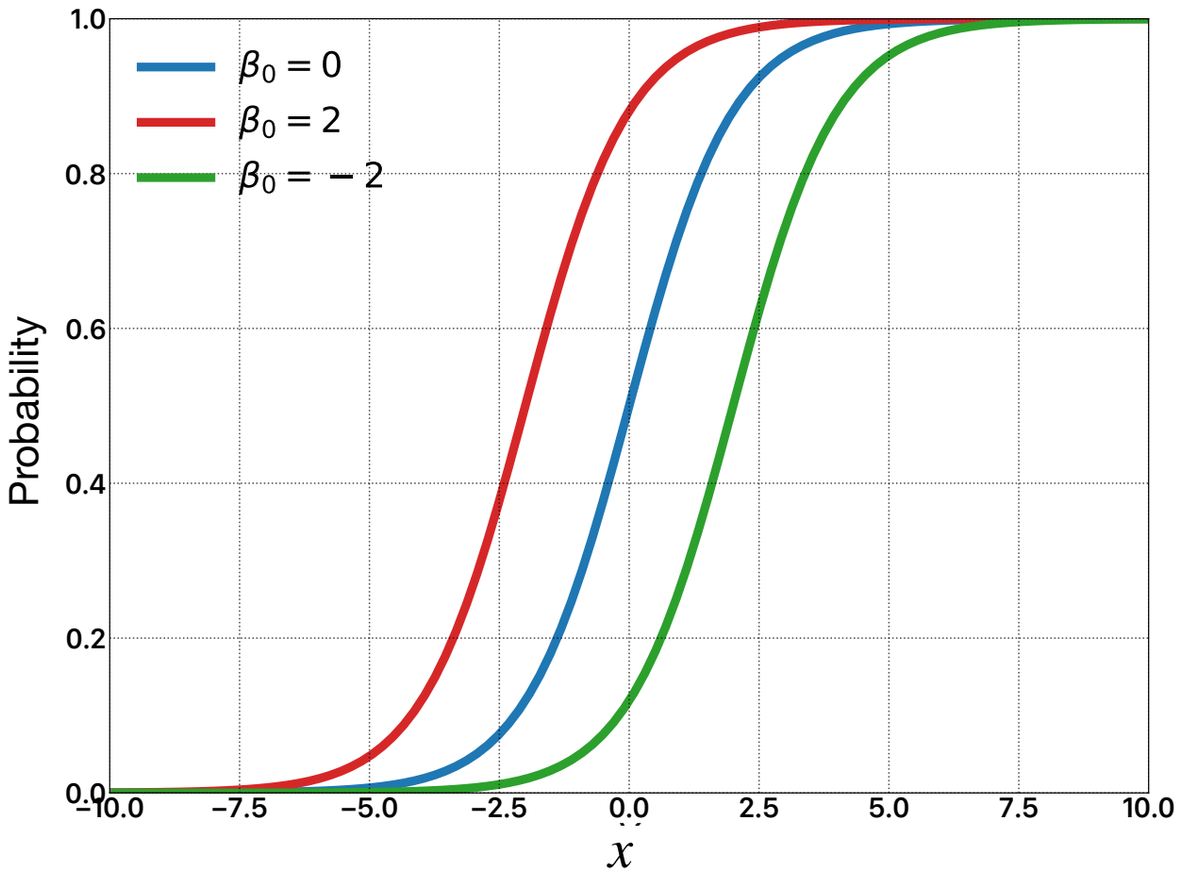


more generally

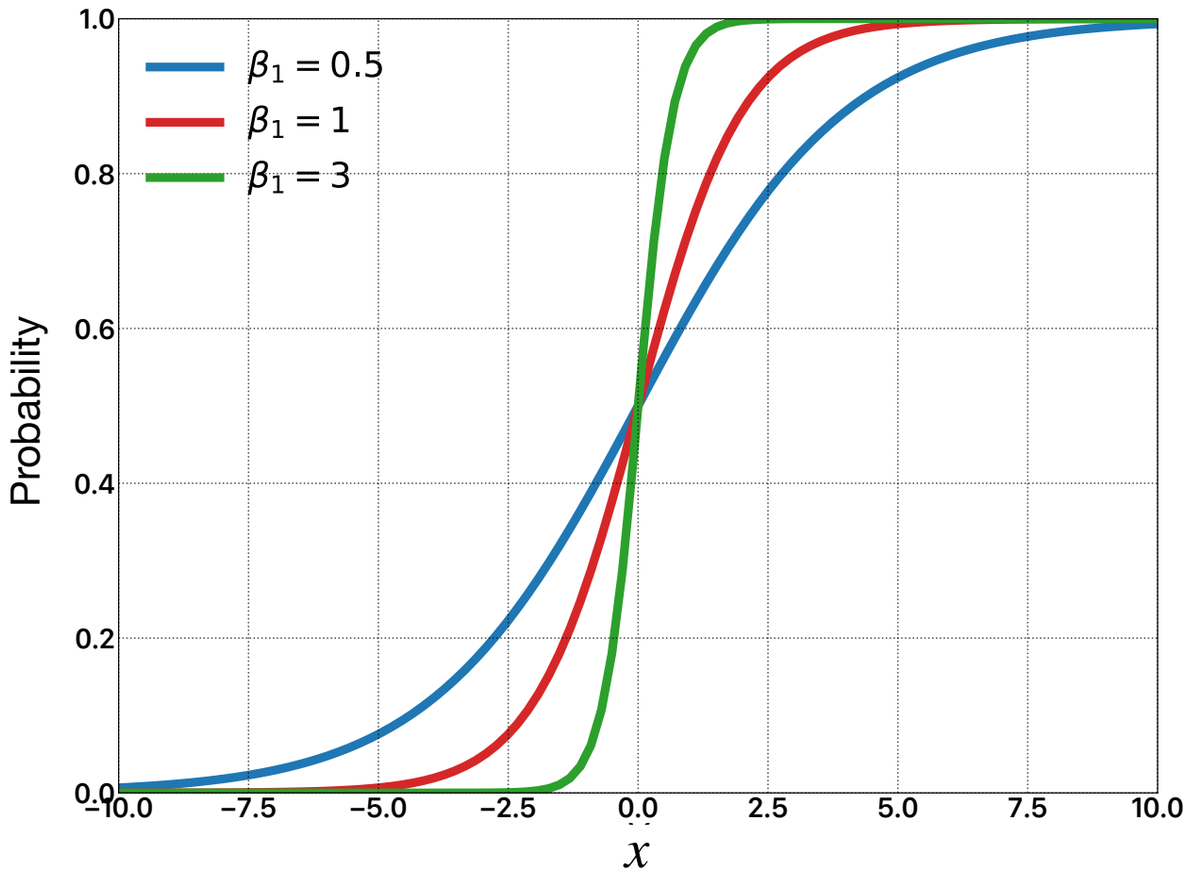
$$ln\frac{p}{1-p} = \beta_0 + \sum_{j=1}^{d} \beta_j x_j$$

Logistic Regression

- Model will predict P(Y = 1) with an S-shaped curve which is the general shape of the logistic function.



β_0 shifts the curve right or left



β_1 controls how steep the S-shaped curve is

Equivalent to a single layer neural network (1 neuron) with a sigmoid transfer function

Logistic Regression in sklearn

- Bike sharing
  - 4 attributes

	yr	mnth	weekday	temp	Class
0	0	3	6	0.344	0
1	0	7	0	0.366	0
2	0	1	1	0.196	1
3	0	12	2	0.200	0
4	0	1	3	0.227	1

```
from sklearn.linear_model import LogisticRegression
LR4 = LogisticRegression(random_state=0).fit(bikes4, y)
print('Accuracy \t %2.2f' %(LR4.score(bikes4, y)))
for name, indx in zip(bikes4.columns, range(4)):
    print('%s\t %2.2f' % (name, LR4.coef_[0][indx]))
```

```
Accuracy     0.86
yr           2.24
mnth         0.10
weekday      0.07
temp         7.59
```

Is year 23 times more important than month?

Coefficients with normalisation

- Scale features to N(0,1) so coefficients can be compared

```
from sklearn.linear_model import LogisticRegression
LR4 = LogisticRegression(random_state=0).fit(bikes4, y)
print('Accuracy \t %2.2f' %(LR4.score(bikes4, y)))
for name, indx in zip(bikes4.columns, range(4)):
    print('%s\t %2.2f' % (name, LR4.coef_[0][indx]))
```

```
Accuracy      0.86
yr          2.24
mnth        0.10
weekday     0.07
temp        7.59
```

```
bikes4_scal = StandardScaler().fit_transform(bikes4)
LR4s = LogisticRegression(random_state=0).fit(bikes4_scal, y)
print('Accuracy \t %2.2f' %(LR4s.score(bikes4_scal, y)))
for name, indx in zip(bikes4.columns, range(4)):
    print('%s\t %2.2f' % (name, LR4s.coef_[0][indx]))
```

```
Accuracy      0.86
yr          1.51
mnth        0.38
weekday     0.21
temp        2.25
```

Maybe 4 times more important

Regression - Overview

- Regression
  - Simple Linear Regression
    - Analytical Solution
- Gradient Descent
  - Multiple Linear Regression
- Logistic Regression
  - Regression for categorical features