Ollscoil na Gaillimhe
University of Galway

# CT4101
## Machine Learning

**Dr. Frank Glavin**
Room CSB-3004, Computer Science Building
Frank.Glavin@University*of*Galway.ie

School of Computer Science

University *of* Galway.ie

# Regression tasks – recap

To date we have mainly looked at one form of supervised learning task, namely classification, where the goal is to predict one class from a finite number of possible discrete classes
In regression tasks, we also have labelled training and testing data, but the labels take the form of floating-point values (real numbers)

Our goal is then to **predict a floating-point number**, rather than a class
E.g., predict the compressive strength of a concrete mix (dependent variable), based on the values of % water content, %cement, %additives, % coarse aggregate, %fine aggregate etc. (independent variables).

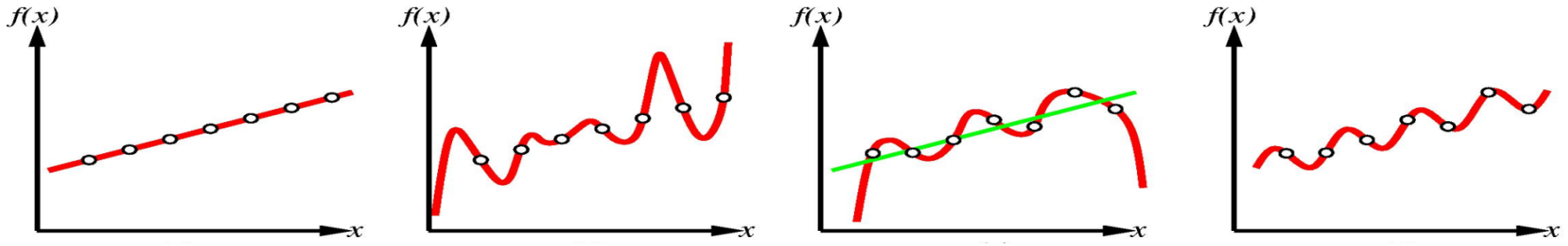
A training set for such a task might consist of hundreds of examples of compressive strength values for concrete cubes obtained from destructive testing (the labels), along with the proportions of the mixture (independent variable values) for each cube that was tested

# Examples of algorithms for regression tasks

- Linear regression
- Decision trees
- *k*-nearest neighbours
- Neural networks (more on this topic later in the module)

# Supervised Learning Considerations [1]

Various hypotheses can be consistent with observations but inconsistent with each other:
Which one should we choose?




(1) Data with exact linear hypothesis

(2) Same data: Exact 7th-Order polynomial hyp.

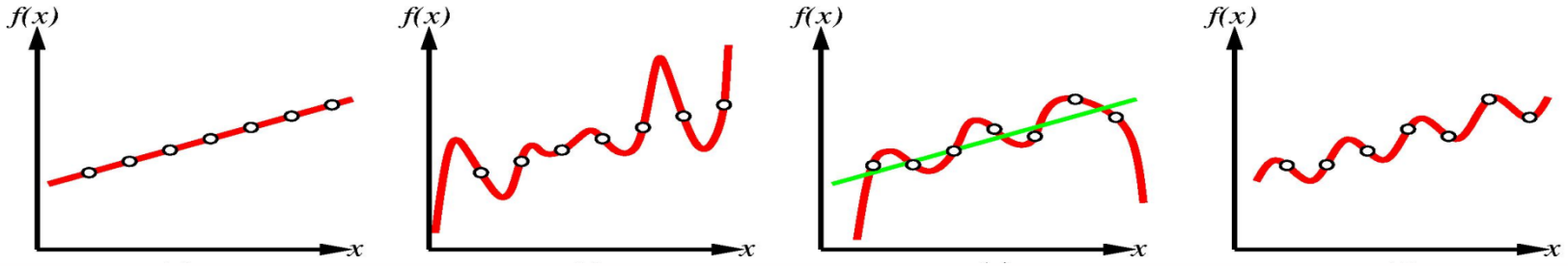
(3) Different data: Exact 5th-order poly and approx. linear hyp.

(4) Same data: Exact sinusoidal hypothesis

General form of a polynomial: $f(x) = a + bx + cx^2 + dx^3 + \cdots$

# Supervised Learning Considerations [2]

Various hypotheses can be consistent with observations but inconsistent with each other:
Which one should we choose?




(1) Data with exact linear hypothesis

(2) Same data: Exact 7th-Order polynomial hyp.

(3) Different data: Exact 5th-order poly and approx. linear hyp.

(4) Same data: Exact sinusoidal hypothesis

One solution: Ockham's Razor:

- Prefer *simplest* hypothesis consistent with data
- Definitions of simplicity (& consistency) may be subject to debate
- Depends strongly on how hypotheses are expressed

# Supervised Learning Considerations [3]

Hypothesis language is too limited?

- Might be **unable** to find hypothesis that exactly matches 'true' function
- If true function is more complex than what hypothesis can express, it will **underfit** the data
- Saw this in previous slide, 3rd and 4th figures

Hypothesis language cannot exactly match true function?

- there will be a trade-off between **complexity** of hypothesis and how well it **fits the data**

# Supervised Learning Considerations [4]

Hypothesis language is very **expressive**?

- Its search space is very large and the **computational complexity** of finding a good hypothesis will be high
- Also need a large amount of data to avoid **overfitting**

As we saw in the section on classification, algorithms express hypotheses differently

In general, would like to use an algorithm for a problem that can express the true underlying function

# Supervised Learning Considerations [5]

But don't forget:
*we never know the true underlying function*
E.g., To avoid problem with poorly fitting data

Could change algorithm so that, as well as searching for coefficients of polynomials, it tries combinations of trig. functions (sin, cos, tan)

Learning problem will become enormously more complex, but will it solve our problems?

**Probably not**: you could easily think up some different kind of mathematical function, to generate a new dataset that the algorithm still cannot represent perfectly.

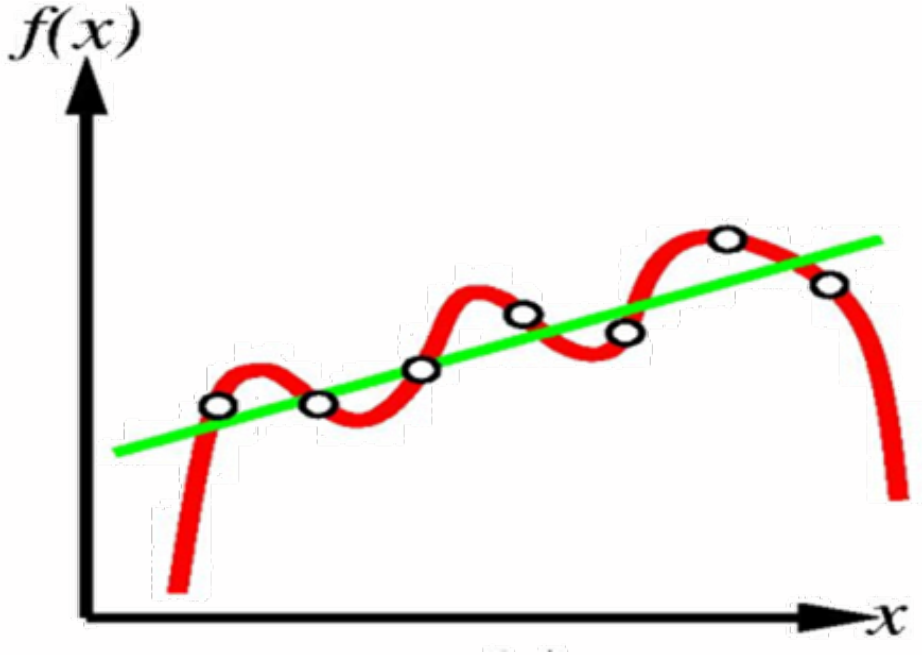
For this reason, often use relatively simple hypothesis languages, in the absence of special knowledge about domain

- more complex languages don't come with any real guarantees
- more simple languages correspond to easier searching

# Noise, Overfitting and Underfitting [1]

NOISE:
imprecise or incorrect attribute values or labels

- Can't always quantify it, but should know from situation if it is present
- E.g. labels may require subjective judgement or values may come from imprecise measurements

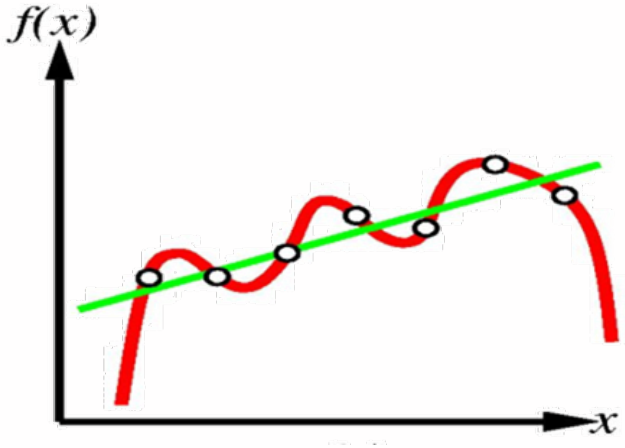


# Noise, Overfitting and Underfitting [2]

If the data might have noise, harder to decide which hypothesis is best:

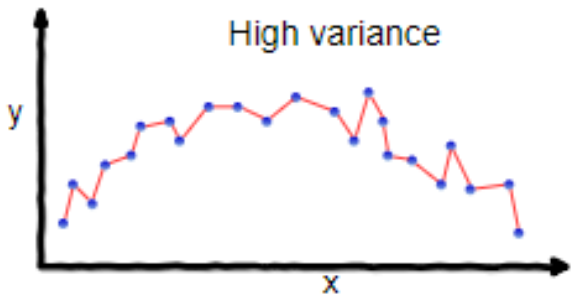
- Linear hypothesis could not fit to it, but polynomial could
- But which would really be the better choice?

Complex methods prone to **overfitting**; simple ones prone to **underfitting**

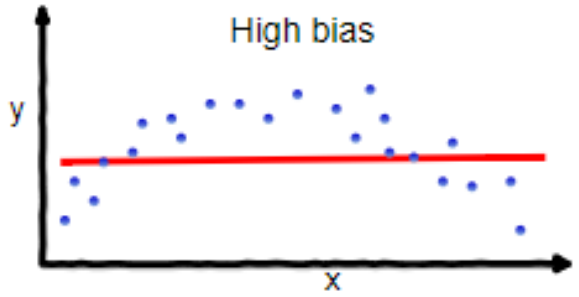



If you increase complexity of hypothesis, you increase ability to fit to the data, but might also increase risk of overfitting

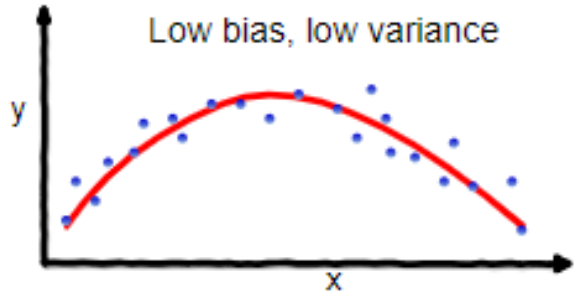
# Bias and Variance




overfitting




underfitting




Good balance

https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229


# Regression: Linear Regression Models

University *of* Galway.ie

# Example regression dataset

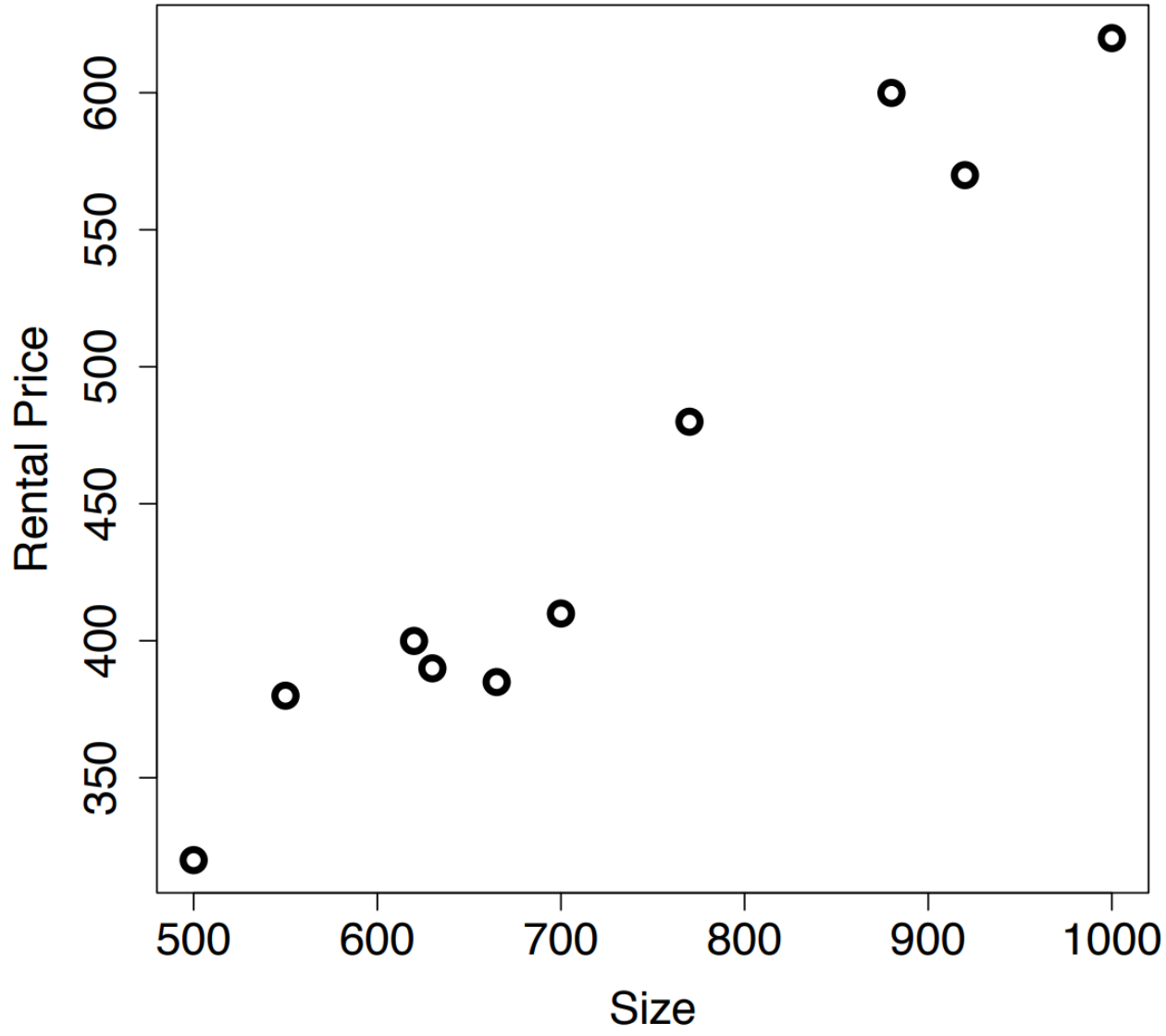
**Table:** The **office rentals dataset**: a dataset that includes office rental prices and a number of descriptive features for 10 Dublin city-centre offices.

| ID | Size | Floor | Broadband Rate | Energy Rating | Rental Price |
|---|---|---|---|---|---|
| 1 | 500 | 4 | 8 | C | 320 |
| 2 | 550 | 7 | 50 | A | 380 |
| 3 | 620 | 9 | 7 | A | 400 |
| 4 | 630 | 5 | 24 | B | 390 |
| 5 | 665 | 8 | 100 | C | 385 |
| 6 | 700 | 4 | 8 | B | 410 |
| 7 | 770 | 10 | 7 | B | 480 |
| 8 | 880 | 12 | 50 | A | 600 |
| 9 | 920 | 14 | 8 | C | 570 |
| 10 | 1,000 | 9 | 24 | B | 620 |

Ollscoil na Gaillimhe
University of Galway

# Scatter plot of size vs. rental price

| ID | Size | Rental Price |
|---|---|---|
| 1 | 500 | 320 |
| 2 | 550 | 380 |
| 3 | 620 | 400 |
| 4 | 630 | 390 |
| 5 | 665 | 385 |
| 6 | 700 | 410 |
| 7 | 770 | 480 |
| 8 | 880 | 600 |
| 9 | 920 | 570 |
| 10 | 1,000 | 620 |



# Parameterised prediction models

A paramaterised prediction model is initialised with a set of random parameters and an error function is used to judge how well this initial model performs when making predictions for instances in a training dataset

Based on the value of the error function the parameters are iteratively adjusted to create a more and more accurate model.

This is the approach taken by many common ML models, e.g., simple linear regression (today's lecture) and neural networks (later lecture)

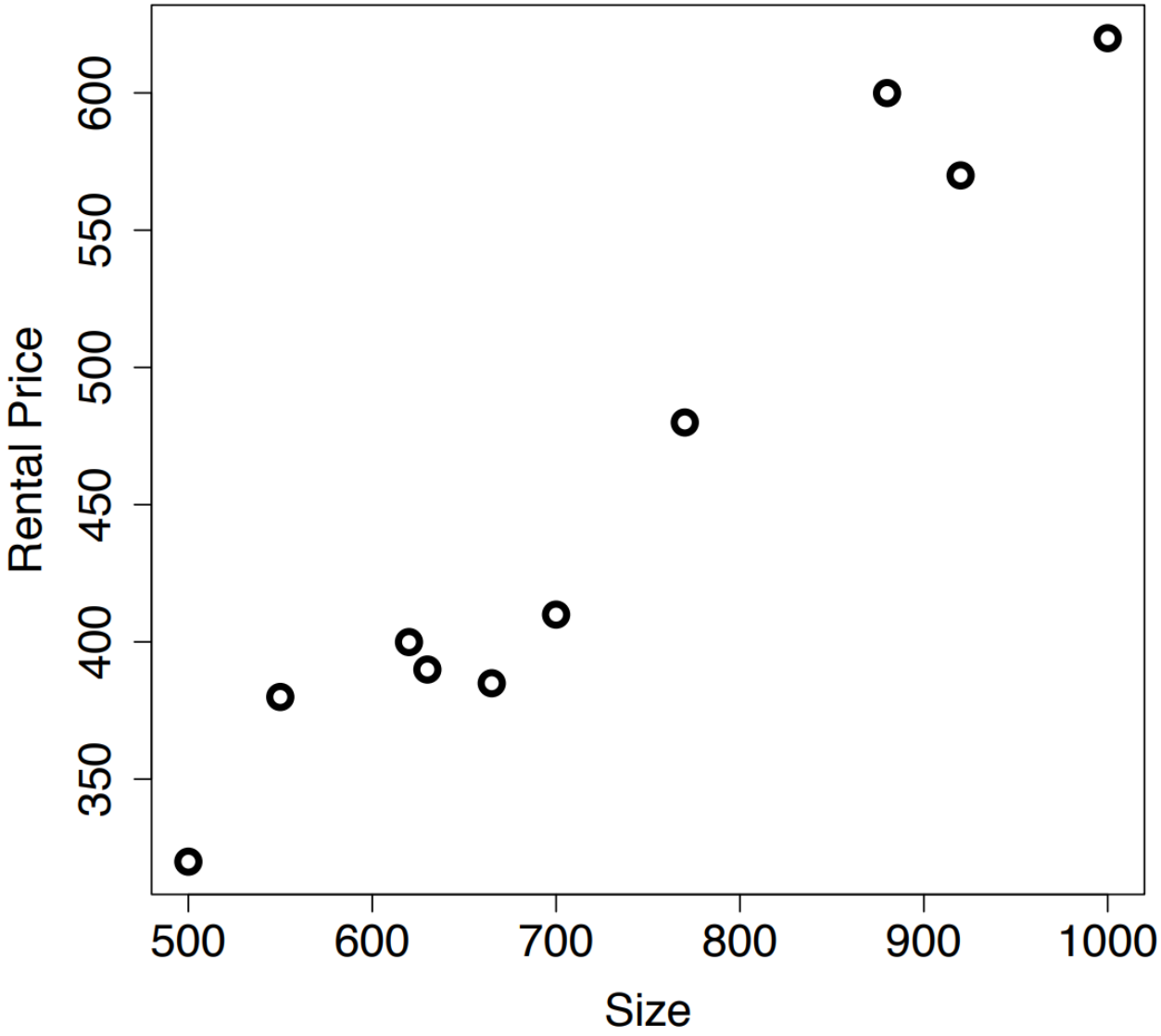
# Developing a simple linear regression model

From the scatter plot it appears that there is a linear relationship between the SIZE and RENTAL PRICE. The equation of a line can be written as:

$$y = mx + c$$

i.e., the equation of a straight line that you will all be familiar with from secondary school mathematics

**Recap**:https://thirdspacelearning.com/gcse-maths/algebra/y-mx-c/

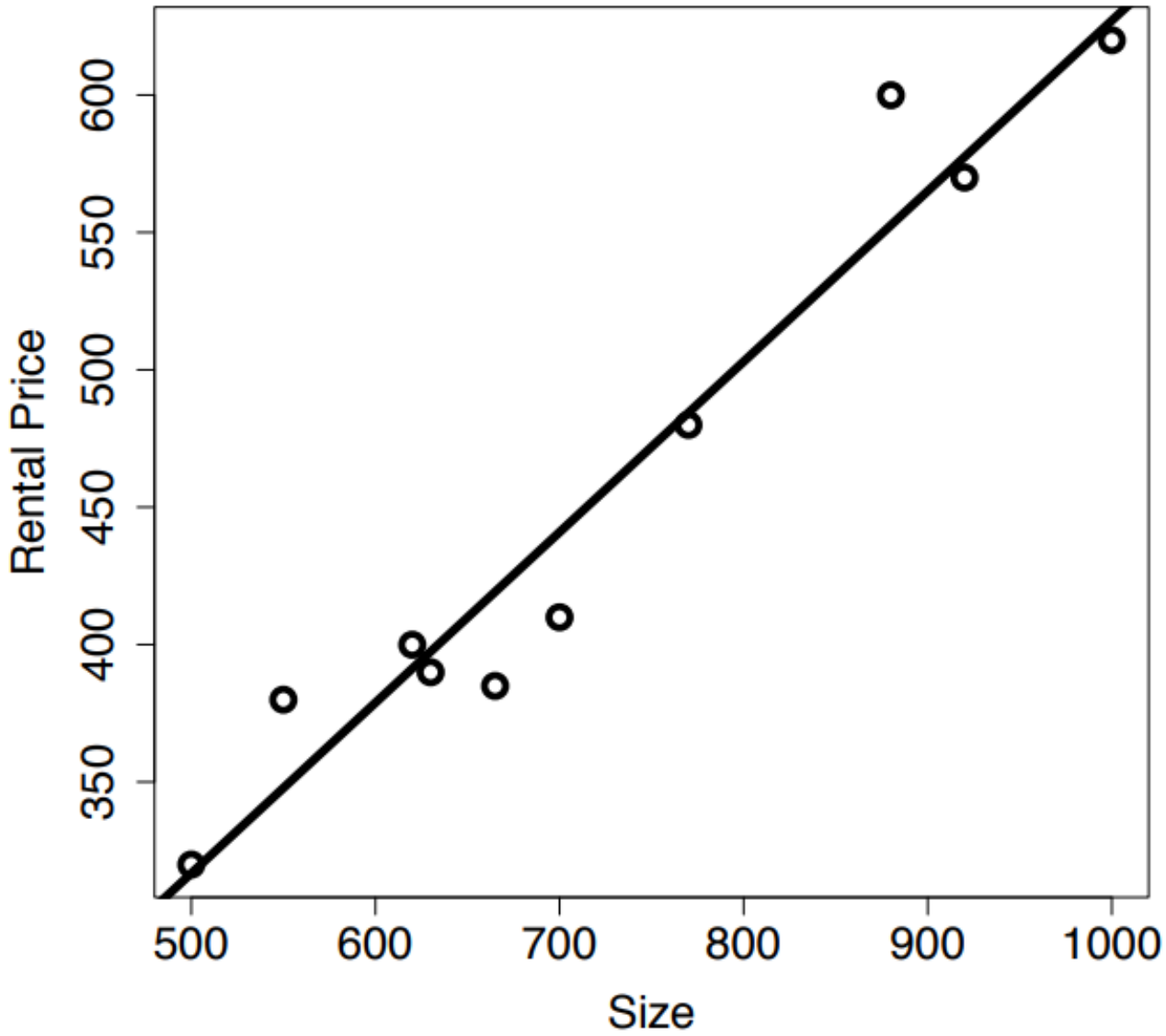


# Developing a simple linear regression model

The scatter plot on the right shows the same scatter plot as shown previously with a simple linear model added to capture the relationship between office sizes and office rental prices.

This model is:

$$Rental\_price = 6.47 + 0.62 \times Size$$



# Making a prediction using simple linear regression

Using this model determine the expected rental price of a 730 square foot office:

$$Renal\_price = 6.47 + 0.62 \times 730 = 459$$

# Multivariate linear regression using vector notation

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \mathbf{w}[0] \times \mathbf{d}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \cdots + \mathbf{w}[m] \times \mathbf{d}[m]$$

$$= \sum_{j=0}^{m} \mathbf{w}[j] \times \mathbf{d}[j]$$

$$= \mathbf{w} \cdot \mathbf{d}$$

**w** is a vector of model weights, *m* is the number of independent variables

**d** is a vector containing the independent variable values

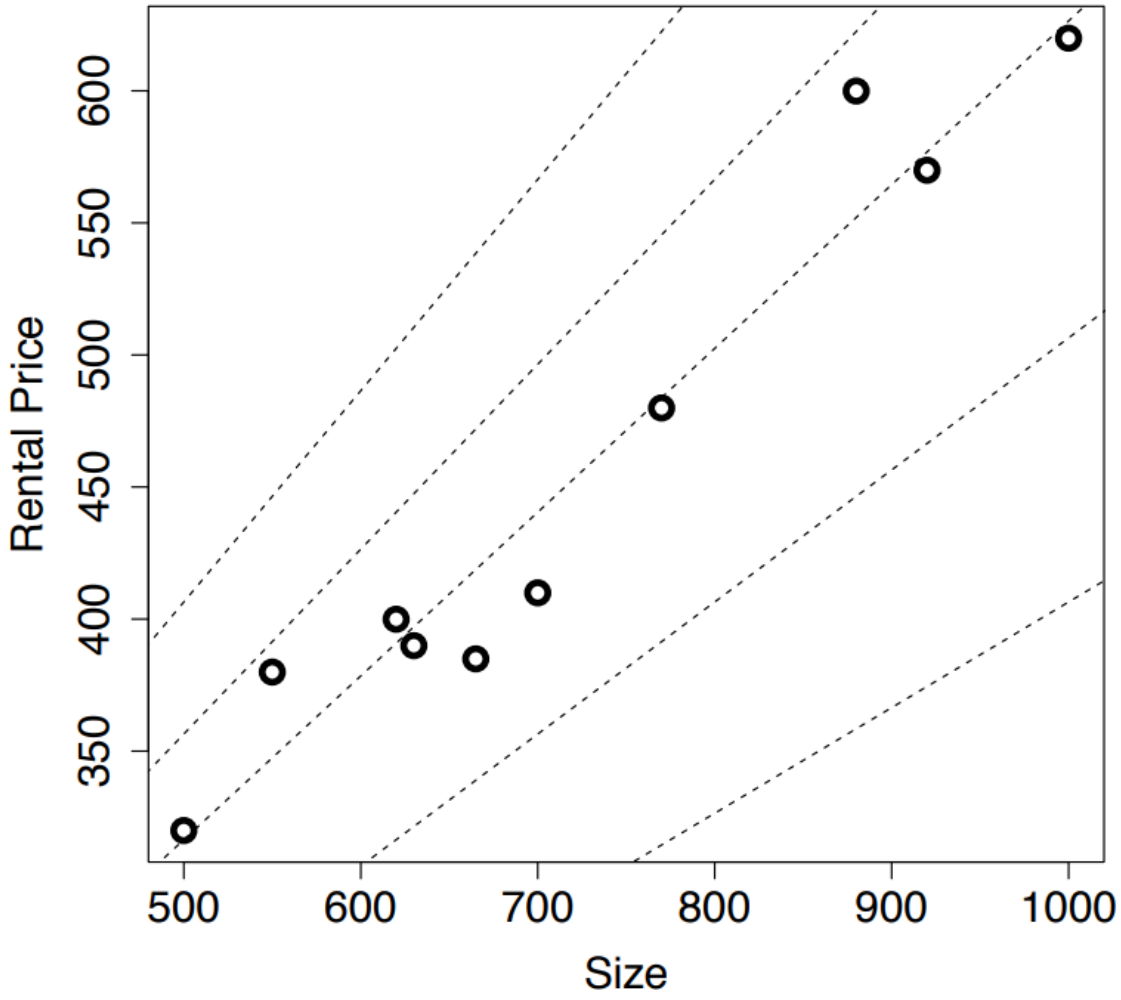
$\mathbb{M}_{\mathbf{w}}(\mathbf{d})$ is the predicted value, **w . d** is the vector dot product

Note that we include a dummy descriptive feature **d**[0] that is always equal to 1 to account for constant effects captured by **w**[0] (this is the intercept *c* in the equation of a straight line)

# Example simple linear regression models

A scatter plot of the SIZE and RENTAL PRICE features from the office rentals dataset.
For all models **w**[0] is set to 6.47
From top to bottom the models use 0.4, 0.5, 0.62, 0.7 and 0.8 respectively for **w**[1].

For linear regression in one independent variable, we have only two components in the weight vector, **w**[0] (intercept) and **w**[1] (slope), and we make predictions based on the value of one independent variable (Size in this case)

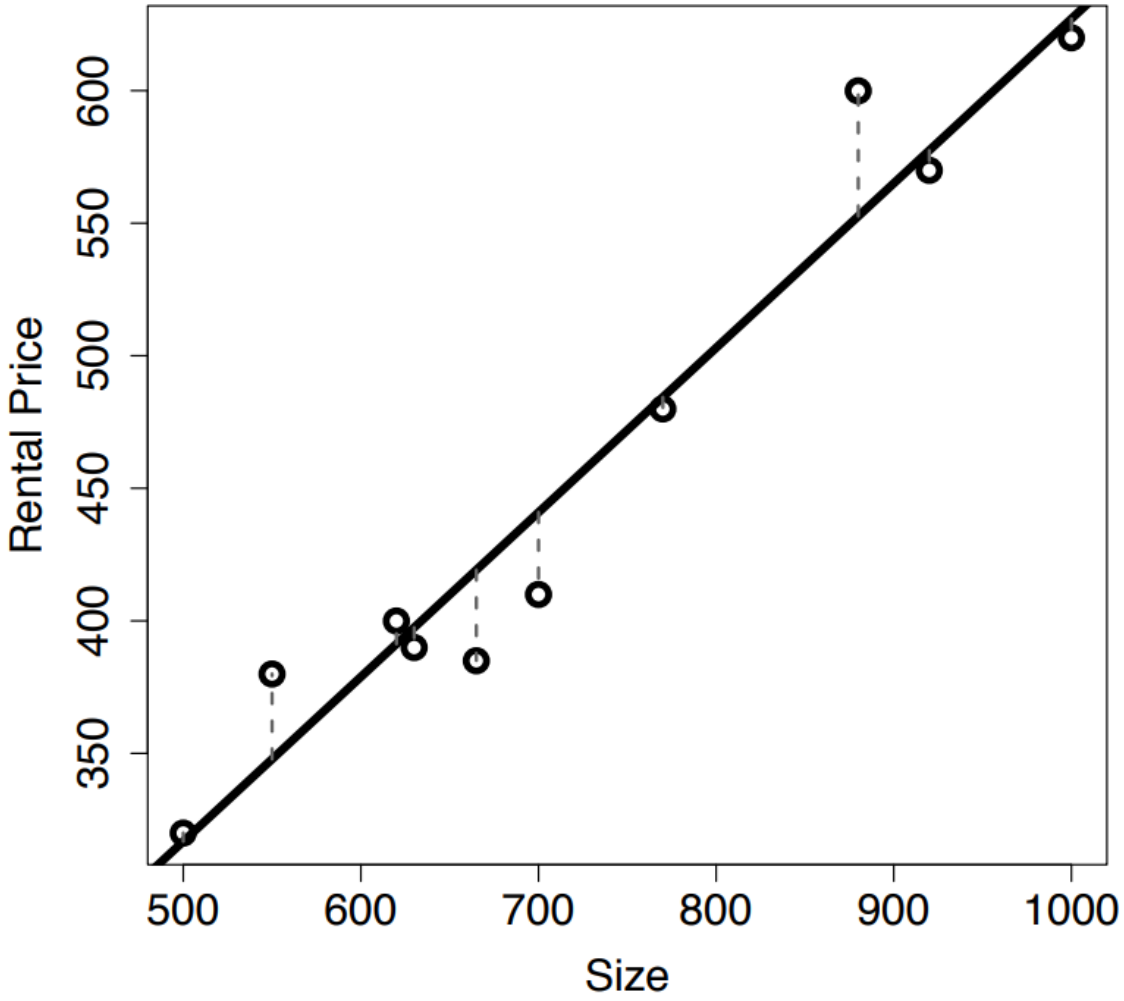


# Measuring error

A scatter plot of the SIZE and RENTAL PRICE features from the office rentals dataset showing a candidate prediction model (with **w**[0] = 6.47 and **w**[1] = 0.62) and the resulting errors.

Errors are measured between the predicted value and the target value (label)

Note that errors may be positive or negative (above or below the regression line)



# Sum of squared errors

$$\text{sum of squared errors} = \frac{1}{2}\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2$$

In order to formally measure the fit of a linear regression model with a set of training data, we require an error function that captures the error between the predictions made by a model and the actual values in a training dataset

Here $t_i \ldots t_n$ is the set of n target values and $\mathbb{M}(d_1) \ldots \mathbb{M}(d_n)$ is the set of predictions

By minimising the sum of squared errors or $L_2$, we can develop a best fit linear regression model

Note that some errors are + and some -. If we simply add these together, + and - errors would cancel each other out. This is why, rather than just using the sum of the errors, we use the sum of the squared errors because this means all values will be positive.

# Example sum of squared errors calculation

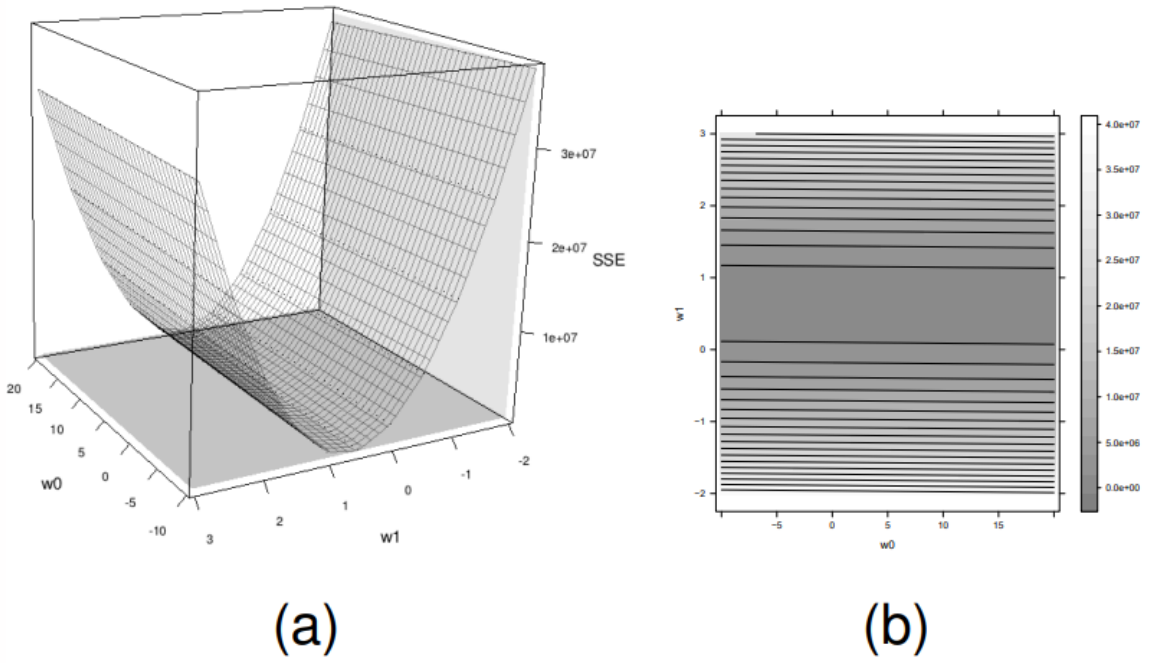
Calculating the sum of squared errors for the candidate model with **w**[0] = 6.47 and **w**[1] = 0.62 making predictions for the office rentals dataset.

| ID | Rental Price | Model Prediction | Error Error | Squared Error |
|---|---|---|---|---|
| 1 | 320 | 316.79 | 3.21 | 10.32 |
| 2 | 380 | 347.82 | 32.18 | 1,035.62 |
| 3 | 400 | 391.26 | 8.74 | 76.32 |
| 4 | 390 | 397.47 | -7.47 | 55.80 |
| 5 | 385 | 419.19 | -34.19 | 1,169.13 |
| 6 | 410 | 440.91 | -30.91 | 955.73 |
| 7 | 480 | 484.36 | -4.36 | 19.01 |
| 8 | 600 | 552.63 | 47.37 | 2,243.90 |
| 9 | 570 | 577.46 | -7.46 | 55.59 |
| 10 | 620 | 627.11 | -7.11 | 50.51 |
| | | | **Sum** | **5,671.64** |
| | | **Sum of squared errors (Sum/2)** | | **2,835.82** |

Ollscoil na Gaillimhe
University of Galway

# Error surfaces for simple linear regression

The x-y plane is known as the **weight space** and the surface is known as the error surface
The model that best fits the training data is the model corresponding to the lowest point on the error surface.
One approach to find this point is the **gradient descent algorithm** (which we will not cover due to time constraints)
The same concepts apply to multivariate linear regression, although error surfaces cannot easily be visualised



# Developing a multivariate model + handling categorical features

The basic structure of the multivariable linear regression model allows for only continuous descriptive features, so we need a way to handle categorical descriptive features.

The most common approach to handling categorical features uses a transformation that converts a single categorical descriptive feature into a number of continuous descriptive feature values that can encode the levels of the categorical feature.

| ID | Size | Floor | Broadband Rate | Energy Rating | Rental Price |
|---|---|---|---|---|---|
| 1 | 500 | 4 | 8 | C | 320 |
| 2 | 550 | 7 | 50 | A | 380 |
| 3 | 620 | 9 | 7 | A | 400 |
| 4 | 630 | 5 | 24 | B | 390 |
| 5 | 665 | 8 | 100 | C | 385 |
| 6 | 700 | 4 | 8 | B | 410 |
| 7 | 770 | 10 | 7 | B | 480 |
| 8 | 880 | 12 | 50 | A | 600 |
| 9 | 920 | 14 | 8 | C | 570 |
| 10 | 1,000 | 9 | 24 | B | 620 |

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \mathbf{w}[0] \times \mathbf{d}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \cdots + \mathbf{w}[m] \times \mathbf{d}[m]$$

Ollscoil na Gaillimhe
University of Galway

# Adjusted dataset

| ID | Size | Floor | Broadband Rate | Energy Rating A | Energy Rating B | Energy Rating C | Rental Price |
|---|---|---|---|---|---|---|---|
| 1 | 500 | 4 | 8 | 0 | 0 | 1 | 320 |
| 2 | 550 | 7 | 50 | 1 | 0 | 0 | 380 |
| 3 | 620 | 9 | 7 | 1 | 0 | 0 | 400 |
| 4 | 630 | 5 | 24 | 0 | 1 | 0 | 390 |
| 5 | 665 | 8 | 100 | 0 | 0 | 1 | 385 |
| 6 | 700 | 4 | 8 | 0 | 1 | 0 | 410 |
| 7 | 770 | 10 | 7 | 0 | 1 | 0 | 480 |
| 8 | 880 | 12 | 50 | 1 | 0 | 0 | 600 |
| 9 | 920 | 14 | 8 | 0 | 0 | 1 | 570 |
| 10 | 1 000 | 9 | 24 | 0 | 1 | 0 | 620 |

# Example multivariate linear regression model

$$\begin{aligned}\text{RENTAL PRICE} = \mathbf{w}[0] &+ \mathbf{w}[1] \times \text{SIZE} + \mathbf{w}[2] \times \text{FLOOR} \\ &+ \mathbf{w}[3] \times \text{BROADBAND RATE} \\ &+ \mathbf{w}[4] \times \text{ENERGY RATING A} \\ &+ \mathbf{w}[5] \times \text{ENERGY RATING B} \\ &+ \mathbf{w}[6] \times \text{ENERGY RATING C}\end{aligned}$$


# Regression: Evaluating Regression Models

University *of* Galway.ie

# Metrics for evaluating regression models

In this section we will introduce some of the most common performance measures used for regression tasks

- **Domain specific** measures of error:
  - Mean squared error (MSE)
  - Root mean squared error (RMSE)
  - Mean absolute error (MAE)
- **Domain independent**: $R^2$

The basic evaluation process is the same as for categorical targets / classification tasks

- Maintain separate training and test sets (e.g., using cross validation)
- Train the regression model on the training set
- Compute the performance measures of interest on both training and test sets

# Mean squared error (MSE)

$$\text{sum of squared errors} = \frac{1}{2}\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2$$

$$\text{mean squared error} = \frac{\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2}{n}$$

$\mathbb{M}(d_1) \ldots \mathbb{M}(d_n)$ is a set of n values predicted by the model, and $t_i \ldots t_n$ is a set of labels

The mean squared error (MSE) performance captures the average difference between the expected target values in the test set and the values predicted by the model

MSE allows us to rank the performance of multiple models on a regression problem

MSE values fall in the range [0, ∞], smaller values indicate better model performance

One criticism: MSE values are not especially meaningful – no sense of how much error occurs on individual predictions due to the squared term

# Root mean squared error (RMSE)

$$\text{root mean squared error} = \sqrt{\frac{\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2}{n}}$$

$\mathbb{M}(d_1) \ldots \mathbb{M}(d_n)$ is a set of n values predicted by the model, and $t_i \ldots t_n$ is a set of labels

RMSE values are in the same units as the target value and so allow us to say something more meaningful about what the error for predictions made by the model will be

RMSE values can be thought of as the 'average' error on each prediction made by a regression model

Due to the inclusion of the squared term, the root mean squared error tends to overestimate error slightly as it overemphasizes individual large errors

# Mean absolute error

$$\text{mean absolute error} = \frac{\sum_{i=1}^{n} abs(t_i - \mathbb{M}(\mathbf{d}_i))}{n}$$

An alternative measure that addresses the problem of large errors dominating the RMSE metric is the mean absolute error (MAE), which does not include a squared term

*abs* in the equation above refers to the absolute value, all other terms have the same meaning as before

As with RMSE, MAE values are in the same units as the target variable so we can say that MAE values give an indication of the 'average' error on each prediction

# Domain independent measures of error - $R^2$

**RMSE** and **MAE** give errors that are in the same units as the target variable – this is attractive as they give an intuitive measure of how a model is performing, e.g., 'a drug dosage prediction model is typically 1.38mg out in its dosage predictions'

One disadvantage – RMSE and MAE values are not sufficient to judge whether a model is making accurate predictions, without having deep knowledge of the domain (i.e., 'is an error of 1.38mg acceptable/unacceptable?')

The make such judgements without deep domain knowledge a normalised *domain independent* measure of error is helpful

**The $R^2$ coefficient** is a domain independent measure and compares the performance of a model on a test set with the performance of an imaginary model that always predicts the average values from the test set

$R^2$ values may be interpreted as as the amount of variation in the target feature that is explained by the descriptive features in the model

# Domain independent measures of error - $R^2$

$$\text{sum of squared errors} = \frac{1}{2}\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2$$

$$\text{total sum of squares} = \frac{1}{2}\sum_{i=1}^{n}(t_i - \bar{t})^2$$

$$R^2 = 1 - \frac{\text{sum of squared errors}}{\text{total sum of squares}}$$

$\bar{t}$ in the total sum of squares equation is the average value of the target variable

$R^2$ values are usually in the range [0,1]* – larger values indicate better performance

# *A note on the range of $R^2$ values

1.0 is always the maximum $R^2$ value

$R^2$ values can be <0 in certain (relatively rare) cases

Negative $R^2$ values indicate very poor model performance, i.e., the model performs worse than a horizontal straight-line hypothesis that always predicts the average value of the target feature

E.g., a negative $R^2$ value on the test set, along with a positive $R^2$ value on the training set, might indicate that the model is overfit to the training data

Suggestions if you encounter negative $R^2$ values:

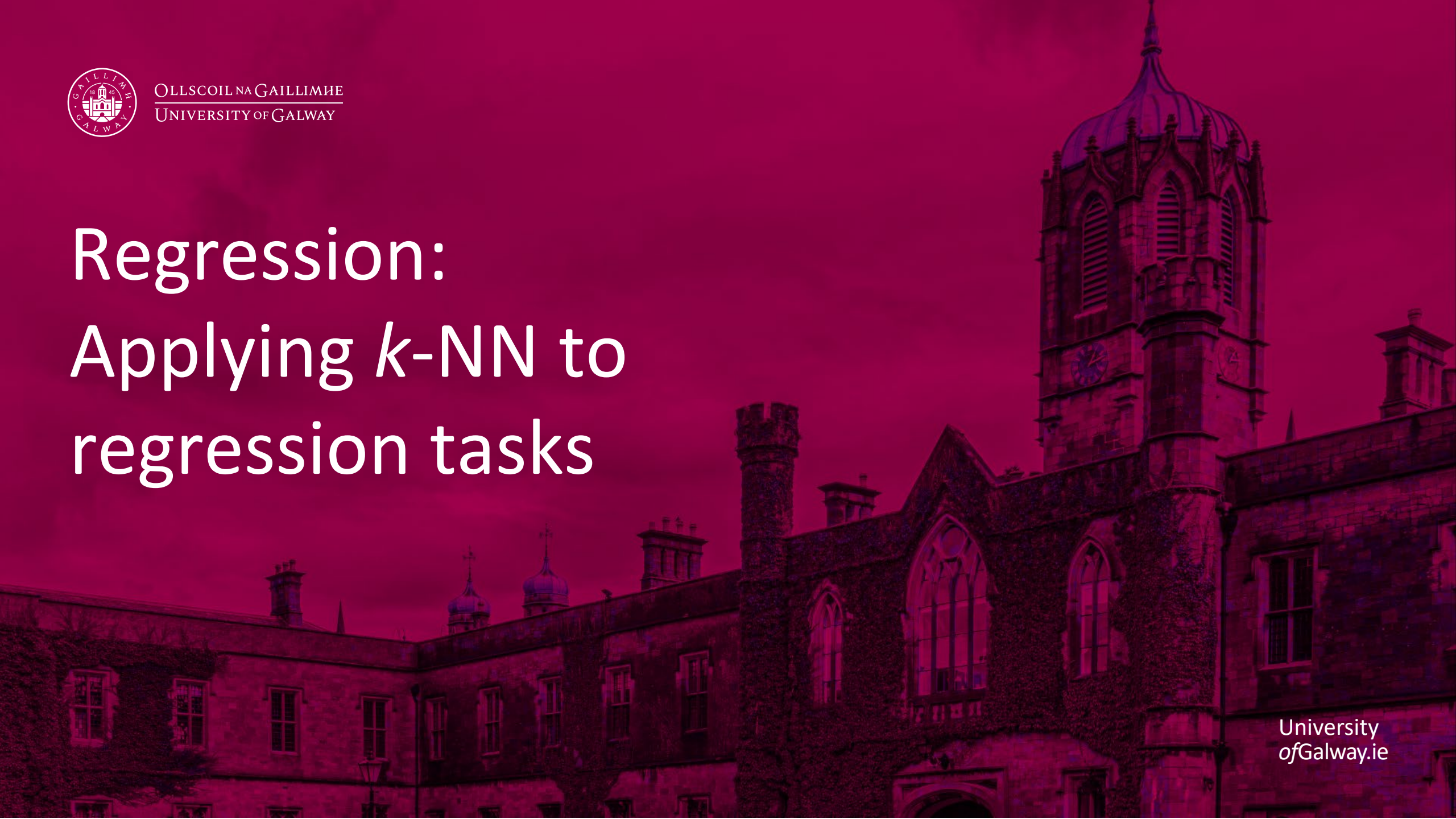
- Adjust hyperparameters
- Try a different type of ML model that uses a very different form of hypothesis (e.g., use a polynomial model instead of a linear model)

# Example error measures for a drug dosage prediction problem

Target units are in mg of drug dosage

| | | Linear Regression | | *k*-NN | |
|---|---|---|---|---|---|
| ID | Target | Prediction | Error | Prediction | Error |
| 1 | 10.502 | 10.730 | 0.228 | 12.240 | 1.738 |
| 2 | 18.990 | 17.578 | -1.412 | 21.000 | 2.010 |
| 3 | 20.000 | 21.760 | 1.760 | 16.973 | -3.027 |
| 4 | 6.883 | 7.001 | 0.118 | 7.543 | 0.660 |
| 5 | 5.351 | 5.244 | -0.107 | 8.383 | 3.032 |
| 6 | 11.120 | 10.842 | -0.278 | 10.228 | -0.892 |
| 7 | 11.420 | 10.913 | -0.507 | 12.921 | 1.500 |
| 8 | 4.836 | 7.401 | 2.565 | 7.588 | 2.752 |
| 9 | 8.177 | 8.227 | 0.050 | 9.277 | 1.100 |
| 10 | 19.009 | 16.667 | -2.341 | 21.000 | 1.991 |
| 11 | 13.282 | 14.424 | 1.142 | 15.496 | 2.214 |
| 12 | 8.689 | 9.874 | 1.185 | 5.724 | -2.965 |
| 13 | 18.050 | 19.503 | 1.453 | 16.449 | -1.601 |
| 14 | 5.388 | 7.020 | 1.632 | 6.640 | 1.252 |
| 15 | 10.646 | 10.358 | -0.288 | 5.840 | -4.805 |
| 16 | 19.612 | 16.219 | -3.393 | 18.965 | -0.646 |
| 17 | 10.576 | 10.680 | 0.104 | 8.941 | -1.634 |

| | | Linear Regression | | *k*-NN | |
|---|---|---|---|---|---|
| ID | Target | Prediction | Error | Prediction | Error |
| 17 | 10.576 | 10.680 | 0.104 | 8.941 | -1.634 |
| 18 | 12.934 | 14.337 | 1.403 | 12.484 | -0.451 |
| 19 | 10.492 | 10.366 | -0.126 | 13.021 | 2.529 |
| 20 | 13.439 | 14.035 | 0.596 | 10.920 | -2.519 |
| 21 | 9.849 | 9.821 | -0.029 | 9.920 | 0.071 |
| 22 | 18.045 | 16.639 | -1.406 | 18.526 | 0.482 |
| 23 | 6.413 | 7.225 | 0.813 | 7.719 | 1.307 |
| 24 | 9.522 | 9.565 | 0.043 | 8.934 | -0.588 |
| 25 | 12.083 | 13.048 | 0.965 | 11.241 | -0.842 |
| 26 | 10.104 | 10.085 | -0.020 | 10.010 | -0.095 |
| 27 | 8.924 | 9.048 | 0.124 | 8.157 | -0.767 |
| 28 | 10.636 | 10.876 | 0.239 | 13.409 | 2.773 |
| 29 | 5.457 | 4.080 | -1.376 | 9.684 | 4.228 |
| 30 | 3.538 | 7.090 | 3.551 | 5.553 | 2.014 |
| | **MSE** | | **1.905** | | **4.394** |
| | **RMSE** | | **1.380** | | **2.096** |
| | **MAE** | | **0.975** | | **1.750** |
| | $R^2$ | | **0.889** | | **0.776** |




# Regression: Applying *k*-NN to regression tasks

# *k*-NN - recap

*k*-Nearest Neighbour algorithm:

- Base prediction on several (*k*) nearest neighbours
- Compute distance from query case to all stored cases, and pick the nearest *k* neighbours

Classification with kNN:

- Neighbours vote on classification of test case

**Regression:**

- **Average the value of the neighbours**
- E.g., Scikit-learn class `sklearn.neighbors.KNeighborsRegressor`
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

Ollscoil na Gaillimhe
University of Galway

# kNN regression – uniform weighting

$$prediction(\boldsymbol{q}) = \frac{1}{k}\sum_{i=1}^{k} t_i$$

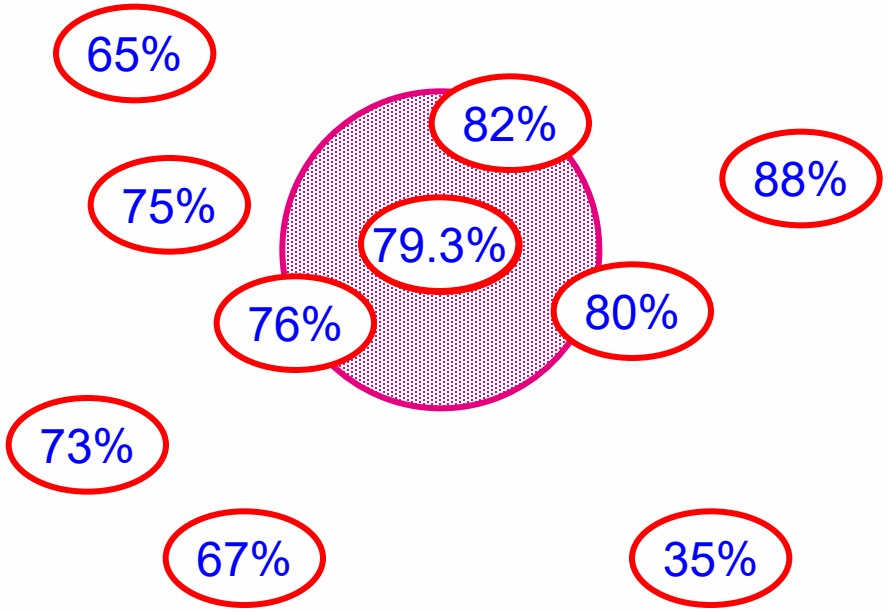
$\boldsymbol{q}$ is a vector containing the attribute values for the query instance
$k$ is the number of neighbours as before
$t_i$ is the target value for neighbour $i$
This assumes that each neighbour is given equal weighting

# kNN regression – distance weighting

$$prediction(\boldsymbol{q}) = \frac{\sum_{i=1}^{k}\left(\frac{1}{dist(\boldsymbol{q},\boldsymbol{d}_i)^2} \times t_i\right)}{\sum_{i=1}^{k}\left(\frac{1}{dist(\boldsymbol{q},\boldsymbol{d}_i)^2}\right)}$$

$\boldsymbol{q}$ is a vector containing the attribute values for the query instance
$dist(\boldsymbol{q},\boldsymbol{d}_i)$ returns the distance between the query and neighbour *i*
This assumes that each neighbour is given a weighting based on the inverse square of its distance from the query

# *k* Nearest Neighbours: Visualisation of Regression Example



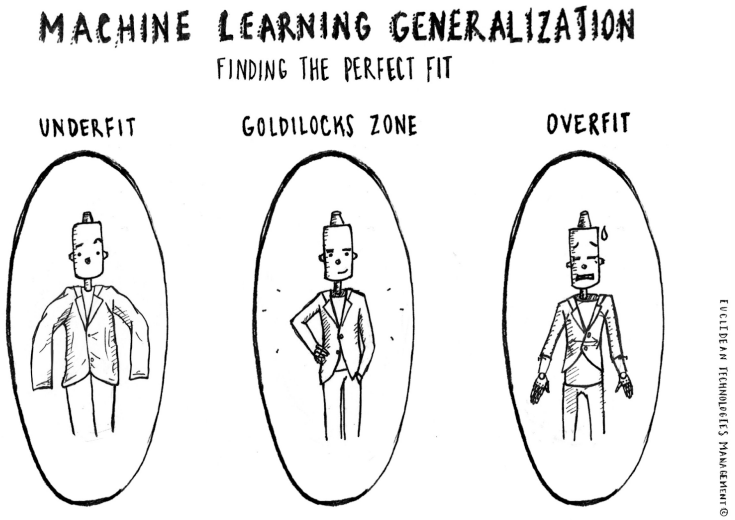

**Estimate: 79.3%**

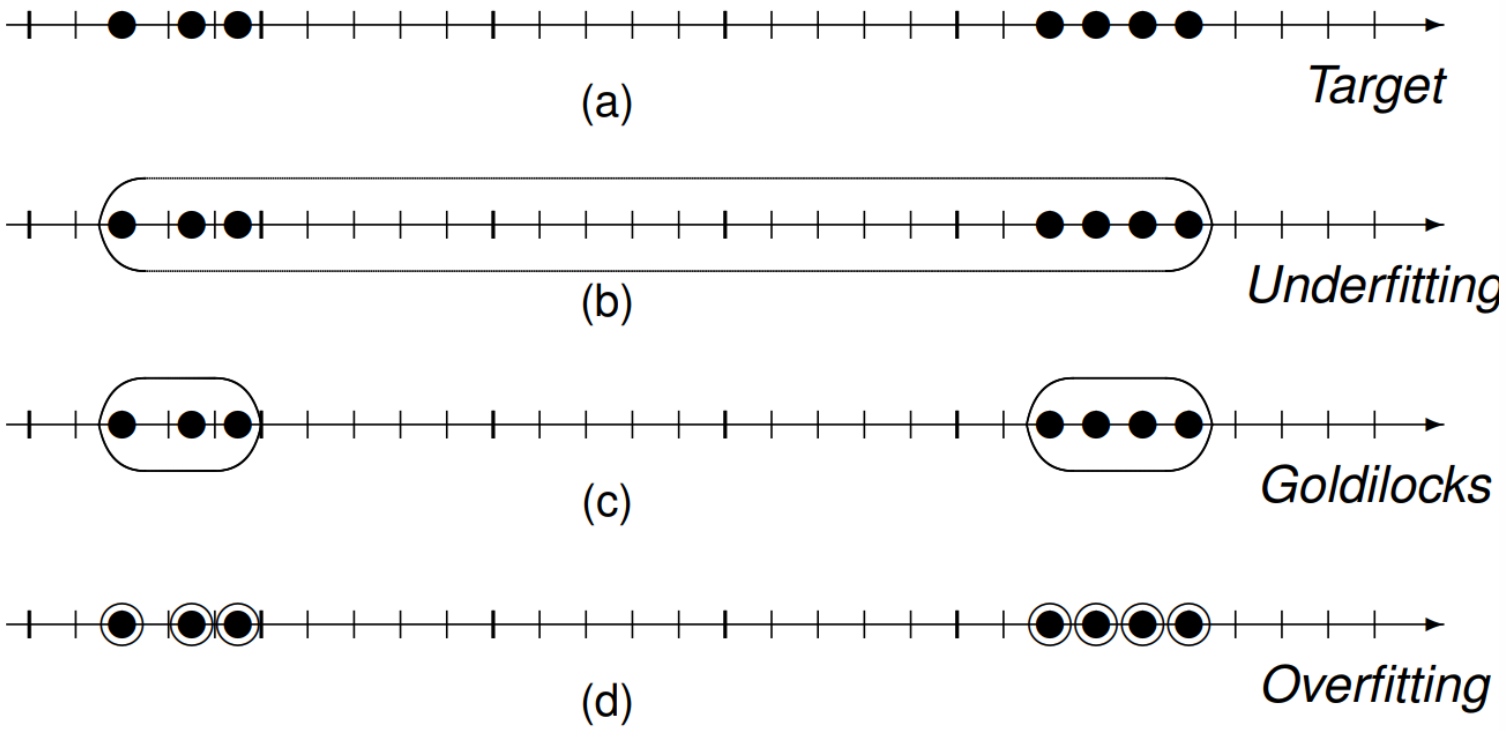
This visualisation assumes Euclidean distance and uniform weighting


# Regression: Applying Decision Trees to Regression

# Regression trees

Regression trees are constructed similarly to those for classification; the main change is that the function used to measure the quality of a split is changed so that it is a measure relevant to regression, e.g., variance, MSE, MAE etc. This adaption is easily made to the ID3/C4.5 algorithm




The aim in regression trees is to group similar target values together at a leaf node

Typically, a regression tree returns the mean target value at a leaf node

# Scikit-learn - DecisionTreeRegressor

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

See example regression line to the right produced in a simple variable regression problem by a `DecisionTreeRegressor`