COMP47460

Regression & Gradient Descent

Aonghus Lawlor
Derek Greene

School of Computer Science
Autumn 2018

Regression

Regression is a well known and well developed method in statistics

Machine learning borrows a lot of ideas from statistics, and regression is a core algorithm is predictive modelling

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

Regression

- Linear dependence: constant rate of increase of one variable with respect to another (as opposed to, e.g., diminishing returns).
- A regression analysis describes the relationship between two or more variables.

- We will try to understand:
  - how to build the linear model
  - make predictions
  - test the significance of the results

Examples:

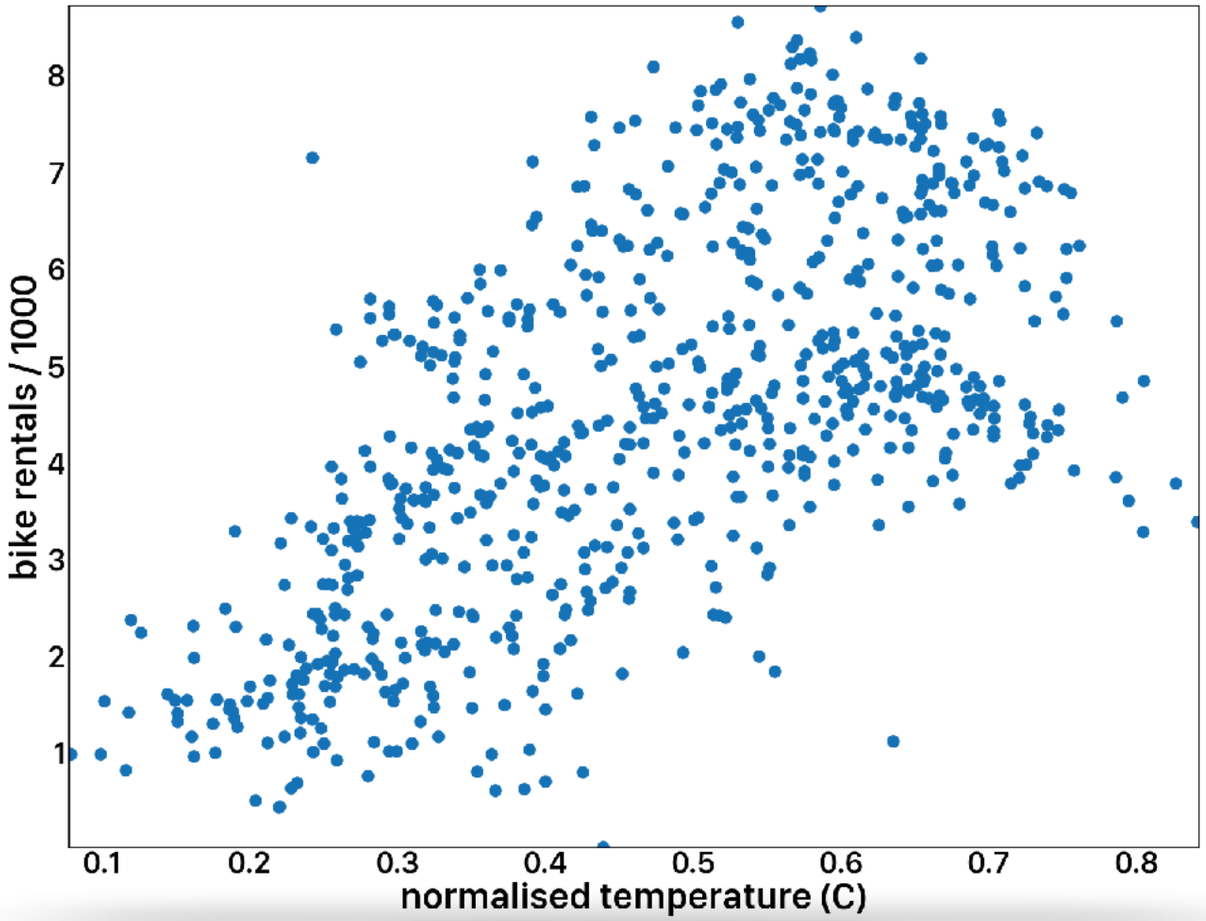
How does earned income relate to educational level?

What is the connections between demand for electricity and the weather

How do house sales depend on interest rates?

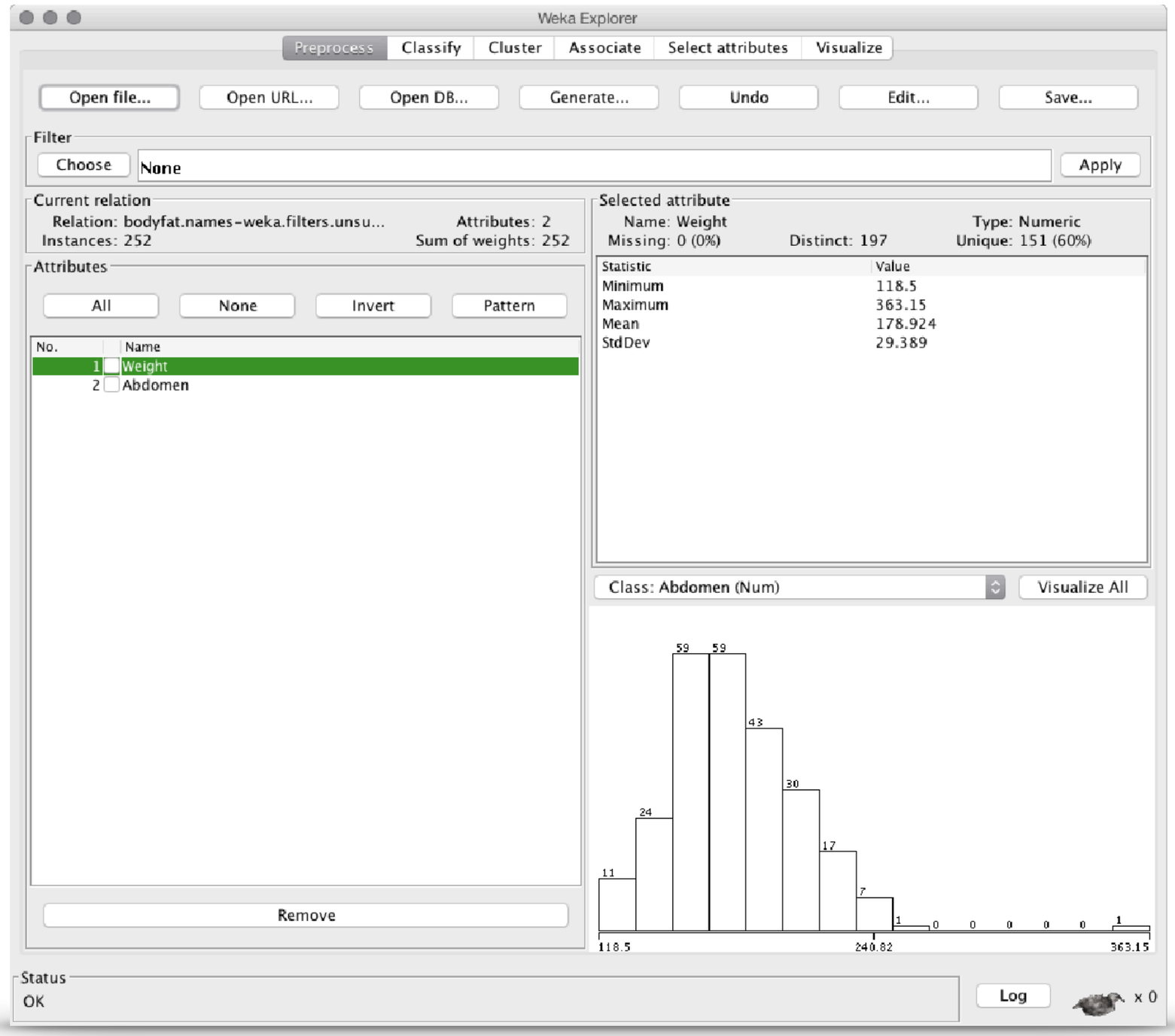
Regression Example

- Bike Sharing Data
- We would like to know how the number of bike rentals per day depends on the 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?



Weka

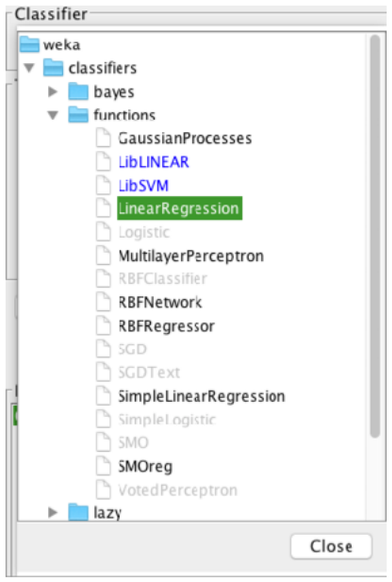
Weka can do Ordinary Least Squares linear regression

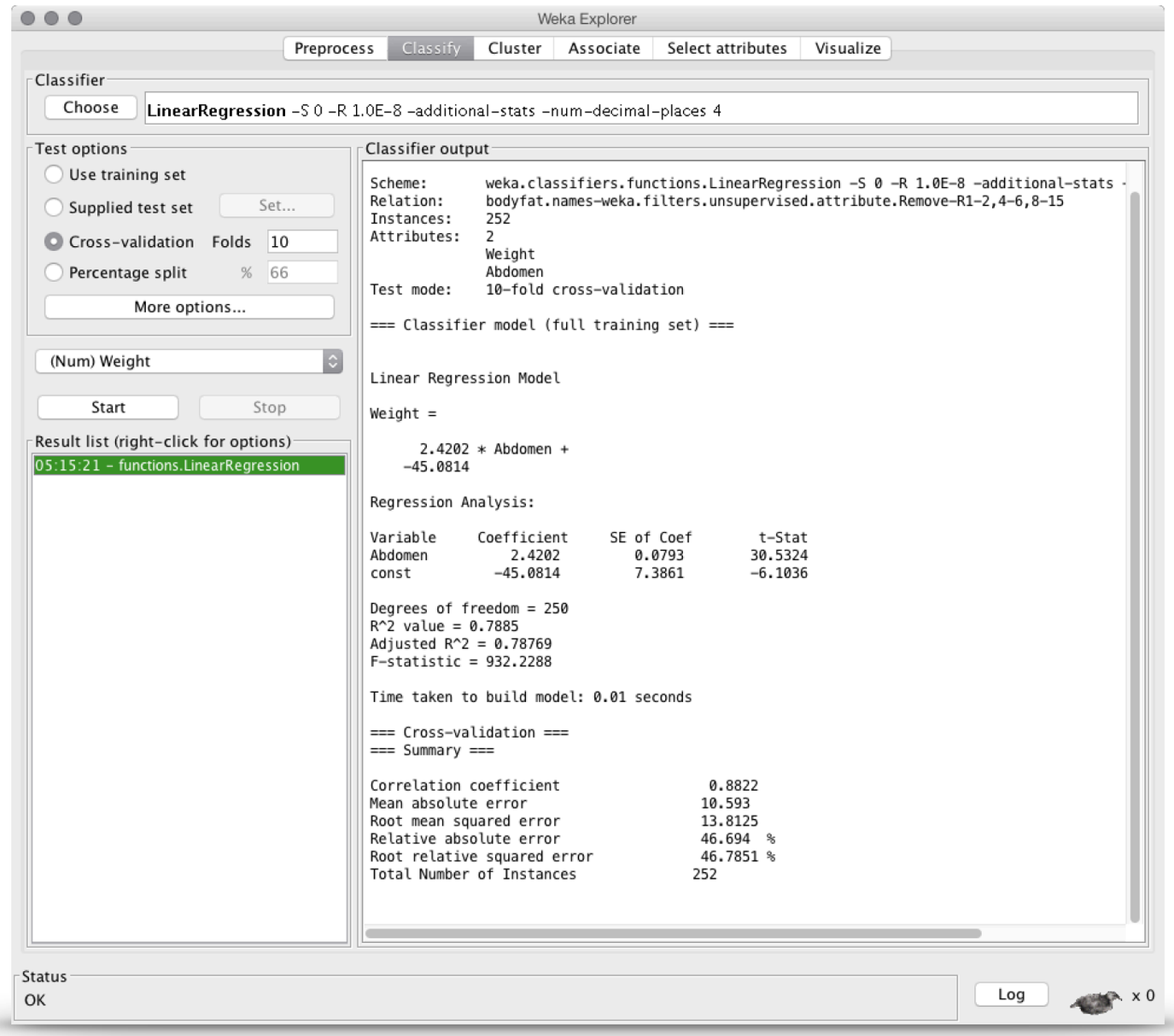


Weka

In Weka:

classifiers->functions->LinearRegression

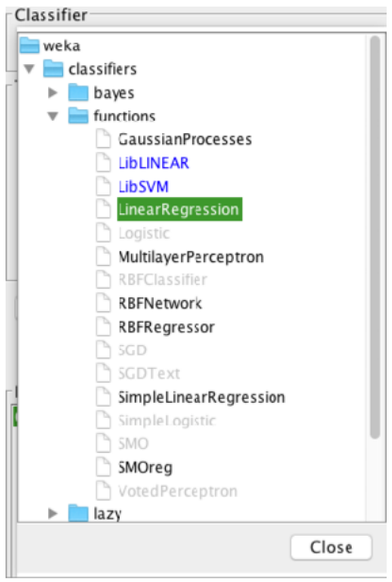


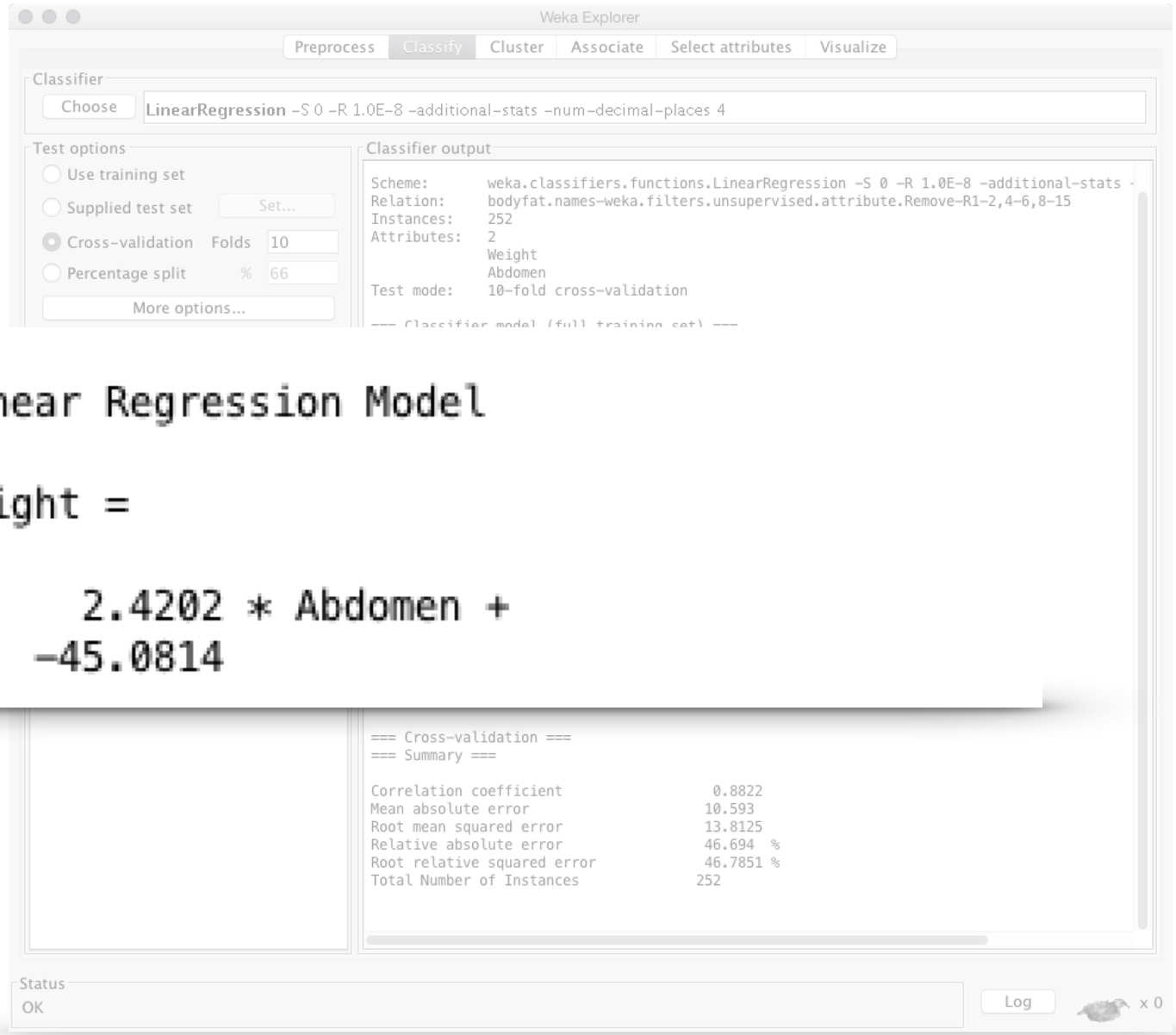


Weka

In Weka:

classifiers->functions->LinearRegression





Weka tells us the linear model is:

Weight = 2.4202 * Abdomen -45.0814

Terminology

X independent variables

Y dependent variables

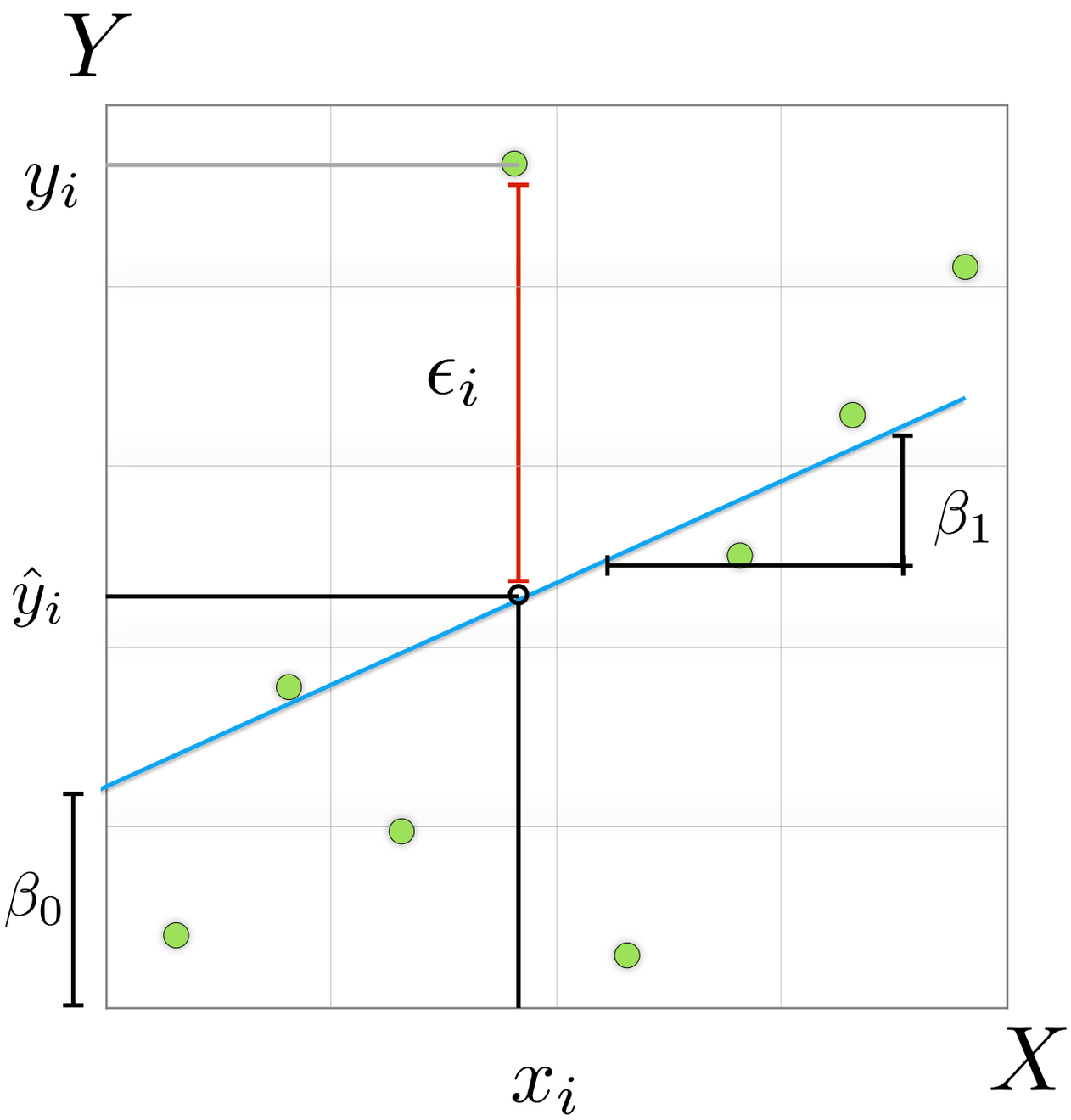
y_i observed value of Y

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

β_0 intercept

β_1 slope

ϵ_i error



Simple Linear Regression

- We assume that Y (the dependent variable) 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
  - β_1: estimated change in average value of Y if we make a unit change in X

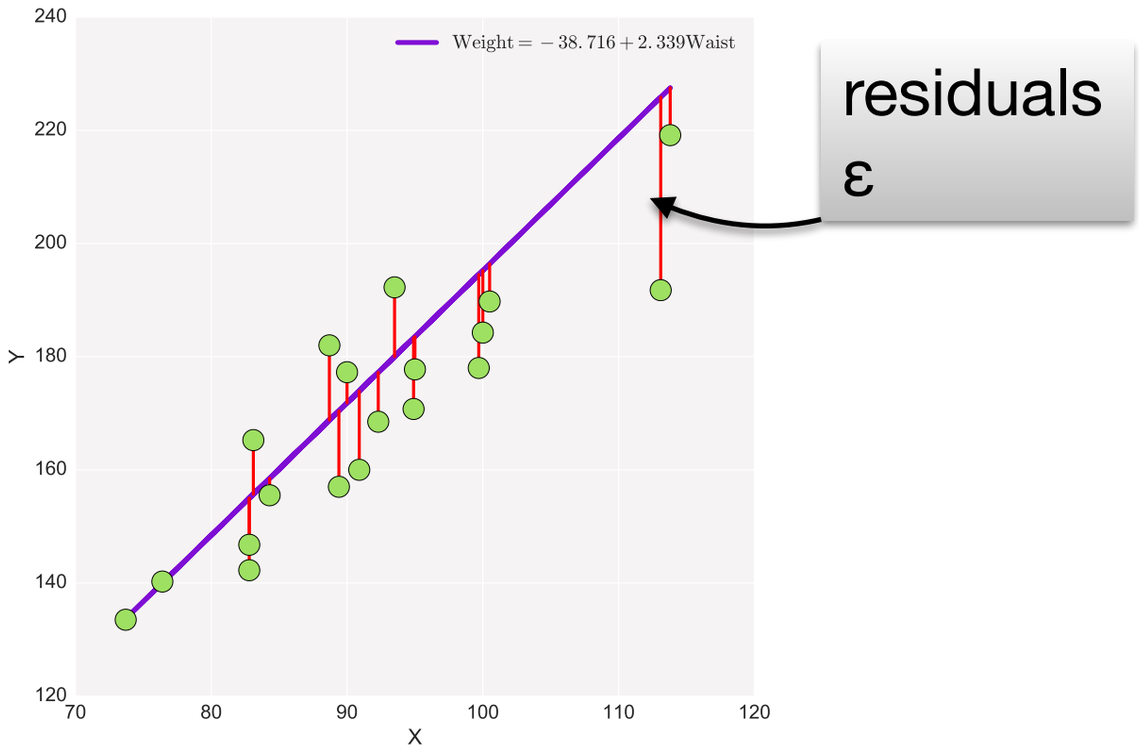
simple linear regression model

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

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?

- 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 (residuals)



sum(squared error) = 2995.6991

Regression Coefficients

The best fit values for the parameters β are found by minimising the sum of squared errors and solving for β_0, β_1:

$$\frac{\partial}{\partial \beta} \sum_{i=1}^{n} \epsilon_i^2 = 0$$

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

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

First compute the coefficients S_{XX} and S_{XY}

$$x_i = (X_i - \bar{X})$$

$$y_i = (Y_i - \bar{Y}_i)$$

use mean centred variables

Best fit parameters:

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

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

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

Fitted (estimated) regression model

Caveat: regression relationship are valid only for values of the regressor variable within the range the original data. Be careful with extrapolation.

Regression Coefficients

For calculation purposes, it is not so convenient to use the mean centred variables, instead use these equivalents:

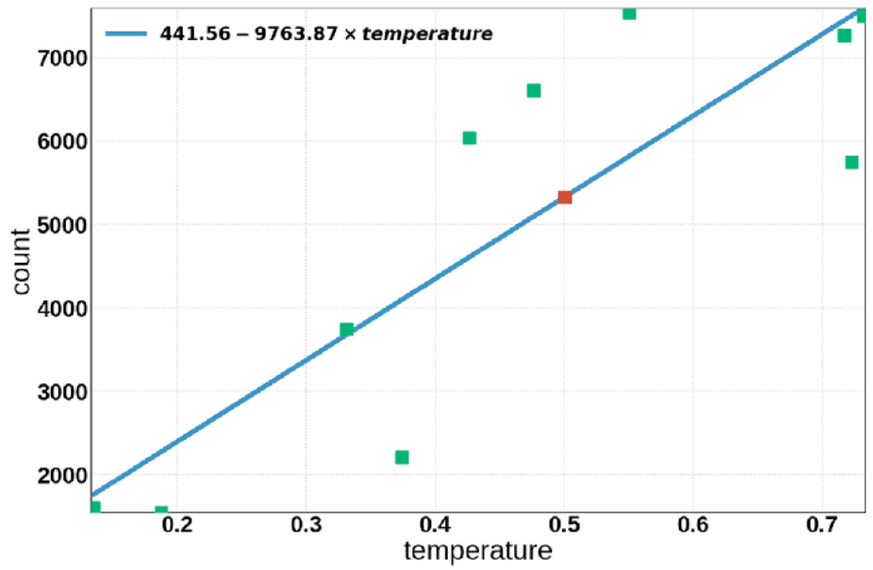
$$\hat{\beta}_1 = \frac{\sum X_i Y_i - \frac{(\sum X_i)(\sum Y_i)}{n}}{\sum X_i^2 - \frac{(\sum X_i)^2}{n}}$$

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

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

For temperature=0.5 we predict the count to be:

441.5571 + 9763.8694 * 0.5 = 5323.492



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

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

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

Errors

- When we want to measure the strength of the linear relationship in the data
- First, consider the sources of error in regression

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

total sum of squared deviations in Y from its mean

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

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

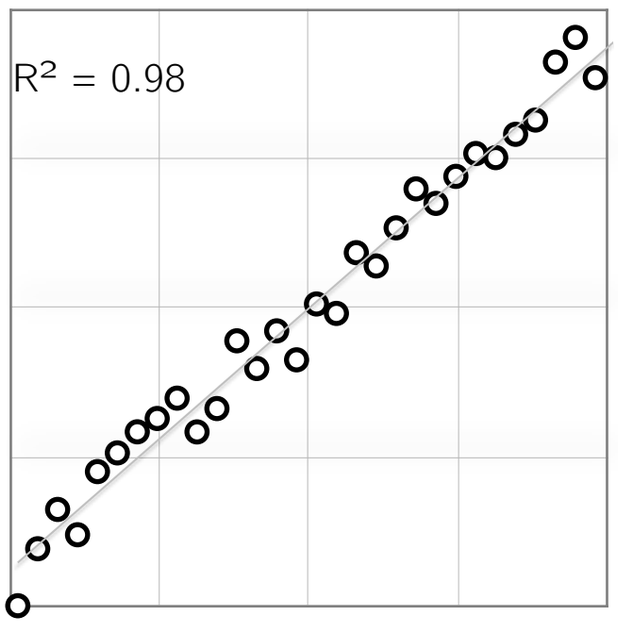
remains unexplained

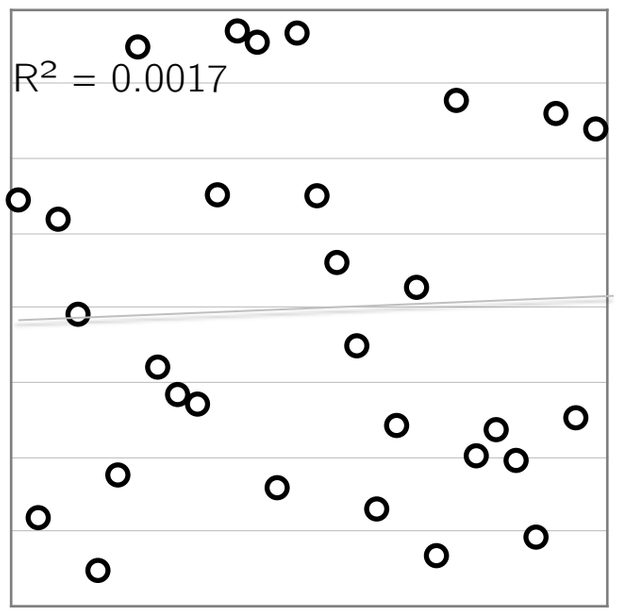
How well did we predict?

- If there is linear relationship the slope (β_1) will be non-zero.
- start with hypothesis that $\beta_1 = 0$ (this implies there is no linear relationship between the variables Y and X)
- we would like to test this hypothesis
- this will give us a statistical measure of how certain we can be there is a linear relationship in the data.

strong *+ve* relationship

β_1 is > 0





no relationship

β_1 is 0

P-Value Testing

General Approach for Testing

1. Calculate a test statistic on the sample data that is relevant to the hypothesis being examined.
2. Convert the result to a *p*-value by comparing its value to the distribution of test statistics under the null hypothesis.
3. Decide, for a specific level of significance, if we should reject or not reject the null hypothesis, based on the *p*-value:

$$p \leq \alpha \implies \text{reject } H_0 \text{ at level } \alpha$$
$$p > \alpha \implies \text{do not reject } H_0 \text{ at level } \alpha$$

"Is it low enough to be significant?"

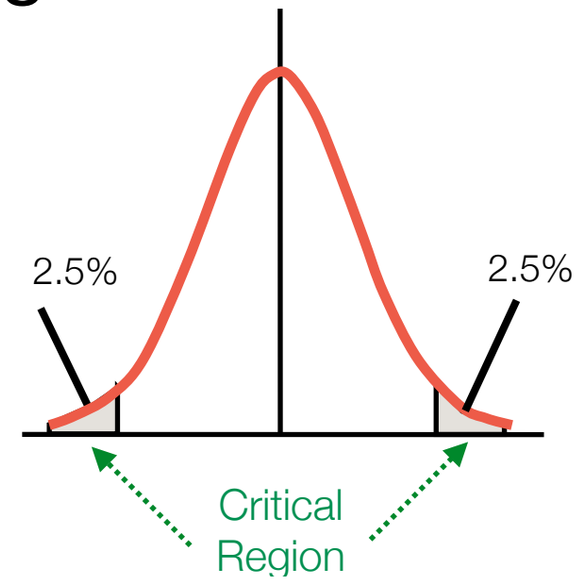
- The actual *p*-value threshold (α) depends on the problem, but 0.05 or 0.01 are often chosen "by default".
- The choice controls the Type I Error rate: "How serious is it to believe that something is true when it is in fact false?"

t-test

- Having determined the slope of our best fit line, we would like to make an inference about the slope by testing:

$$H_0: \quad \hat{\beta}_1 = 0 \quad \text{null hypothesis}$$

$$H_1: \quad \beta_1^0 \neq 0$$



- H_1 is the two-tailed alternative hypothesis
- The appropriate test is a two-sided *t-statistic* with (n-2) degrees of freedom, due to the standard error of β_1 in the denominator.
- The computed *t value* is compared with the appropriate critical *t value* for the required confidence level.

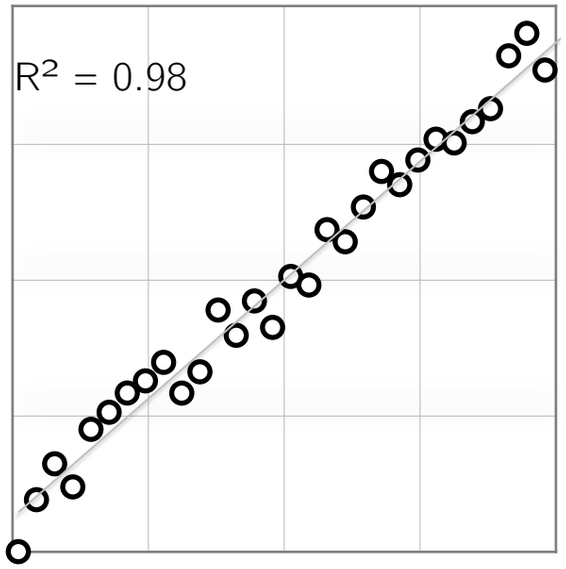
$$t = \frac{\hat{\beta}_1 - \beta_1^0}{s.e(\hat{\beta}_1)}$$

$$s.e(\hat{\beta}_1) = \sqrt{\frac{1}{n-2} \frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n} (X_i - \bar{X})^2}}$$

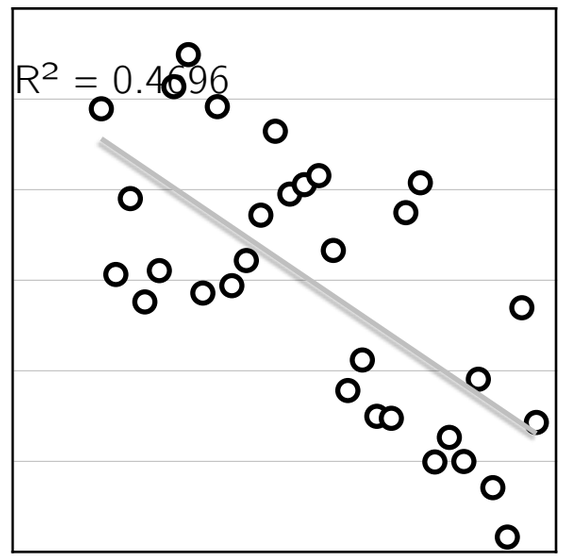
Correlation

- 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
- tells us how well our regression line matches the real data. The closer R^2 is to 1 the better the fit.

strong +*ve* relationship



weak -*ve* relationship



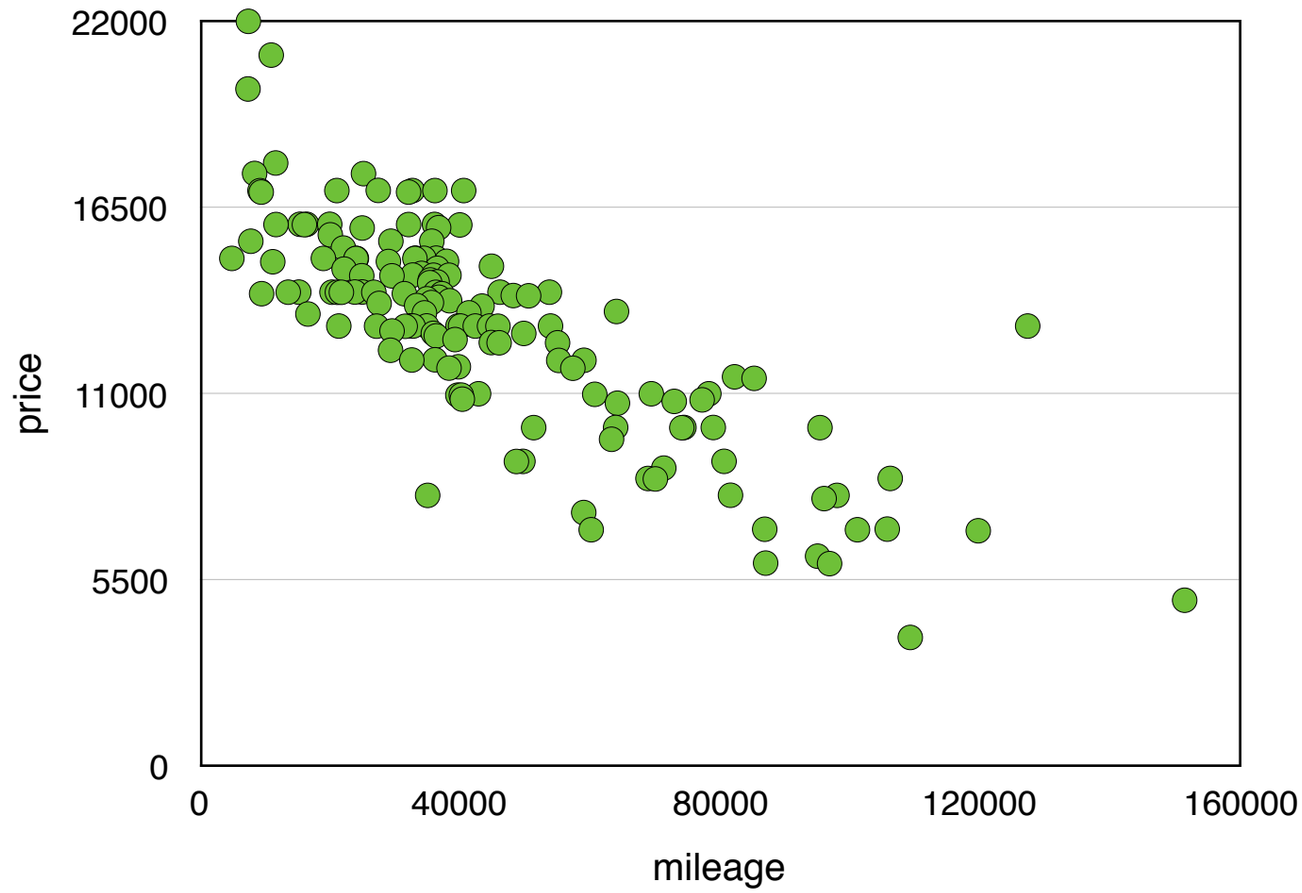
$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

correlation coefficient R^2

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

Example t-test

- used cars are bought at auction and the dealer would like to be able to predict the sale price given the mileage on the odometer.
- the best fit linear model is:



price = -0.093 * mileage + 17091.05

- Is a linear model a good fit to the data (t-test)?
- How closely does the line fit (Cor)?

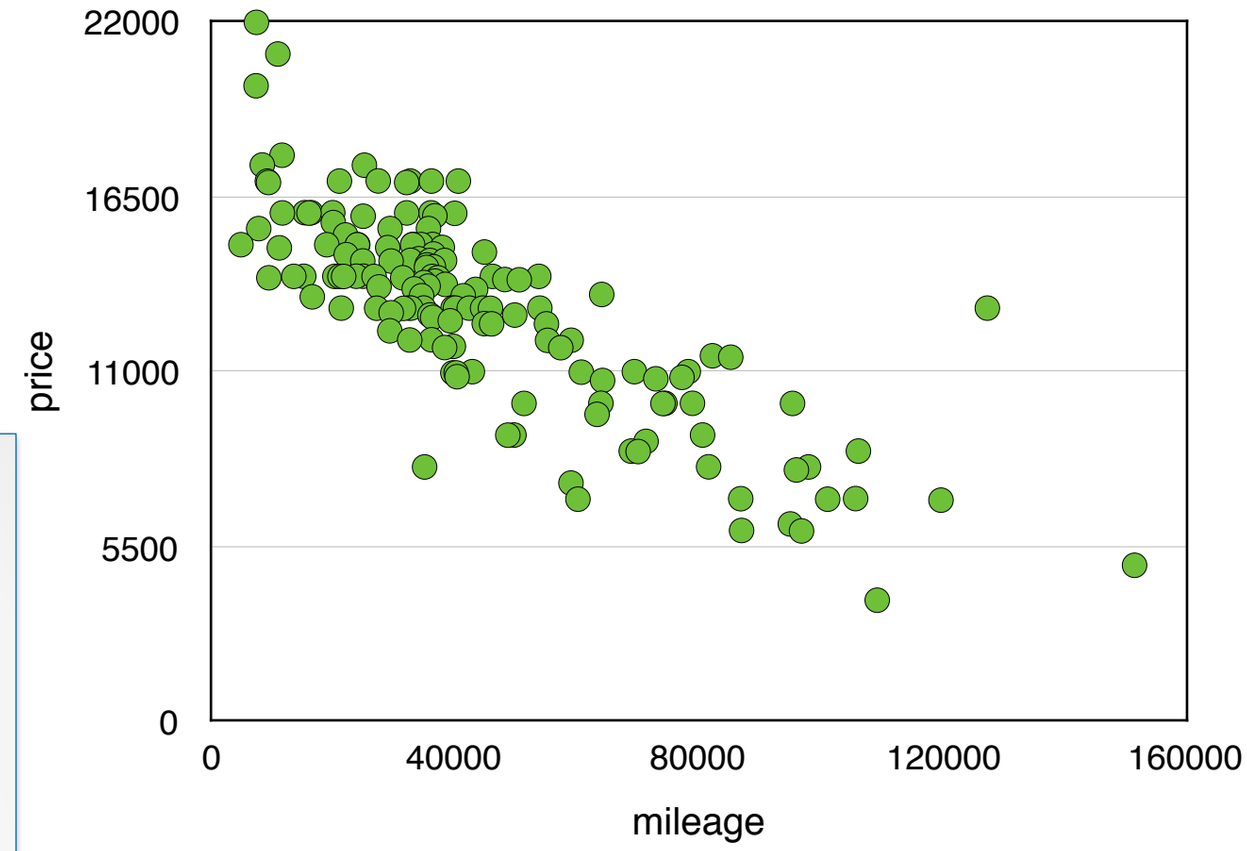
Example t-test

- compute the sum of the squared error, which tells us how close the fit is to the actual data

$$t = \frac{\hat{\beta}_1 - 0}{s.e(\hat{\beta}_1)}$$
$$= \frac{-0.093}{0.00563} = -16.657$$

- is this significant? Ans: Yes
- we can accept H_0 with 95% confidence if:

$$-t_{\alpha/2,n-2} < t < t_{\alpha/2,n-2}$$
$$-1.976 < t < 1.976 \quad (\alpha = 5\%)$$



$$R^2 = \frac{SSR}{SST} = 0.650$$

- 65% of the total variability in the price is accounted for by the mileage of the car
- This indicates a reasonably strong linear relationship between the price at auction and the mileage on the odometer

Regression with categorical features

Now consider a problem where we would like to predict a response which takes the values {0, 1}.

For example, the variables could be age, gender cholesterol level and we want to predict whether the patient has heart disease => {0: yes, 1: no}.

Linear regression does not work well for this problem:

- based on a linear model of the parameters
- doesn’t work well outside the variable range
- doesn’t map predictions to categorial values

Logistic Regression

Logistic Regression addresses the problem of estimating a probability, P(Y = 1) for variable values X.

The logistic regression model uses a function, called the logistic function, to model P(Y = 1):

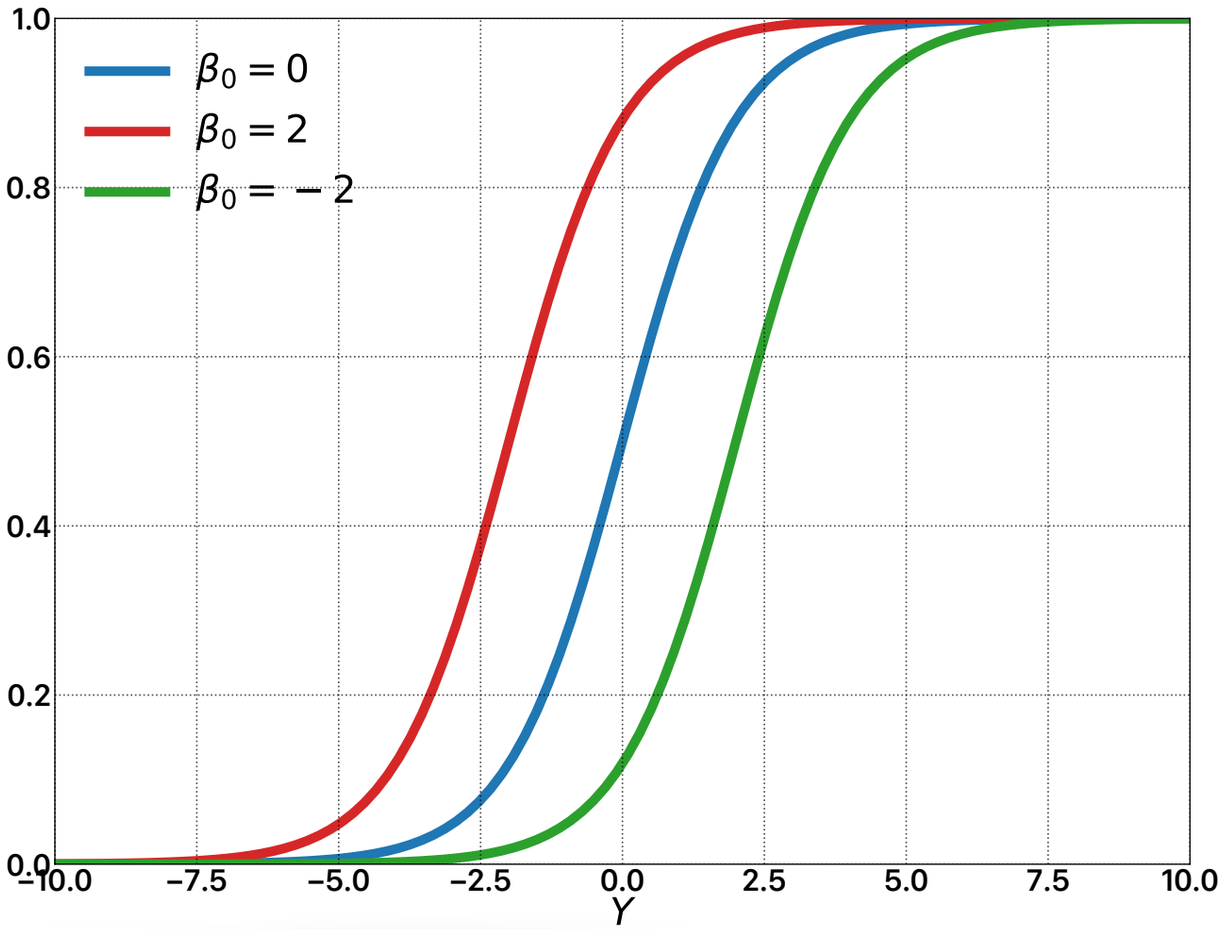
»

$$P(Y = 1) = \frac{\exp^{\beta_0 + \beta_1 X}}{1 + \exp^{\beta_0 + \beta_1 X}}$$

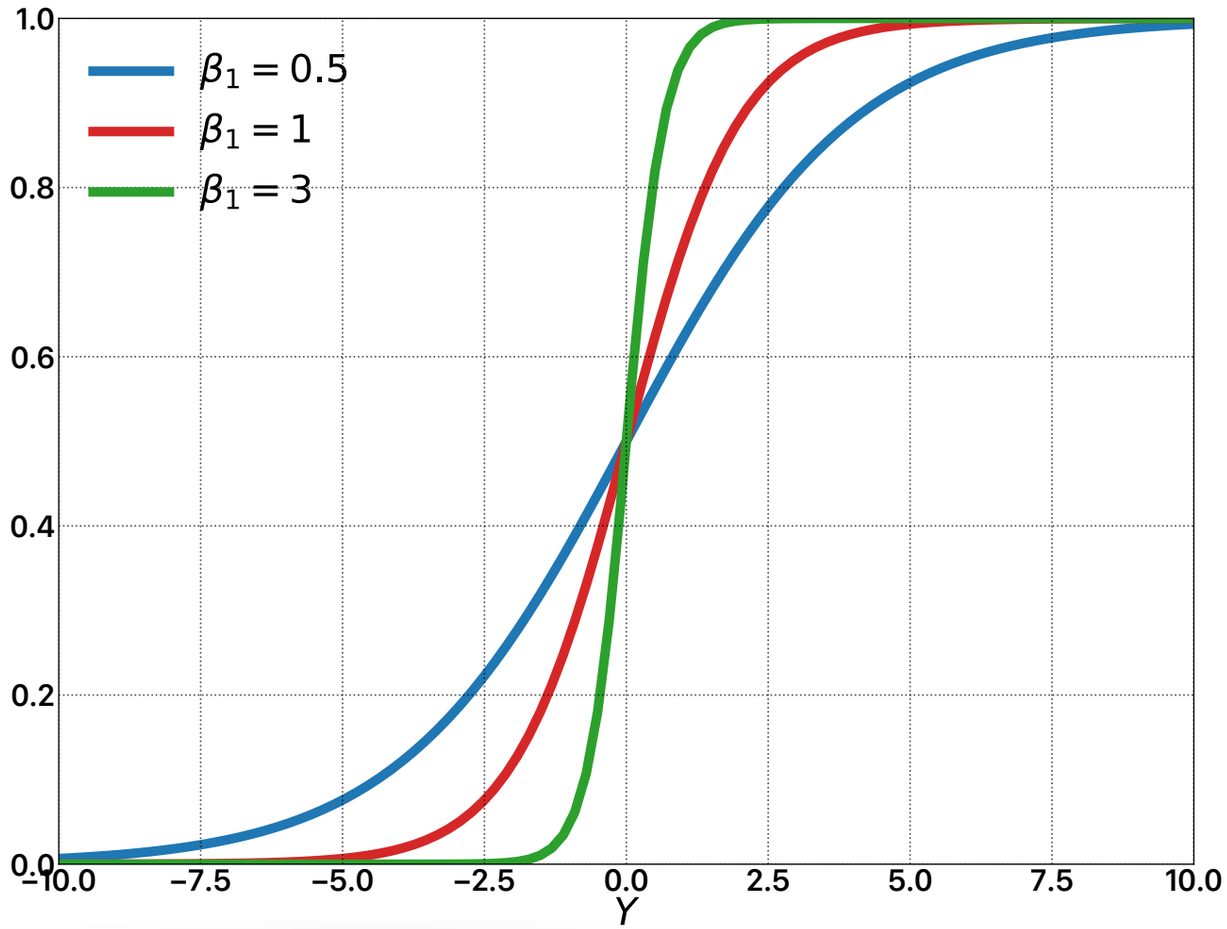
As a result the model will predict P(Y = 1) with an S-shaped curve which is the general shape of the logistic function.

- β_0 shifts the curve right or left
- β_1 controls how steep the S-shaped curve is.
- The logistic function is an example of an activation function

Logistic Regression



β_0 shifts the curve right or left



β_1 controls how steep the S-shaped curve is.

Logistic Regression

We can re-arrange the logistic formula:

$$\log\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X$$

where: $\text{odds ratio} = \left(\frac{P(Y=1)}{1-P(Y=1)}\right)$

when the odds ratio is greater than 1, it describes a ***positive*** relationship (eg. *as tumour size "increases,"* the odds of malignancy *increases*.

when the odds ratio is less than 1, it describes a ***negative*** relationship (eg. *as tumour size "decreases,"* the odds of malignancy de*creases*.

$$\text{odds ratio} = \frac{\text{odds(malignant)}}{\text{odds(benign)}}$$

logistic regression is said to model the log-odds with a linear function of the predictors or features, X.

Interpretation:

- β_0 is the log-odds of a benign tumour, P(Y=0)
- $\beta_0 + \beta_1$ is the log-odds of a malignant tumour, P(Y=1)
- a one unit change in X is associated with a β_1 change in the log-odds of Y = 1;
- a one unit change in X is associated with an e^{β_1} change in the odds that Y = 1.

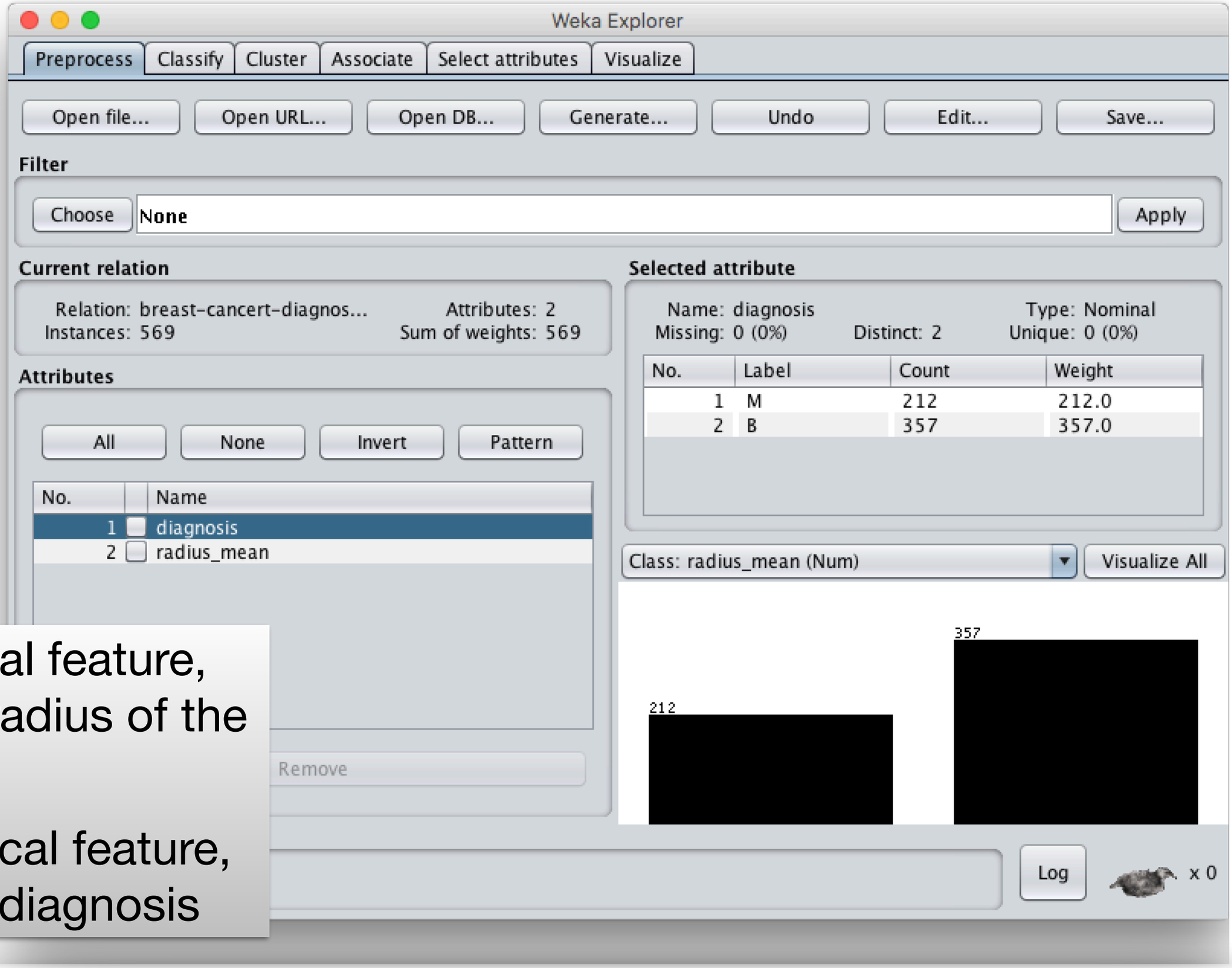
Logistic Regression- how to solve?

- Estimating Parameters:
- In simple linear regression there are closed form solutions for the estimated parameters β_i's.
- In Logistic regression there is no such closed-form solution
- It is common to use iterative and other techniques to find the best fit parameters

- Given independent observations of y, what is the likelihood function for p:

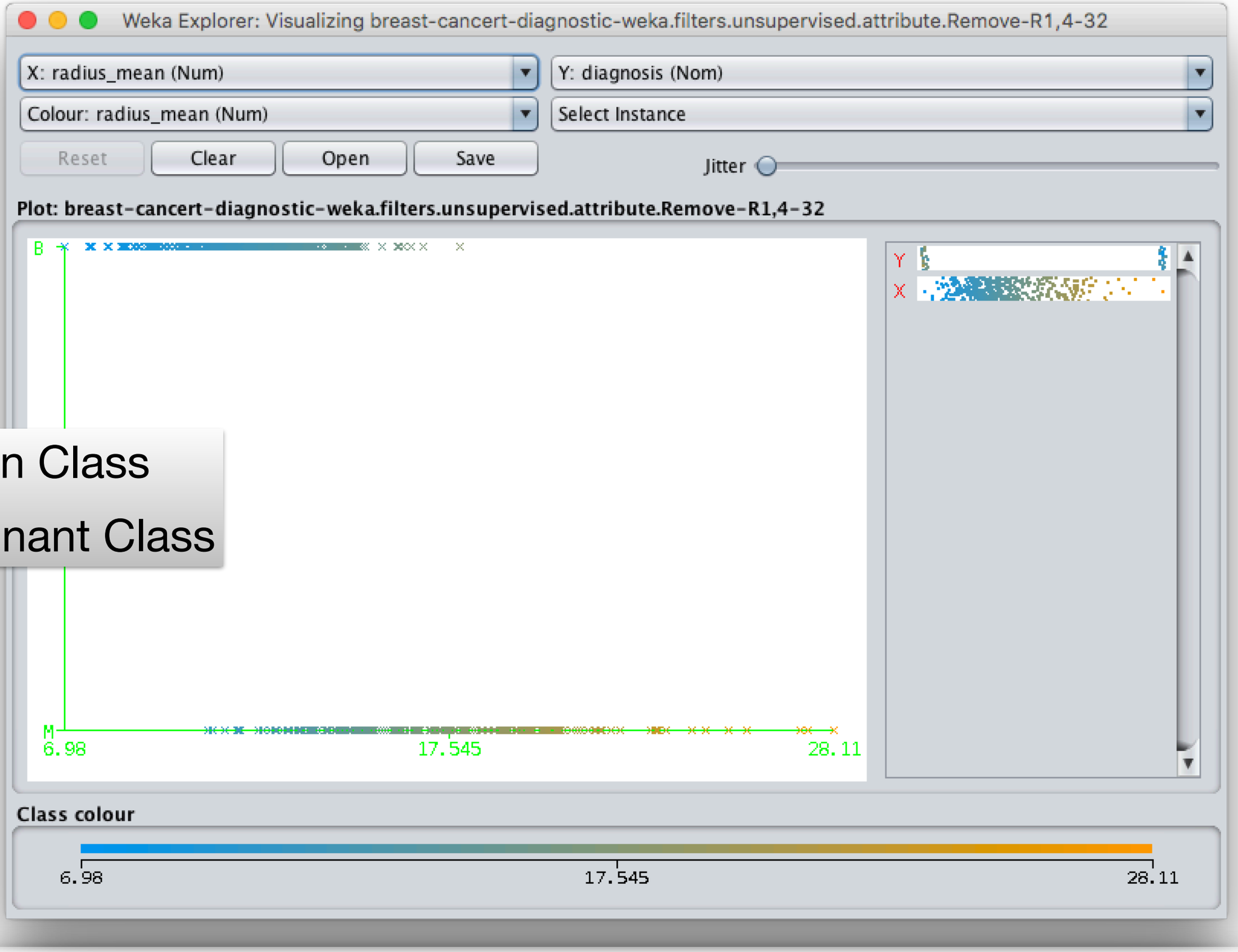
$$L(p|Y) = \prod P(Y_i = y_i) = \prod p_i^{y_i}(1 - p_i)^{(1-y_i)}$$

- we could try to take the log and differentiate to find the maximum, but this is messy. Iterative approaches are easier.



X: numerical feature, the mean radius of the tumour

Y: categorical feature, {M, B} the diagnosis



B: Benign Class

M: Malignant Class

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Classifier

al-places 4

ier output

```
Run information ===

:       weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
ion:    breast-cancert-diagnostic-weka.filters.unsupervised.attribute.Remove-R1,4-32
ces:    569
ibutes:   2
          diagnosis
          radius_mean
 mode:    evaluate on training data

Classifier model (full training set) ===

stic Regression with ridge parameter of 1.0E-8
ficients...
            Class
able            M
=================
us_mean    1.0336
rcept    -15.2459


 Ratios...
            Class
able            M
=====================
radius_mean    2.8111


Time taken to build model: 0.03 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances         500               87.8735 %
Incorrectly Classified Instances        69               12.1265 %
Kappa statistic                          0.7353
Mean absolute error                      0.1788
```

- weka
  - classifiers
    - bayes
    - functions
      - GaussianProcesses
      - Logistic
      - MultilayerPerceptron
      - SGD
      - SGDText
      - SimpleLinearRegression
      - SimpleLogistic
      - SMO
      - SMOreg
      - VotedPerceptron
    - lazy
    - meta
    - misc
    - rules
    - trees

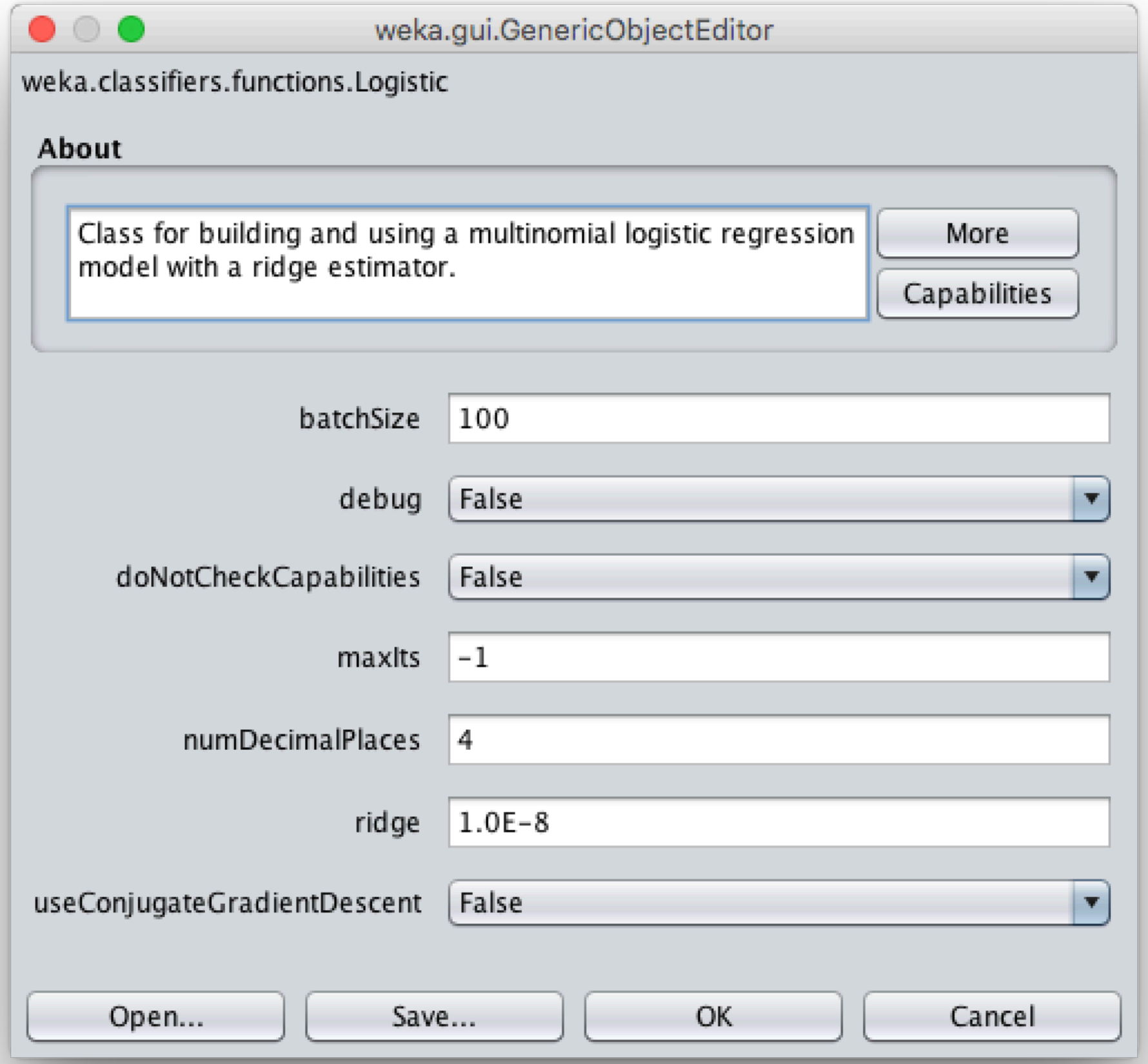
Close

Status

OK

Log

x 0


weka.gui.GenericObjectEditor
weka.classifiers.functions.Logistic
About
Class for building and using a multinomial logistic regression model with a ridge estimator.
More
Capabilities
batchSize
100
debug
False
doNotCheckCapabilities
False
maxIts
-1
numDecimalPlaces
4
ridge
1.0E-8
useConjugateGradientDescent
False
Open...
Save...
OK
Cancel

Logistic Regression

Interpreting the logistic regression model:

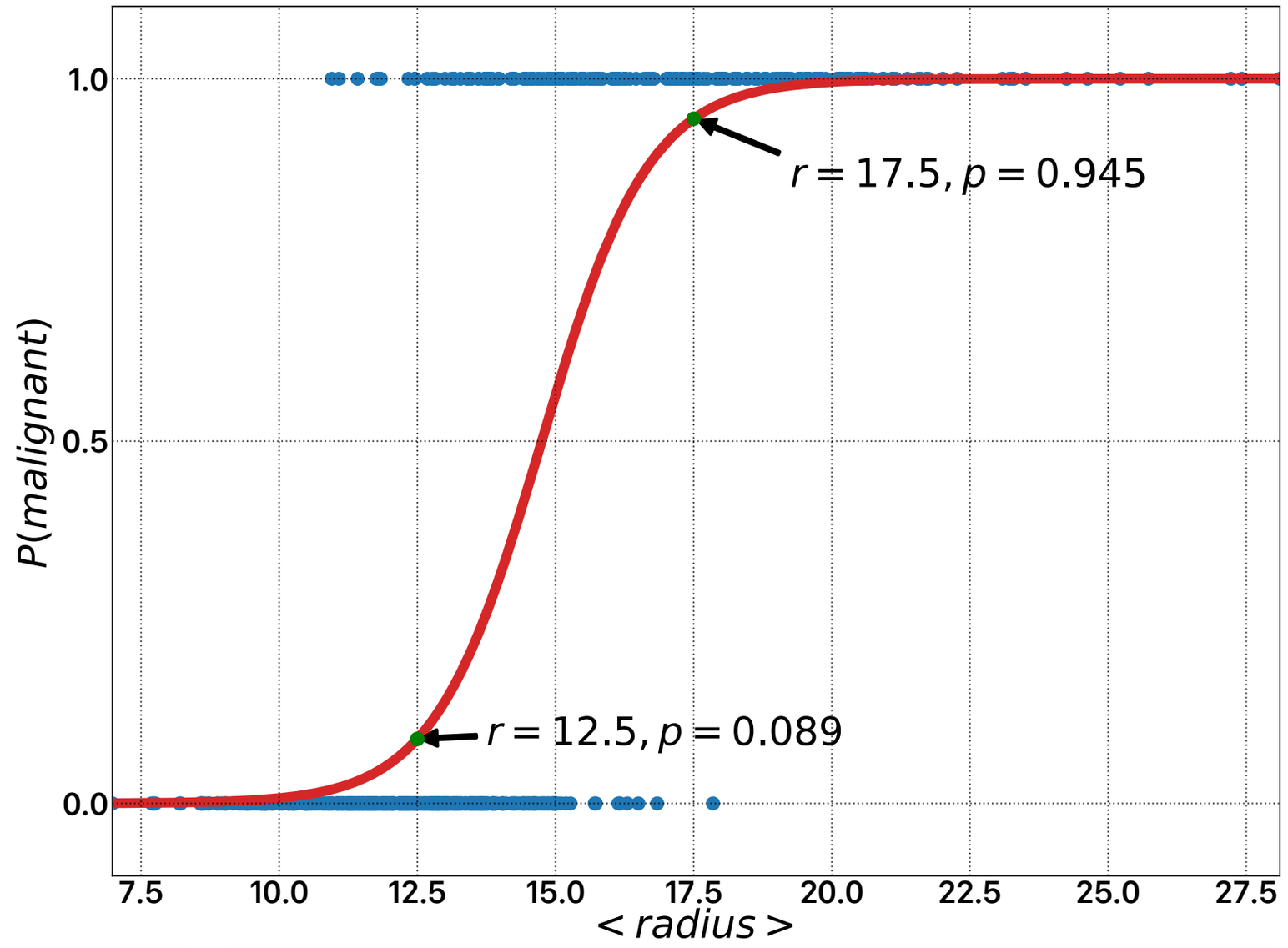
Logistic Regression

Coefficients...

Variable	M
radius_mean	1.0336
Intercept	-15.2459

Odds Ratios...

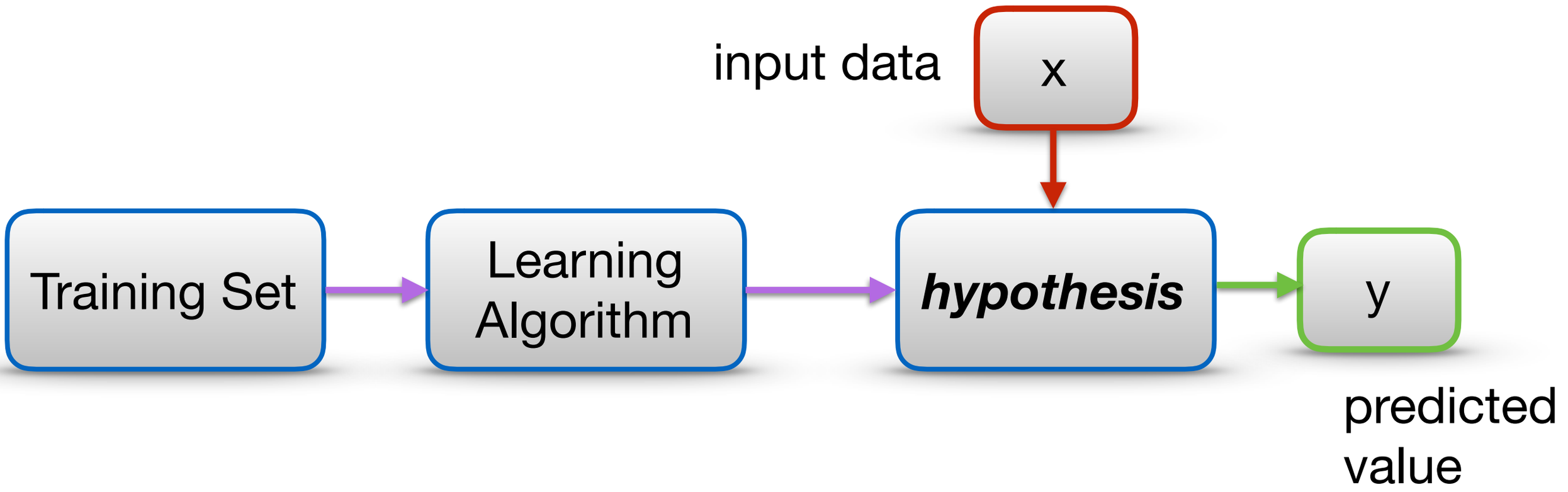
Variable	M
radius_mean	2.8111



$$P(\text{malignant}) = \text{logistic}(\langle \text{radius} \rangle, \beta_0 = -15.25, \beta_1 = 1.03)$$

Supervised Learning Problem

- Given a training set, we would like to learn a function h : X → Y so that h(x) is a “good” predictor for the corresponding value of y.
- the function h is called a hypothesis
- example:
- input data: tumour radius size
- output data: malignant or benign



Supervised Learning Problem

- we have already seen examples of the hypothesis function:

- linear regression:

$$h(x) = \beta_0 + \beta_1 x$$

- logistic regression:

$$h(x) = \frac{\exp^{\hat{\beta}_0 + \hat{\beta}_1 x}}{1 + \exp^{\hat{\beta}_0 + \hat{\beta}_1 x}}$$

What have we learned?

LEARNING

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

one feature

$$h(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots$$

$$h_\beta(X) = \sum_j \beta_j x_j$$

assume x_0=1

many features

Evaluation: *loss function* for penalising errors:

$$J(\beta) = \sum_{i=1} \left(h_\beta(x_i) - y_i \right)^2$$

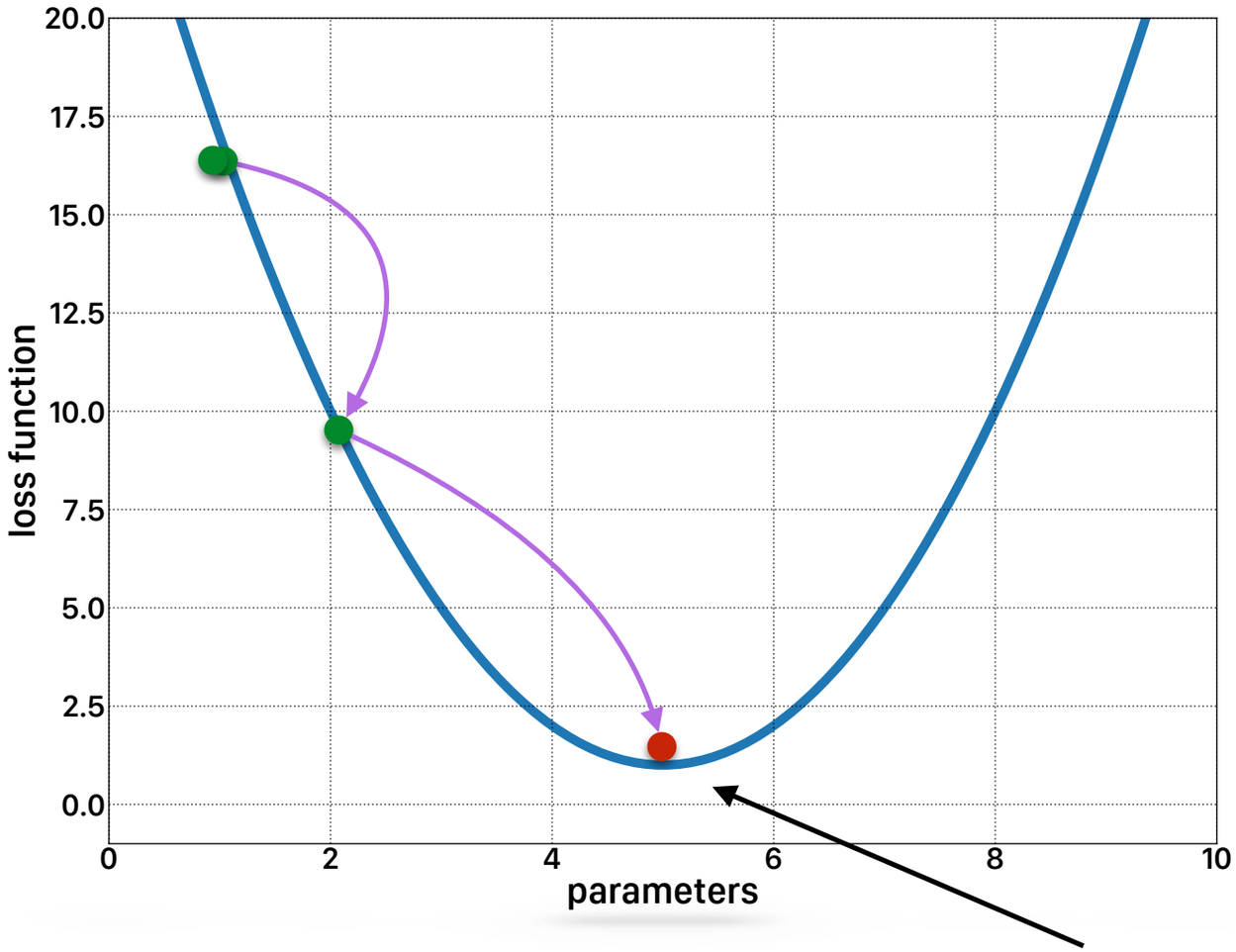
Optimisation:

$$\min_\beta J(\beta)$$

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:

$$\beta_j^{(i+1)} := \beta_j^{(i)} - \alpha \frac{\partial}{\partial \beta_j} J(\beta^{(i)})$$

- we start with some initial value of the parameters (could be randomly chosen or a reasonable first guess)
- the we compute the gradient of the loss function with respect to that parameter
- then adjust the parameter by a small amount (controlled by α), the opposite direction to the gradient (minus sign).
- By adjusting α, we can change how quickly we converge to the minimum. Large α -> risk of overshooting the minimum, small α -> 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:

$$\begin{aligned}\beta_0 :=& \beta_0 - \alpha \frac{\partial}{\partial \beta_0} J(\beta) \\ =& \beta_0 - \alpha \frac{\partial}{\partial \beta_0} \frac{1}{2n} \sum_{i=1}^{n} (h_\beta(x^{(i)}) - y^{(i)})^2 \\ =& \beta_0 - \frac{\alpha}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x^{(i)} - y^{(i)})\end{aligned}$$

$$\begin{aligned}\beta_1 :=& \beta_1 - \alpha \frac{\partial}{\partial \beta_1} J(\beta) \\ =& \beta_1 - \alpha \frac{\partial}{\partial \beta_1} \frac{1}{2n} \sum_{i=1}^{n} (h_\beta(x^{(i)}) - y^{(i)})^2 \\ =& \beta_1 - \frac{\alpha}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x^{(i)} - y^{(i)}) x^{(i)}\end{aligned}$$

factor of x here!

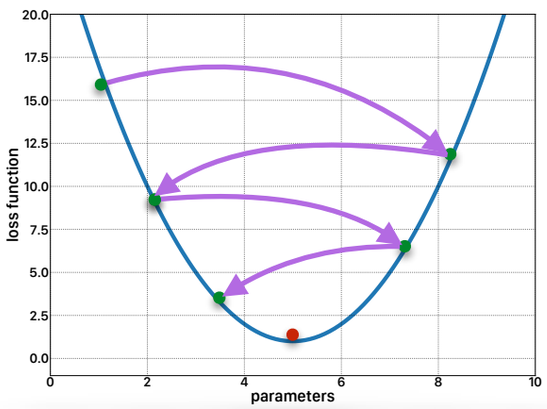
The update step for logistic regression is found by similar steps- using the derivative of the logistic function:

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

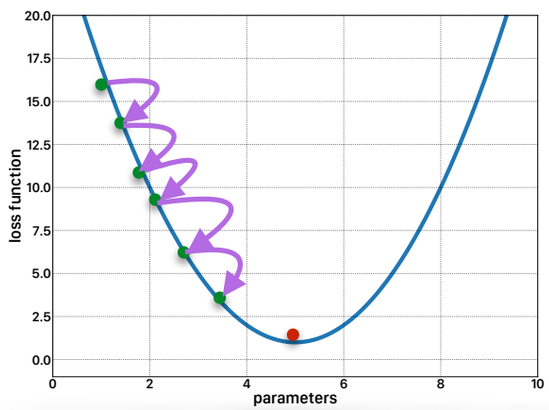
Gradient Descent

- The learning rate, α, determines the size of the adjustment made to each weight at each step in the process.
- What is the best value of α to choose?
    - There are many strategies to choosing the best value of α.
    - Most of the time we use rules of thumb, and also trial and error.
    - A typical range for α 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

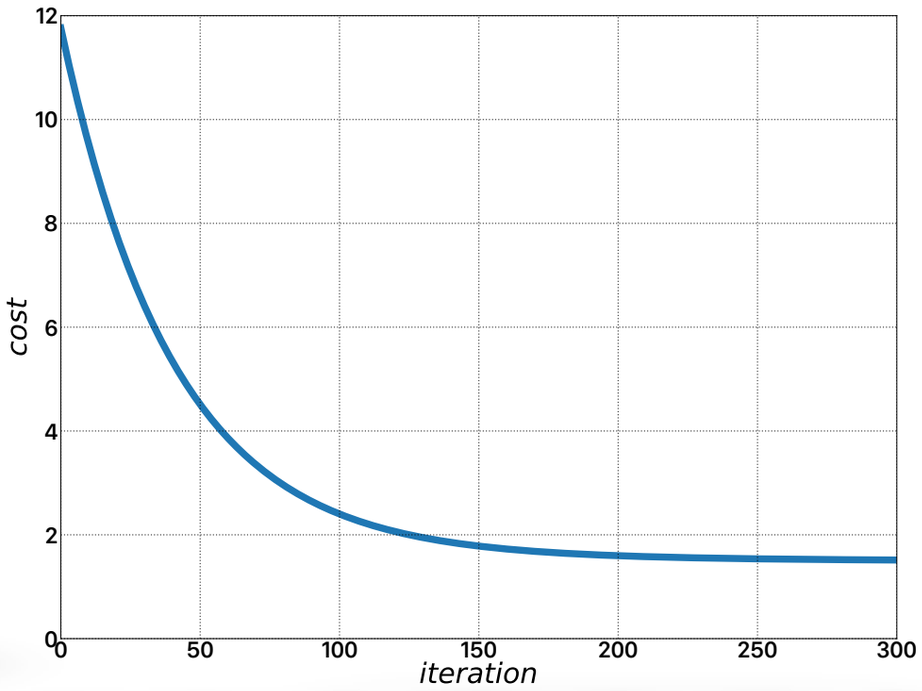


Gradient Descent

A basic algorithm for gradient descent:

1: **function** GRADIENTDESCENT(x, α, k_{max})
2: Require: Set of training instances D
3: Require: learning rate α
4: Require: Maximum number of iterations k_{max} or
5: other convergence criterion
6: $\beta_0 \leftarrow$ random point parameter space
7: $\beta_1 \leftarrow$ random point parameter space
8: **for** $k = 0$ to k_{max} **do**
9: $\beta_0^* = \beta_0 - \frac{\alpha}{n} \sum_{i=1}^{n} (h(x^{(i)}) - y^{(i)})$
10: $\beta_1^* = \beta_1 - \frac{\alpha}{n} \sum_{i=1}^{n} (h(x^{(i)}) - y^{(i)}) x^{(i)}$
11: $\beta_0 \leftarrow \beta_0^*$
12: $\beta_1 \leftarrow \beta_1^*$
13: **end for**
14: **end function**

updates are done after the sum!



We can compute the loss function, or cost at each iteration, and we see that it decreases monotonically.

Once the loss stops changing, we have done enough iterations.

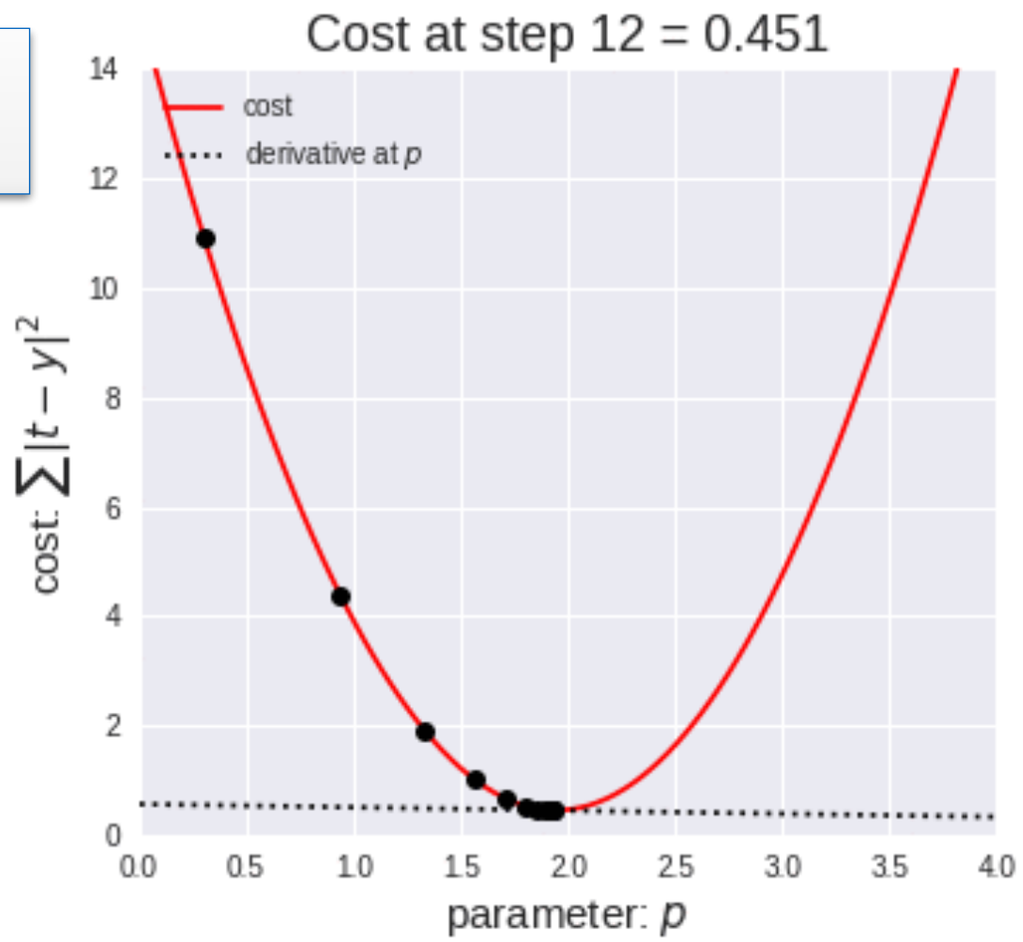
A sequence of steps of the algorithm, β_0=0, β_1=0, and α=0.01

i	β_0	β_1	$\frac{\alpha}{n} \sum_{i=1}^{n} (h(x^{(i)}) - y^{(i)})$	$\frac{\alpha}{n} \sum_{i=1}^{n} (h(x^{(i)}) - y^{(i)}) x^{(i)}$	cost
0	0.045043	0.023356	-0.045043	-0.023356	11.762546
1	0.089526	0.046440	-0.044482	-0.023084	11.512937
2	0.133454	0.069254	-0.043928	-0.022815	11.269426
3	0.176834	0.091803	-0.043380	-0.022549	11.031864
4	0.219674	0.114089	-0.042840	-0.022286	10.800105
5	0.261979	0.136116	-0.042306	-0.022027	10.574008

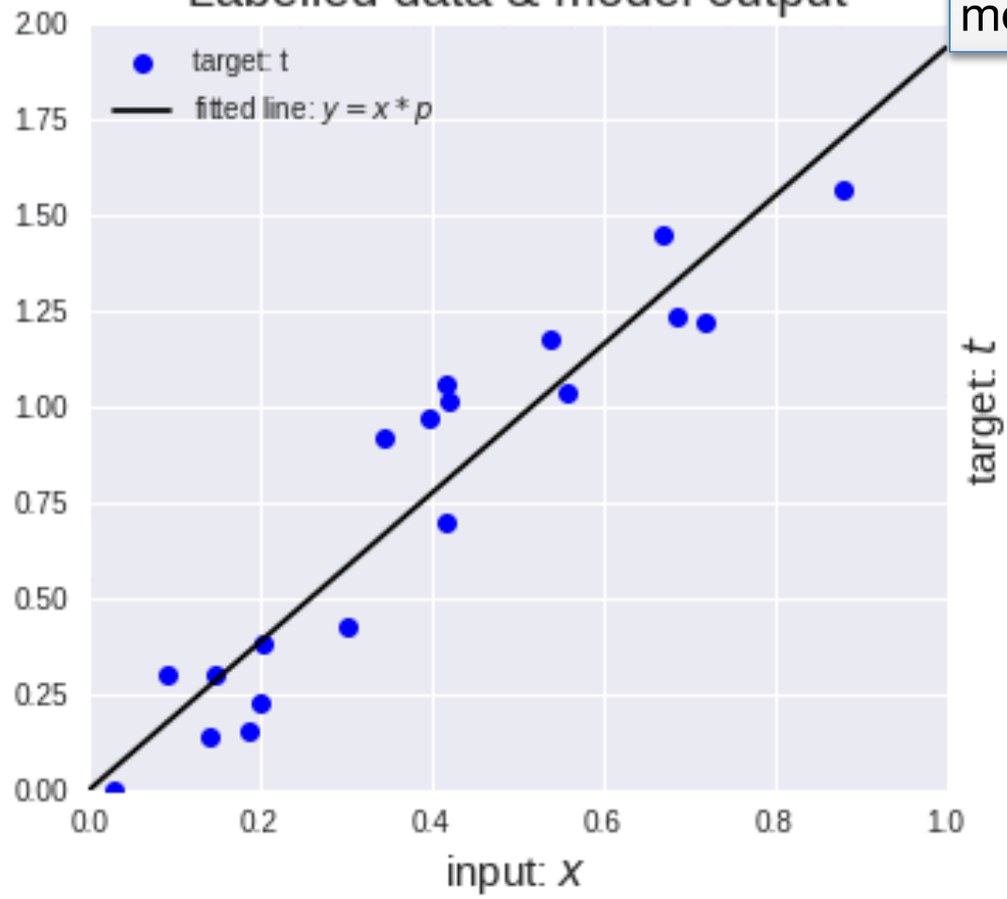
Bike rental dataset: moodle link

Gradient Descent in action

cost function



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:

$$\mathcal{E}(\beta) = \frac{1}{N}\sum_{i}^{N} J^i(\beta) = \frac{1}{N}\sum_{i}^{N} \frac{1}{2N}(h_\beta(x_i) - y_i))^2$$

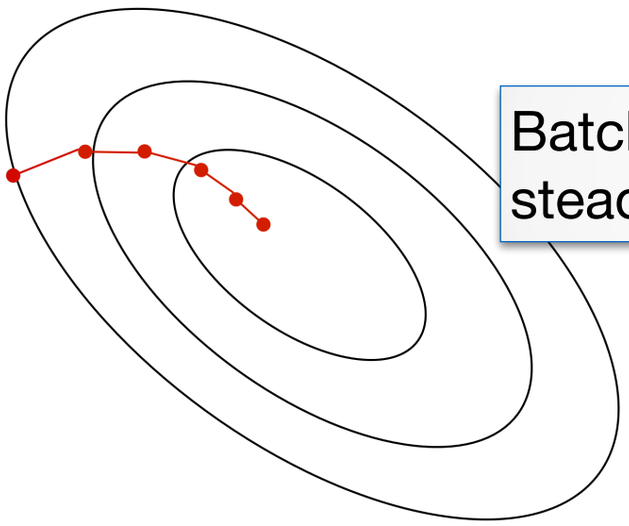
- this is called **Batch** Gradient Descent, because we need to sum over all training examples before we can make the updates.
- The problem with Batch GD is that it is very slow for large datasets!

Stochastic Gradient Descent

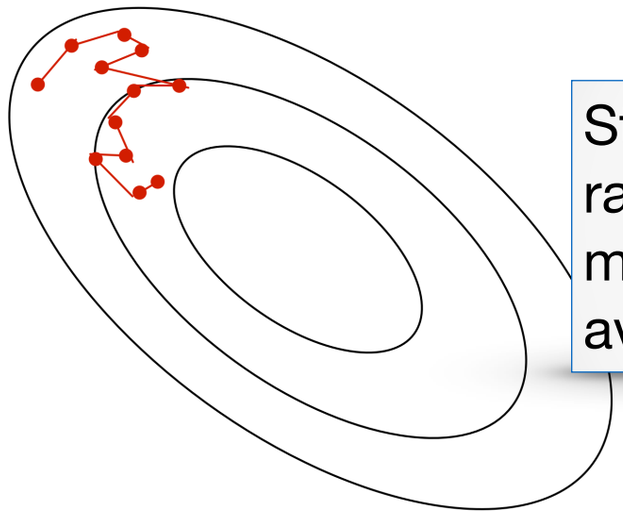
- in Stochastic GD we choose a single training example (perhaps at random), compute the loss and do the update directly:

$$\beta_j^{(i+1)} := \beta_j^{(i)} - \alpha \frac{\partial}{\partial \beta_j} J(\beta^{(i)})$$

- stochastic GD can surprisingly good progress with even a very small number of training examples.
- should converge closely to batch GD.
- But one 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

Simple Linear Regression

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 1 -C -R 1.0E-8 -additional-s
Relation:     bike_sharing-weka.filters.unsupervised.attribute.Remove-R1-10,12-15
Instances:    731
Attributes:   2
              atemp
              cnt
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

cnt =

   7501.8339 * atemp +
    945.824
```

bike rentals $= 7501.8339 \times temperature + 945.824$

Gradient Descent

500 iterations

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.functions.SGD -F 2 -L 0.01 -R 1.0E-4 -E 500 -C 0.001 -N -S 1
Relation:     bike_sharing-weka.filters.unsupervised.attribute.Remove-R1-10,12-15
Instances:    731
Attributes:   2
              atemp
              cnt

Loss function: Squared loss (linear regression)

cnt =

      7495.5297 atemp
 +    1020.4324
```

bike rentals $= 7495.5297 \times temperature + 1020.4324$

Summary

We looked at simple linear regression models, least squares fit and interpretation

We looked at logistic regression models for categorical features

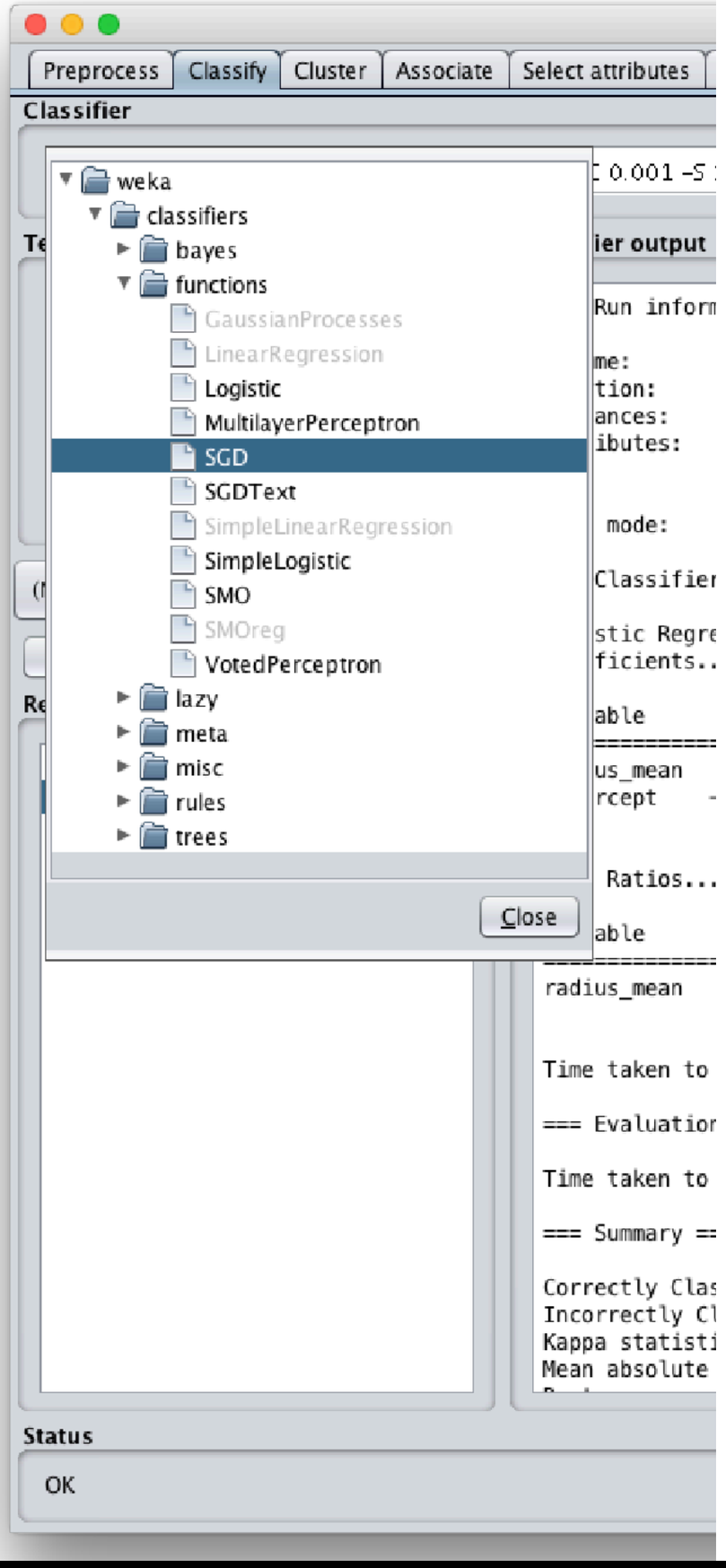
We looked at Gradient Descent to learn the parameters of a simple linear regression model.

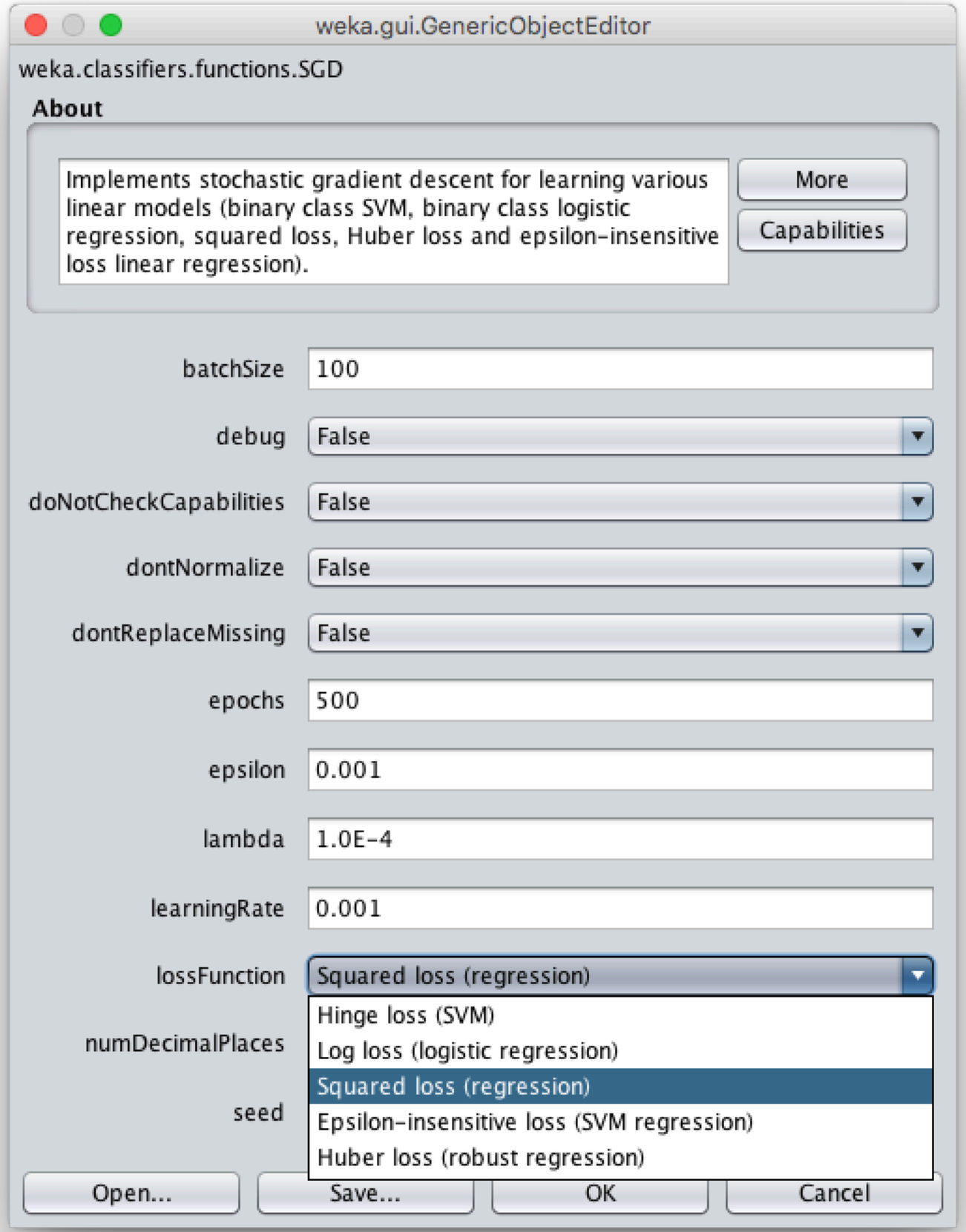
Gradient Descent is an optimisation algorithm used to find the values of parameters of a function that minimises a cost function.

Gradient descent is best used when the parameters cannot be calculated analytically, and must be searched for by an optimisation algorithm (eg. logistic regression)

Gradient Descent is one of the key algorithms in Machine Learning.

GD in Weka


Preprocess
Classify
Cluster
Associate
Select attributes
Classifier
weka
classifiers
bayes
functions
GaussianProcesses
LinearRegression
Logistic
MultilayerPerceptron
SGD
SGDText
SimpleLinearRegression
SimpleLogistic
SMO
SMOreg
VotedPerceptron
lazy
meta
misc
rules
trees
Close
Status
OK


weka.gui.GenericObjectEditor
weka.classifiers.functions.SGD
About
Implements stochastic gradient descent for learning various linear models (binary class SVM, binary class logistic regression, squared loss, Huber loss and epsilon-insensitive loss linear regression).
More
Capabilities
batchSize
100
debug
False
doNotCheckCapabilities
False
dontNormalize
False
dontReplaceMissing
False
epochs
500
epsilon
0.001
lambda
1.0E-4
learningRate
0.001
lossFunction
Squared loss (regression)
Hinge loss (SVM)
Log loss (logistic regression)
Squared loss (regression)
Epsilon-insensitive loss (SVM regression)
Huber loss (robust regression)
numDecimalPlaces
seed
Open...
Save...
OK
Cancel