Error Based Learning

Regression & SVMs

Sarah Jane Delany

Adapted from ML for PDA book

Big Idea

Error based approach to learning:

- A parameterised prediction model is initialised with a set of random parameters
- An error function is used to judge how well this initial model performs when making predictions on the training set
- Based on the value of the error function the parameters are iteratively adjusted to create a more and more accurate model

- e.g. similar to how humans learn a new sport

Simple Linear Regression

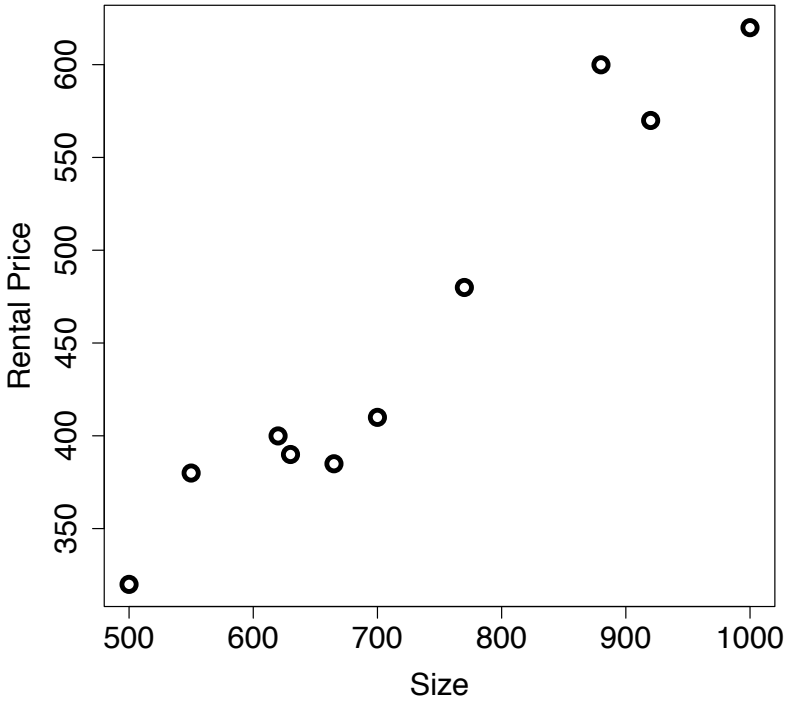
- Regression is a well-known and widely-applied method in statistics.
- Machine learning borrows many ideas from statistics, and regression is a core algorithm in predictive modelling.

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

Task:
Predict the office rental price based on the office descriptive features

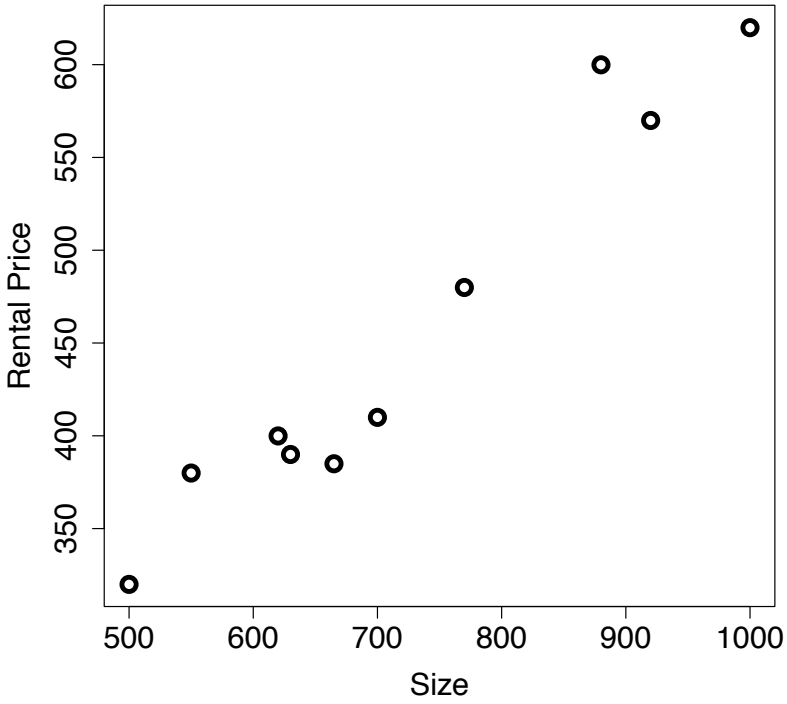
Simple Linear Regression

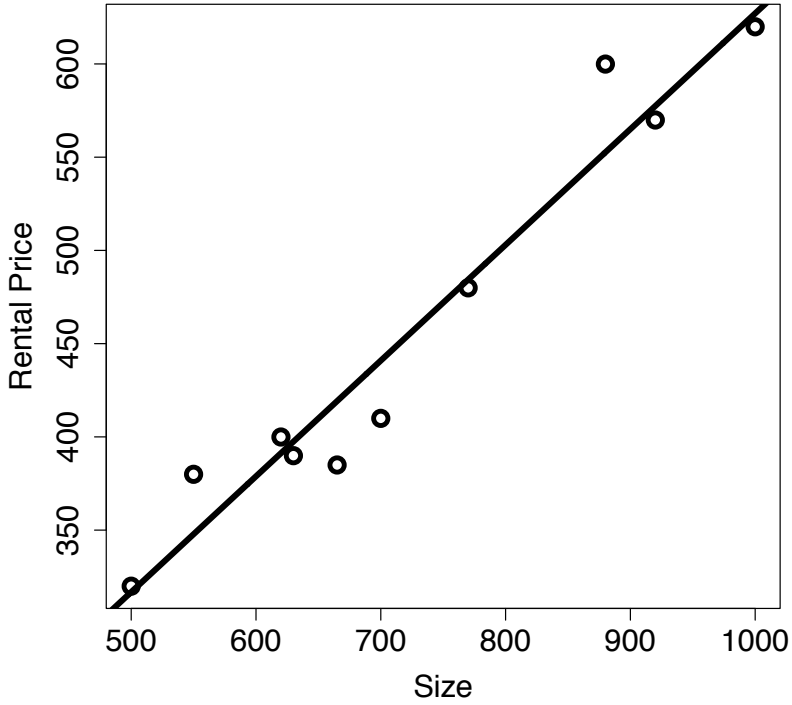
- Consider first just one descriptive feature, *size*



Simple Linear Regression

- Consider first just one descriptive feature, *size*





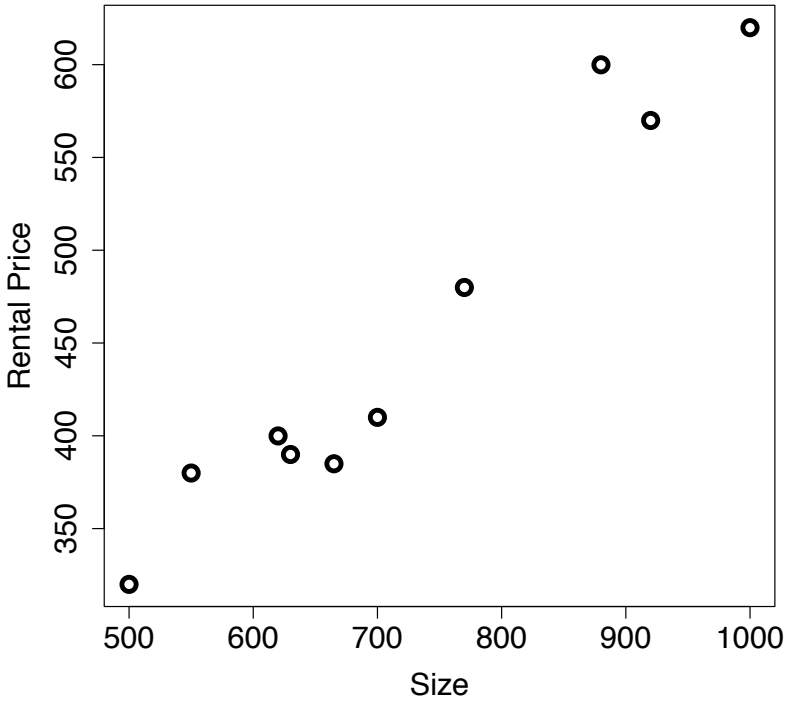
Equation of a line: $y = mx + b$

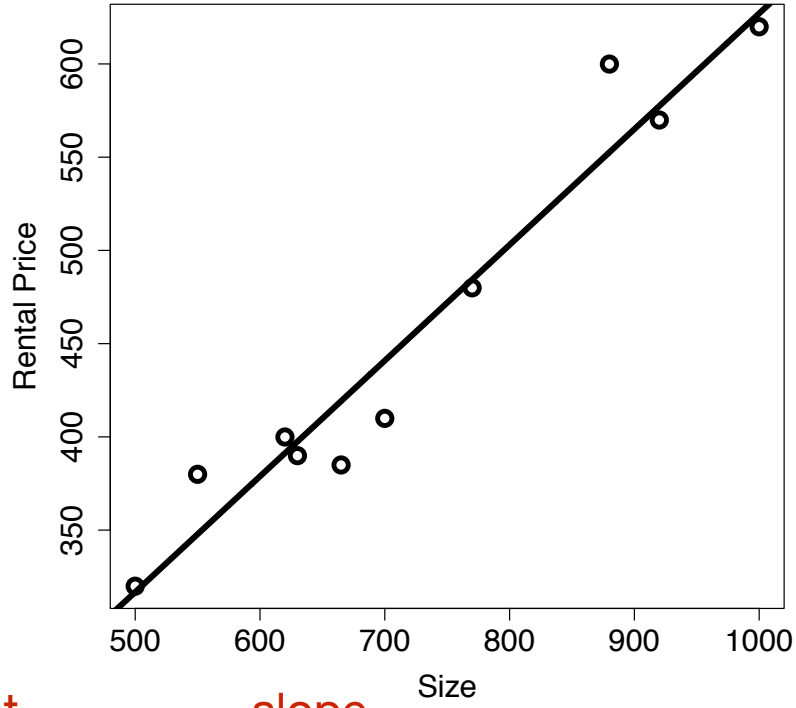
slope

intercept

Simple Linear Regression

- Consider first just one descriptive feature, *size*





intercept slope

$$\text{Rental Price} = 6.47 + 0.62 \times \text{Size}$$

What's the predicted price for office of 730 sq m?

$$6.47 + 0.62 \times 730 = 459.07$$

Simple Linear Regression

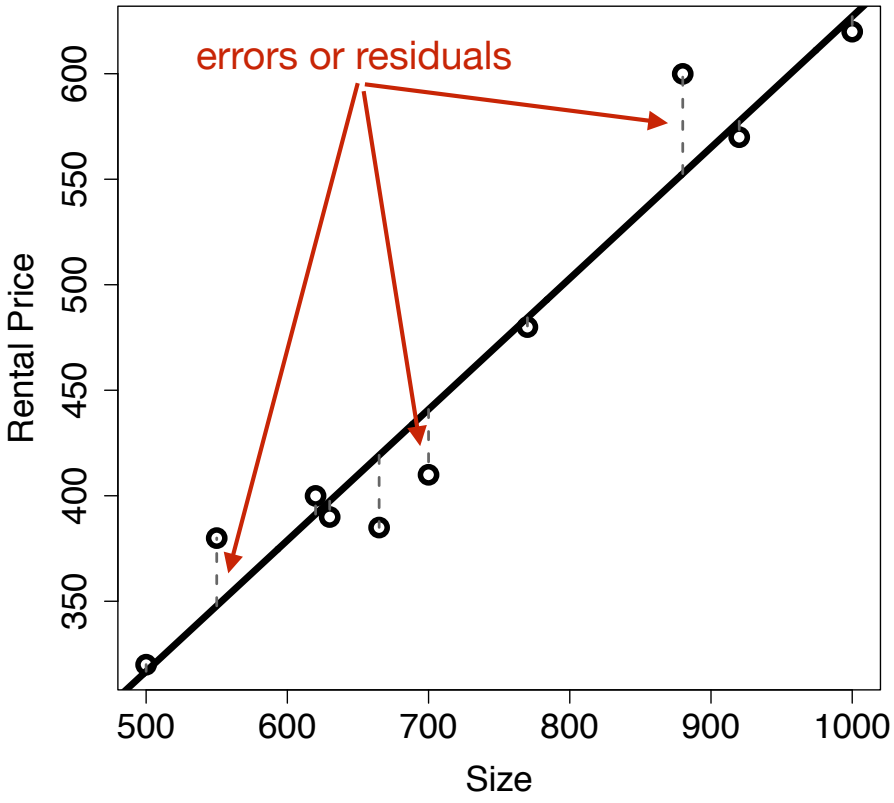
$$\mathbb{M}_{\mathbf{w}}(d) = \mathbf{w}[0] + \mathbf{w}[1] \times \mathbf{d}[1]$$

where w is a vector $<\mathrm{w}[0]\ ,\ \mathrm{w}[1]>$ and $\mathrm{w}[0]\ ,\ \mathrm{w}[1]$ are called weights

d is an example (instance) with one descriptive feature $\mathrm{d}[1]$

- Key to Linear Regression is determining the optimal weights for the model, i.e. that best fit the training data
- Use an error function to measure how well the model fits the training data

Measuring Error



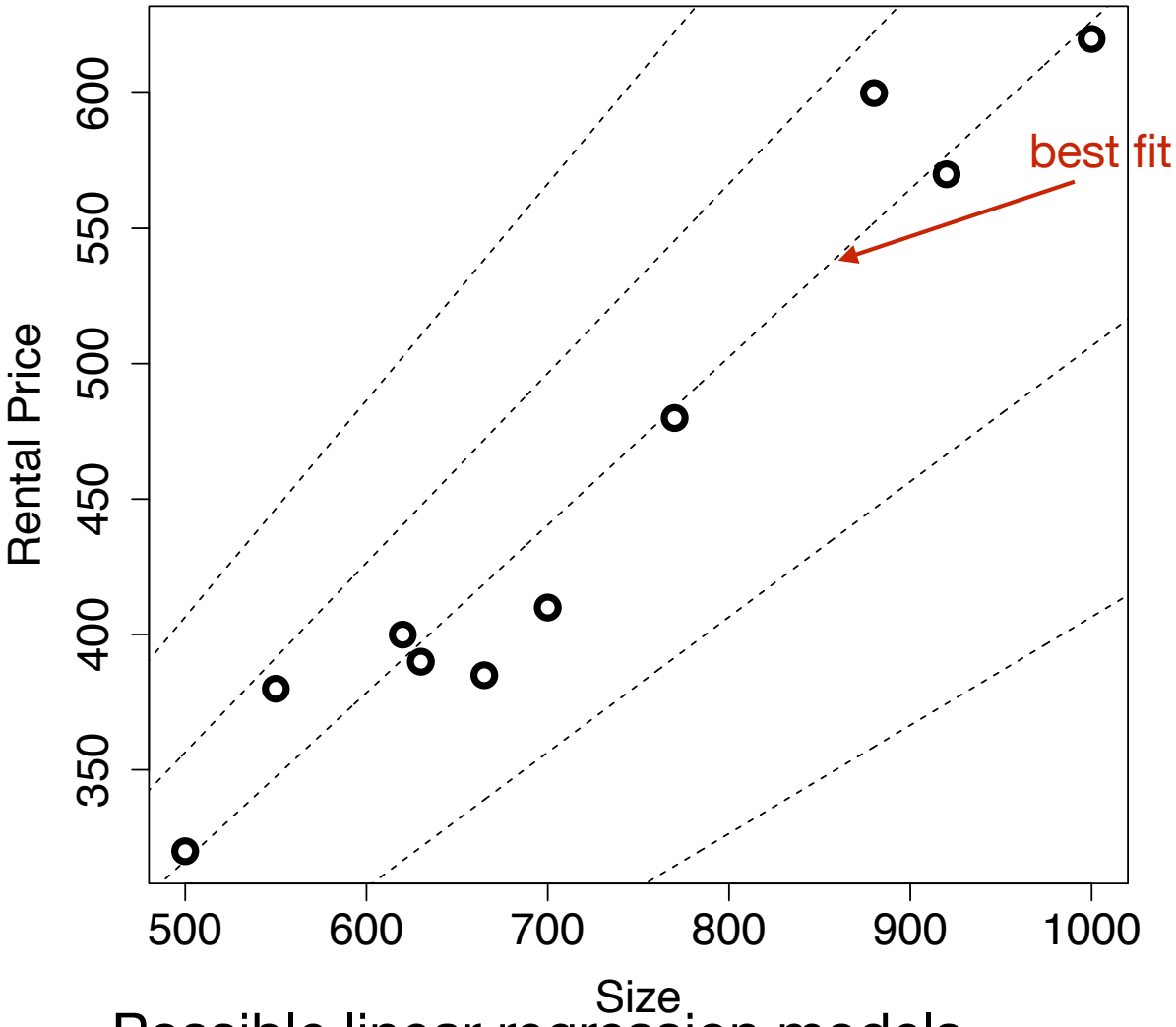
- The error function (aka loss function) captures the error between the predictions and the actual values in a training dataset
- Most commonly used is sum of squared errors error function, L_2

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}(t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i[1]))^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}(t_i - (\mathbf{w}[0] + \mathbf{w}[1] \times \mathbf{d}_i[1]))^2$$

D dataset with
n instances with **d**
descriptive features
and target feature *t*

Simple Linear Regression



Possible linear regression models with $w[0] = 6.47$ and $w[1] = 0.4, 0.5, \mathbf{0.62}, 0.7, 0.8$

Measuring Error

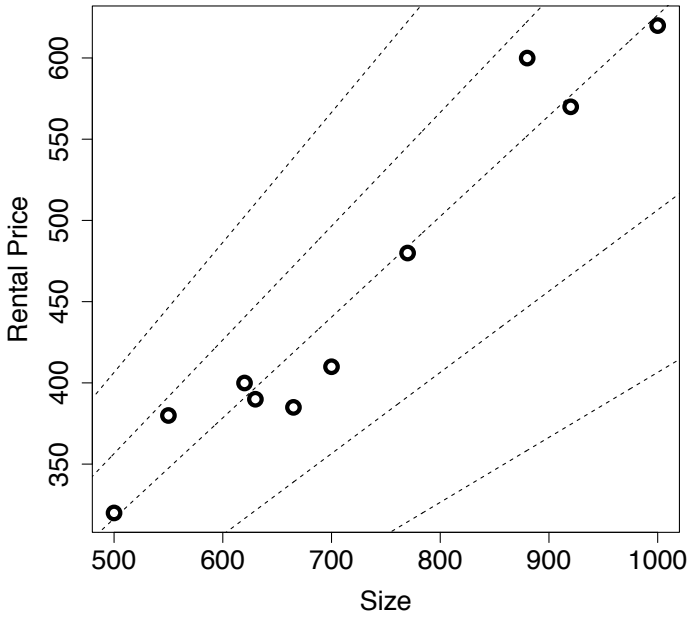
- Calculation of SSE loss function for example on the training data with w[0]=6.47 & w[1]=0.62:

ID	Rental Price	Model Prediction	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**

Measuring Error

- Calculation of loss function for our example on the training data:

ID	Rental Price	Model Prediction	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**



best fit

SSE:

w[1] = 0.4 is 136,218

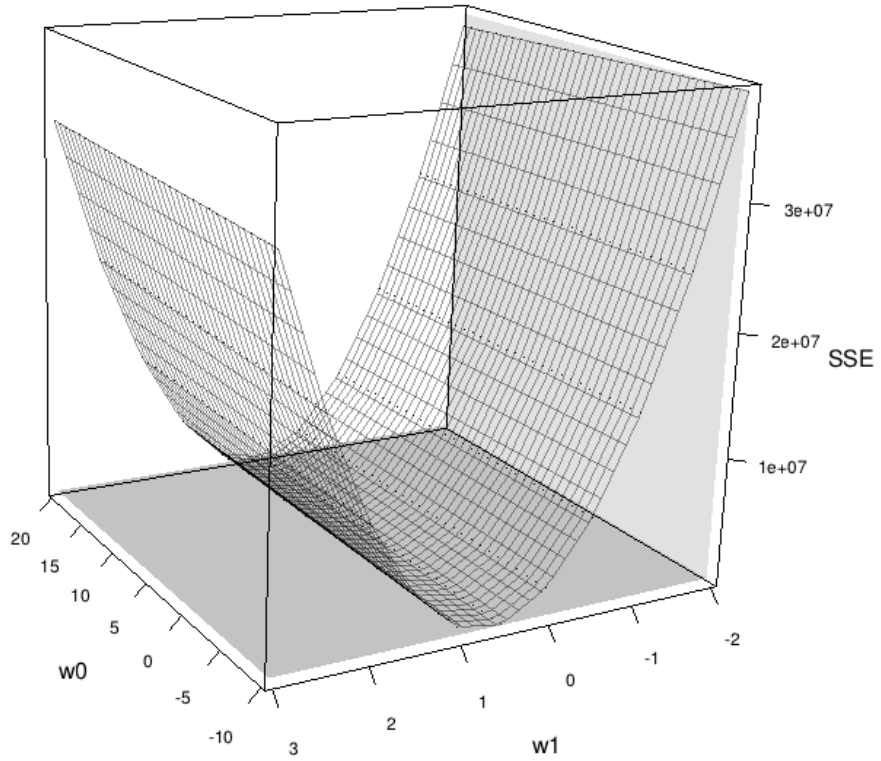
w[1] = 0.5 is 42,712

w[1] = 0.7 is 20,092

w[1] = 0.8 is 90,978

Error Surfaces

- For every possible combination of weights, w[0] and w[1], there is a corresponding sum of squared errors value that can be joined together to make a surface known as the error surface



3D surface plot generated by plotting SSE values for every combination of
$w[0]$ in range [-10,20] and
$w[1]$ in range [-2,3]

- Model that best fits the training data is the model corresponding to the lowest point on the error surface
- Can't use a brute force search on large datasets…

Error Surfaces

- Due to the linearity of the model, error surfaces:
  - are convex
  - have a global minimum (i.e. a unique set of weights with the lowest SSE)
- Least squares optimisation is finding the global minimum of the error surface will give us the set of weights defining the model that best fits the data
- The partial derivatives of a surface wrt a point measure the slope at that point. The lowest point in the surface will have zero slope
- So, the point on the error surface where the partial derivatives of the error surface wrt $\mathrm{w}[0]$ and $\mathrm{w}[1]$ are equal to 0 is the global minimum

Least Squares Optimisation

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}(t_i - (\mathbf{w}[0] + \mathbf{w}[1] \times \mathbf{d}_i[1]))^2$$

$$\frac{\partial}{\partial \mathbf{w}[0]}\frac{1}{2}\sum_{i=1}^{n}(t_i - (\mathbf{w}[0] + \mathbf{w}[1] \times \mathbf{d}_i[1]))^2 = 0$$

$$\frac{\partial}{\partial \mathbf{w}[1]}\frac{1}{2}\sum_{i=1}^{n}(t_i - (\mathbf{w}[0] + \mathbf{w}[1] \times \mathbf{d}_i[1]))^2 = 0$$

- Different ways to find this global minimum, we will look at a guided search approach known as gradient descent
- Gradient descent is one of the most important algorithms in machine learning

Multivariate Linear Regression

- Extend our simple linear regression example to m, more than one, descriptive features:

$$\begin{aligned}\mathbb{M}_{\mathbf{w}}(\mathbf{d}) &= \mathbf{w}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \cdots + \mathbf{w}[m] \times \mathbf{d}[m] \\ &= \mathbf{w}[0] + \sum_{j=1}^{m} \mathbf{w}[j] \times \mathbf{d}[j]\end{aligned}$$

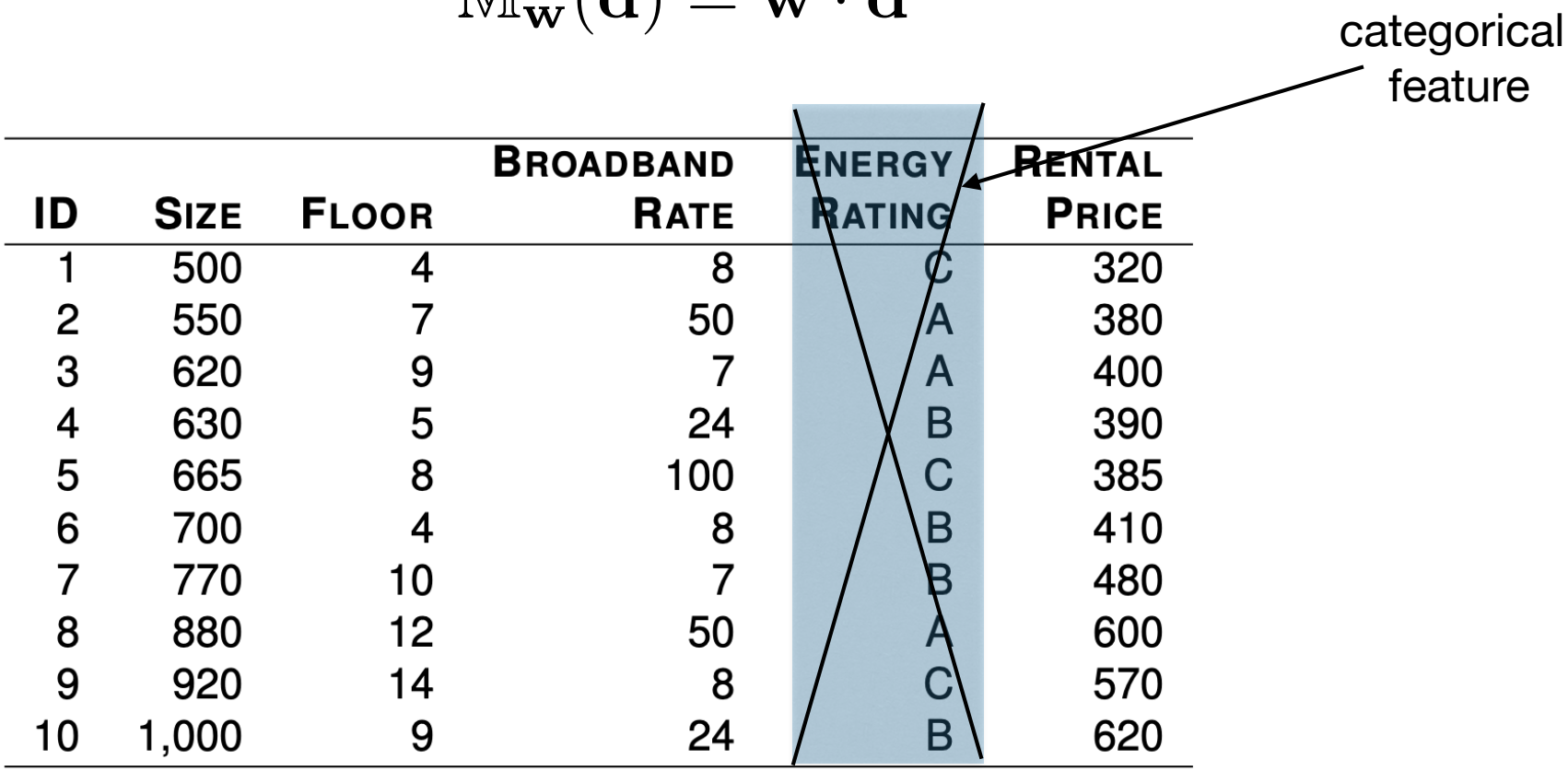
- Add in a dummy descriptive feature $\mathrm{d}[0] = 1$ *(to make it neater)*:

$$\begin{aligned}\mathbb{M}_{\mathbf{w}}(\mathbf{d}) &= \mathbf{w}[0] \times \mathbf{d}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \ldots + \mathbf{w}[m] \times \mathbf{d}[m] \\ &= \sum_{j=0}^{m} \mathbf{w}[j] \times \mathbf{d}[j] \\ &= \mathbf{w} \cdot \mathbf{d}\end{aligned}$$

dot product of two vectors is the sum of the product of their corresponding elements

Example:

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \mathbf{w} \cdot \mathbf{d}$$



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

$$\begin{aligned}\text{Rental Price} = \mathbf{w}[0] + \mathbf{w}[1] \times \text{Size} + \mathbf{w}[2] \times \text{Floor} \\ + \mathbf{w}[3] \times \text{Broadband Rate}\end{aligned}$$

Multivariate Linear Regression

- Sum of squared errors loss function changes slightly to become:

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}(t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i))^2$$

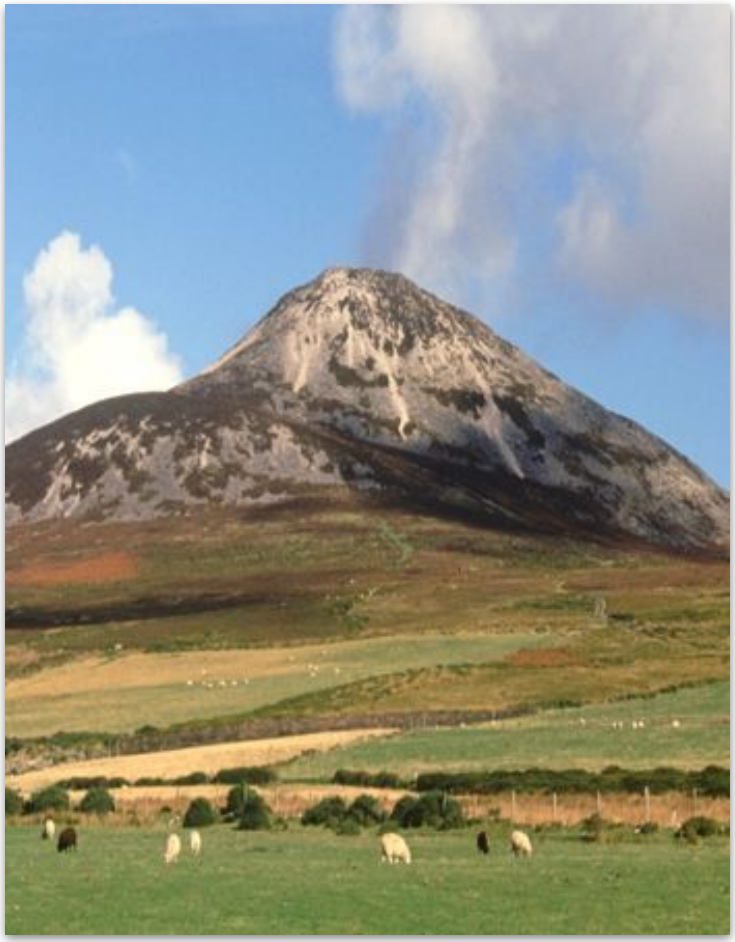
$$= \frac{1}{2}\sum_{i=1}^{n}(t_i - (\mathbf{w} \cdot \mathbf{d}_i))^2$$

- Find best-fit weights by gradient descent, a guided search from a random starting point

Gradient Descent

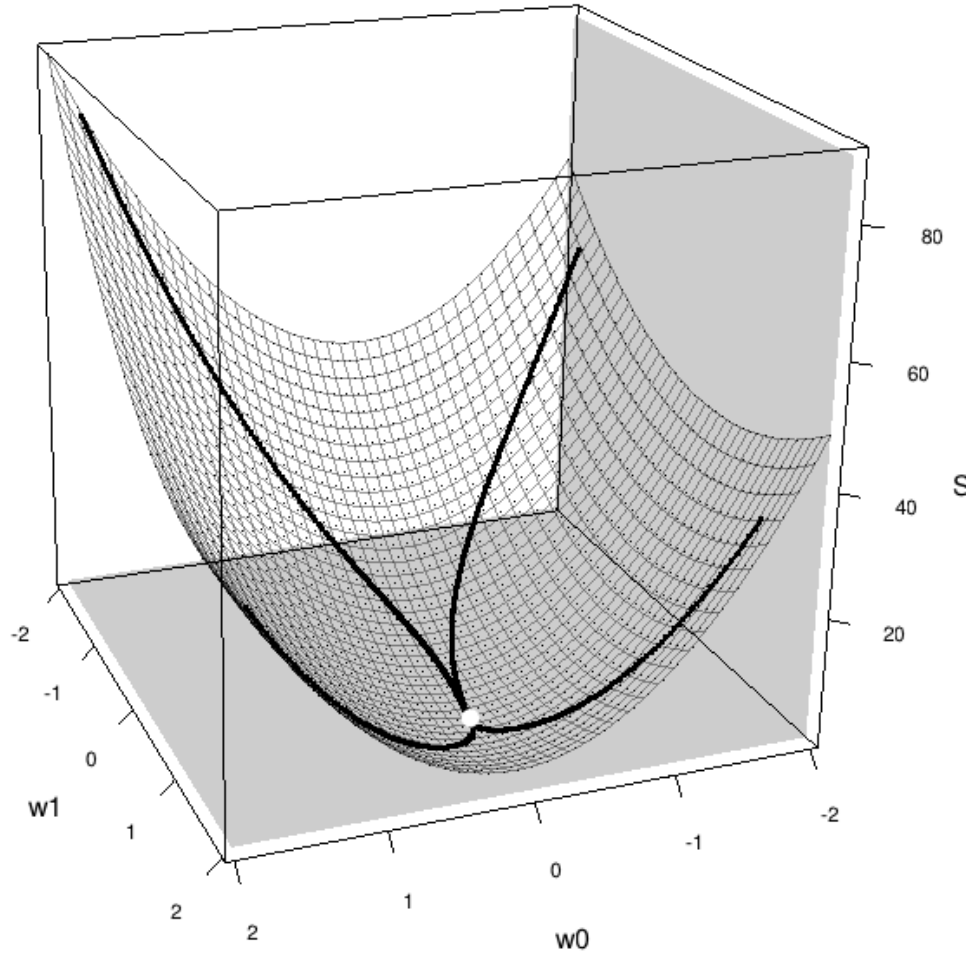
- Shortest route to bottom (typically) is straight down - steepest gradient

- Iterative process, moving in small steps, selecting the step that will move downwards each time

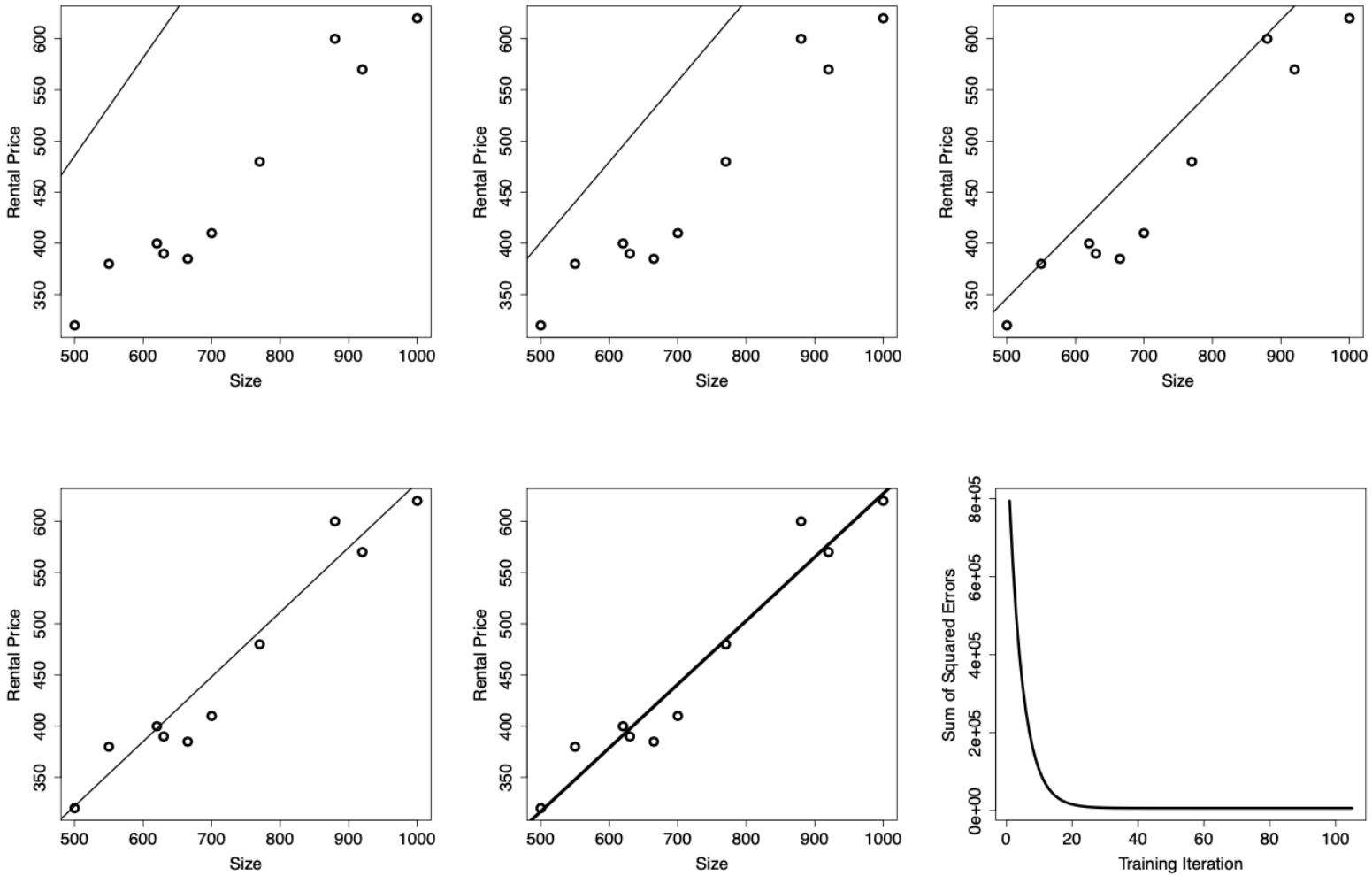


Gradient Descent Algorithm

- Select a random point in the weight space (i.e. each weight is assigned a random value within some sensible range)
- Calculate the SSE for training data, which defines a point on the error surface
- Determine the slope of the error function, by evaluating the derivative at this point on the error surface
- Adjust the weights using the direction of the error surface gradient and move to a new position on the error surface
- Repeat until the global minimum is reached



Illustrative Example



- A selection of models developed during the gradient descent process for simple, one descriptive variable, problem

Adjusting the weights

- Each weight is considered independently and a small value, called a delta value, is added
- Delta value must ensure that the change leads to a movement *downwards* on the error surface
- The *direction* and *magnitude* of the weight adjustment is determined by the gradient (slope) of the error surface at the current position
- The error surface is defined by the error or loss function, L_2
- The gradient of the error surface is given by the value of the partial derivative of L_2 wrt a particular weight at that point

Calculating the gradient

- Consider the simple case of one training example (d,t)

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}[j]} L_2\left(\mathbb{M}_{\mathbf{w}}, \mathcal{D}\right) &= \frac{\partial}{\partial \mathbf{w}[j]}\left(\frac{1}{2}\left(t - \mathbb{M}_{\mathbf{w}}\left(\mathbf{d}\right)\right)^2\right) \\ &= (t - \mathbb{M}_{\mathbf{w}}(\mathbf{d})) \times \frac{\partial}{\partial \mathbf{w}[j]}(t - \mathbb{M}_{\mathbf{w}}(\mathbf{d})) \\ &= (t - \mathbb{M}_{\mathbf{w}}(\mathbf{d})) \times \frac{\partial}{\partial \mathbf{w}[j]}(t - (\mathbf{w} \cdot \mathbf{d})) \\ &= (t - \mathbb{M}_{\mathbf{w}}(\mathbf{d})) \times -\mathbf{d}[j]\end{aligned}$$

- For a set of n training instances

$$\frac{\partial}{\partial \mathbf{w}[j]} L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \sum_{i=1}^{n}\left(\left(t_i - \mathbb{M}_{\mathbf{w}}\left(\mathbf{d}_i\right)\right) \times -\mathbf{d}_i[j]\right)$$

- The direction of the gradient calculated in this way is towards the highest values on the error surface whereas we need to move towards lower values on the error surface

Adjusting the weights

- The small step downwards, i.e. the delta value, is:

$$-\frac{\partial}{\partial \mathbf{w}[j]} L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \sum_{i=1}^{n} ((t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)) \times \mathbf{d}_i[j])$$

- Each weight is adjusted by

$$\mathbf{w}[j] \leftarrow \mathbf{w}[j] + \alpha \underbrace{\sum_{i=1}^{n} ((t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)) \times \mathbf{d}_i[j])}_{\text{errorDelta(} D\text{, w[j])}}$$

learning rate

Adjusting the weights

instance error

feature value

$$\mathbf{w}[j] \leftarrow \mathbf{w}[j] + \alpha \underbrace{\sum_{i=1}^{n} ((t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)) \times \mathbf{d}_i[j])}$$

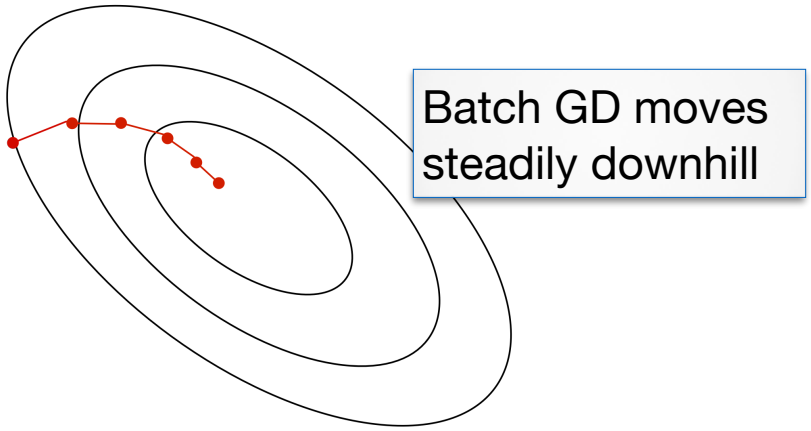
- If the errors show that the predictions of the model are too **high**, then $\mathbf{w}[j]$ should be decreased if $\mathbf{d}_i[j]$ is positive and should be increased if $\mathbf{d}_i[j]$ is negative

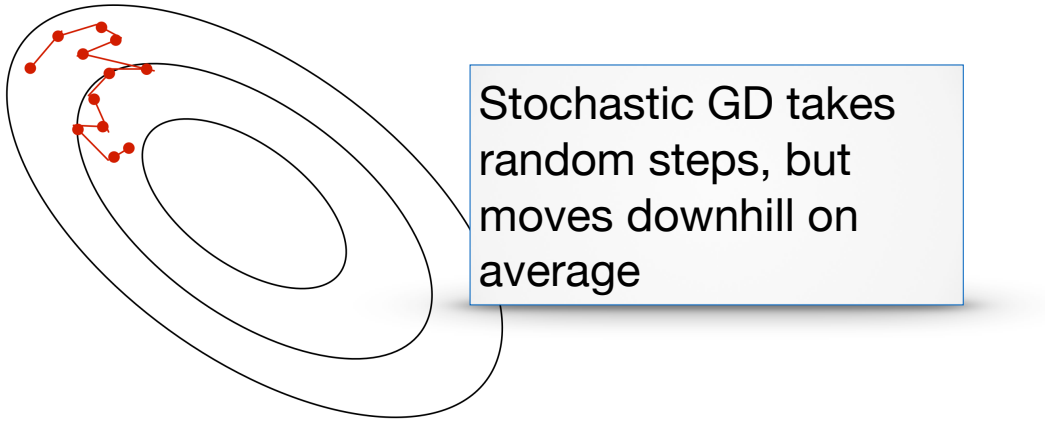
- If the errors show that the predictions of the model are too **low**, then $\mathbf{w}[j]$ should be increased if $\mathbf{d}_i[j]$ is positive and should be decreased if $\mathbf{d}_i[j]$ is negative

- This is known as Batch Gradient Descent as each weight is updated once after summing error over all instances in the dataset

Stochastic Gradient Descent

- In Stochastic GD we choose a single training example (perhaps at random), compute the loss and do the update directly
- Works well even with small number of training examples
- One at a time is slow, compromise is mini-batch gradient descent using a random batch of training examples
- Gives smoother convergence





Gradient Descent Algorithm

Require: set of training instances $\mathcal{D}$
Require: a learning rate α that controls how quickly the algorithm converges
Require: a function, **errorDelta**, that determines the direction in which to adjust a given weight, **w**[*j*], so as to move down the slope of an error surface determined by the dataset, $\mathcal{D}$
Require: a convergence criterion that indicates that the algorithm has completed

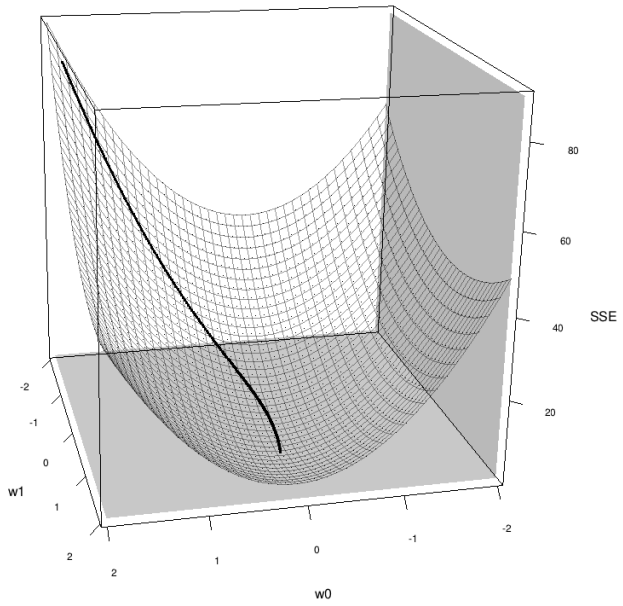
1: **w** $\leftarrow$ random starting point in the weight space
2: **repeat**
3: **for** each **w**[*j*] in **w do**
4: **w**[*j*] $\leftarrow$ **w**[*j*] $+ \alpha \times$ **errorDelta**($\mathcal{D}$, **w**[*j*])
5: **end for**
6: **until** convergence occurs

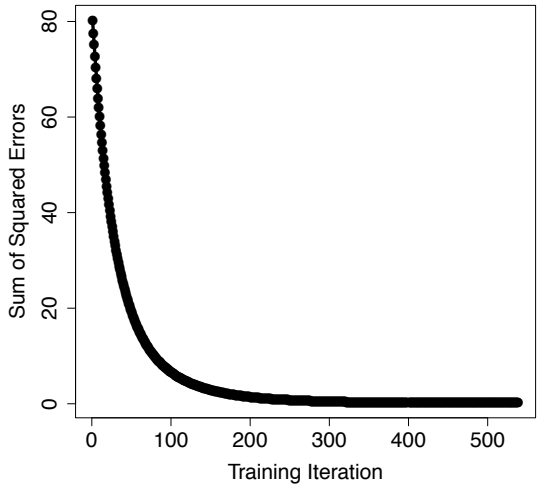
Choosing Learning Rate & Initial Weights

- The learning rate α determines the size of the adjustment made to each weight at each step of the process
- Choosing learning rates is not a well defined science
- Most people use rules of thumb and trial and error…
- A typical range is [0.00001, 10]

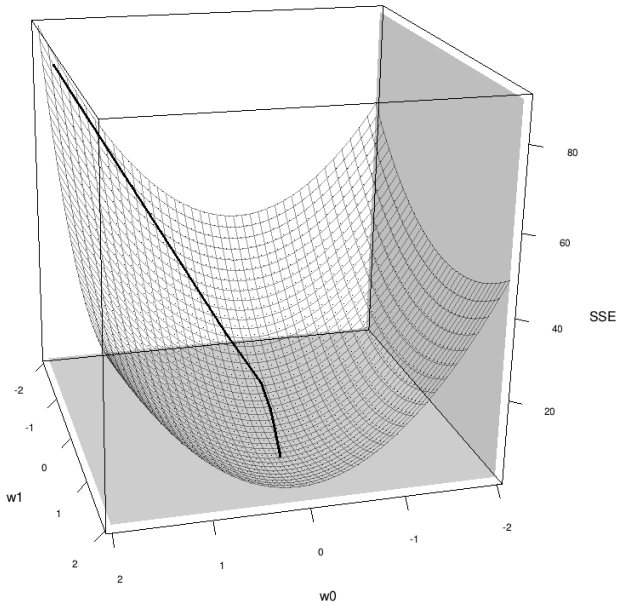
- Initial weights are chosen randomly from a predefined range that is input to the algorithm
- There is no well established proven methods for choosing this range but it is much easier to deal with normalised data
- Based on empirical evidence using range [-0.2, 0.2] tends to work well

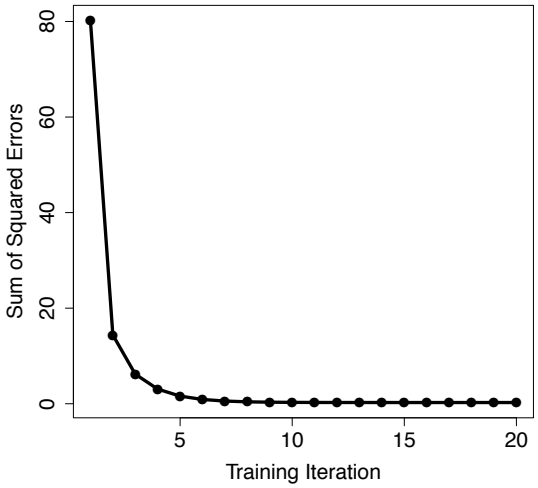
Learning Rate



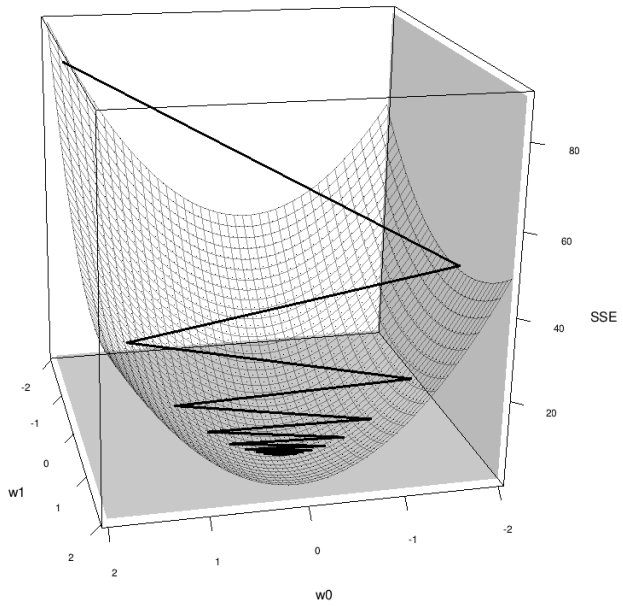


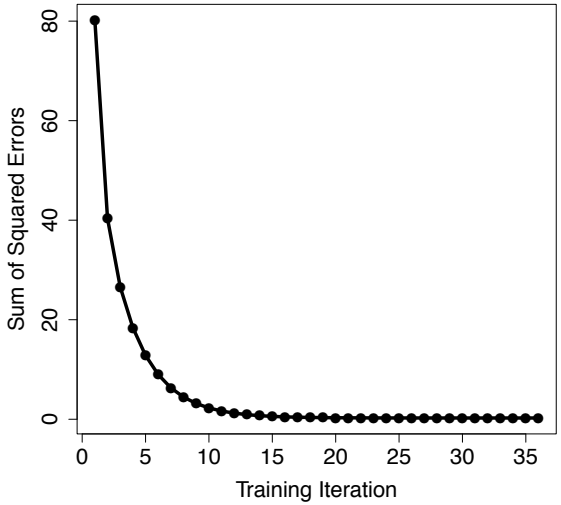
$\alpha = 0.002$





$\alpha = 0.08$





$\alpha = 0.18$

Interpreting Multivariable LR Models

- Useful feature of regression models is that weights indicate the effect of each descriptive feature on the predictions

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

Descriptive Feature	Weight
Size	0.6270
Floor	-0.1781
Broadband Rate	0.071396

- *Sign* indicates a positive or negative impact on the target
- *Magnitude* shows how much the value of the target changes for each unit change in the descriptive feature

Interpreting Multivariable LR Models

- Performing a *t*-test can determine the importance of each descriptive feature in the model

t-statistic = weight/stdError

Descriptive Feature	Weight	Standard Error	*t*-statistic	*p*-value
SIZE	0.6270	0.0545	11.504	<0.0001
FLOOR	-0.1781	2.7042	-0.066	0.949
BROADBAND RATE	0.071396	0.2969	0.240	0.816

- The null hypothesis is that the feature does not have a significant impact on the model
- Reject the null hypothesis if the p-value is less than the required significance level, normally .05 (5%) or .01 (1%)
- Significance level controls the Type I error rate: how serious is it to believe that there is a difference when in fact there isn't.

Handling Categorical Features

- Change categorical features to numeric features using One Hot Encoding.
- Disadvantage is to increase the number of features and therefore weights
- Can reduce features by one using the Drop First approach

Handling Categorical Features

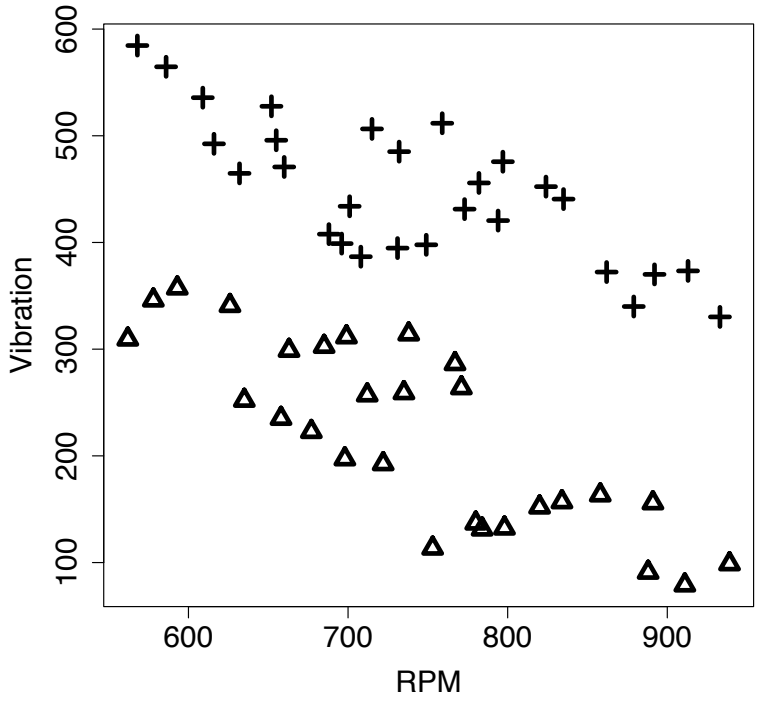
- Change categorical features to numeric features using One Hot Encoding.
- Disadvantage is to increase the number of features and therefore weights
- Can reduce features by one using the Drop First approach

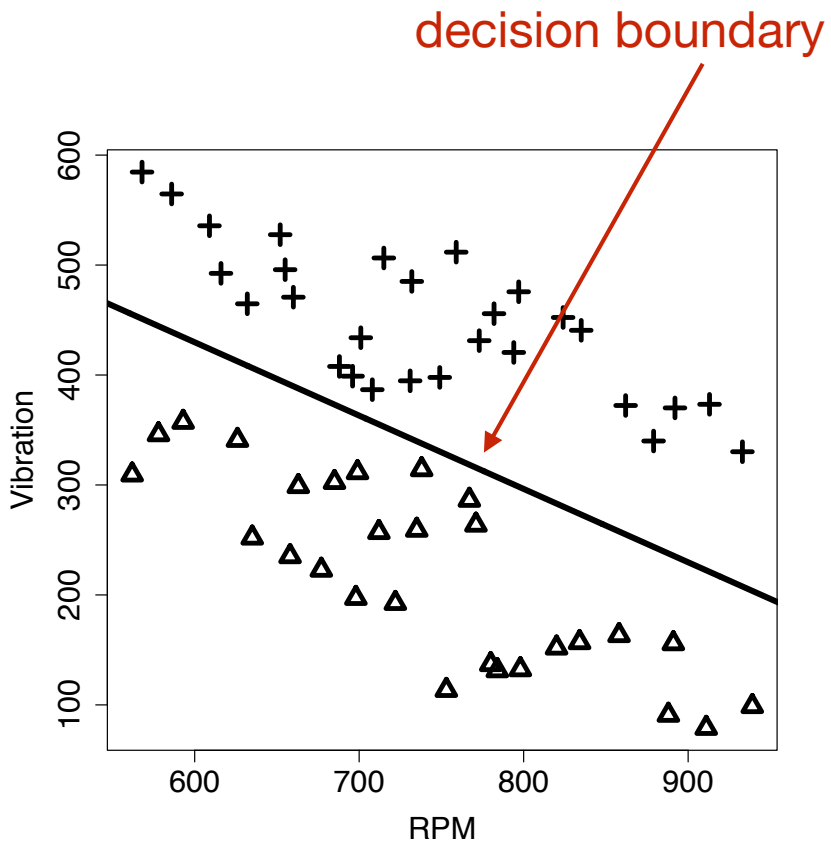
ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

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

Handling Categorical Target Features

- We would like to predict an output which takes the values {0,1},
- E.g. Input variables are measurements of RPM and Vibration and output is whether the generator is faulty or not
  ⇒ {0: good, 1: faulty}

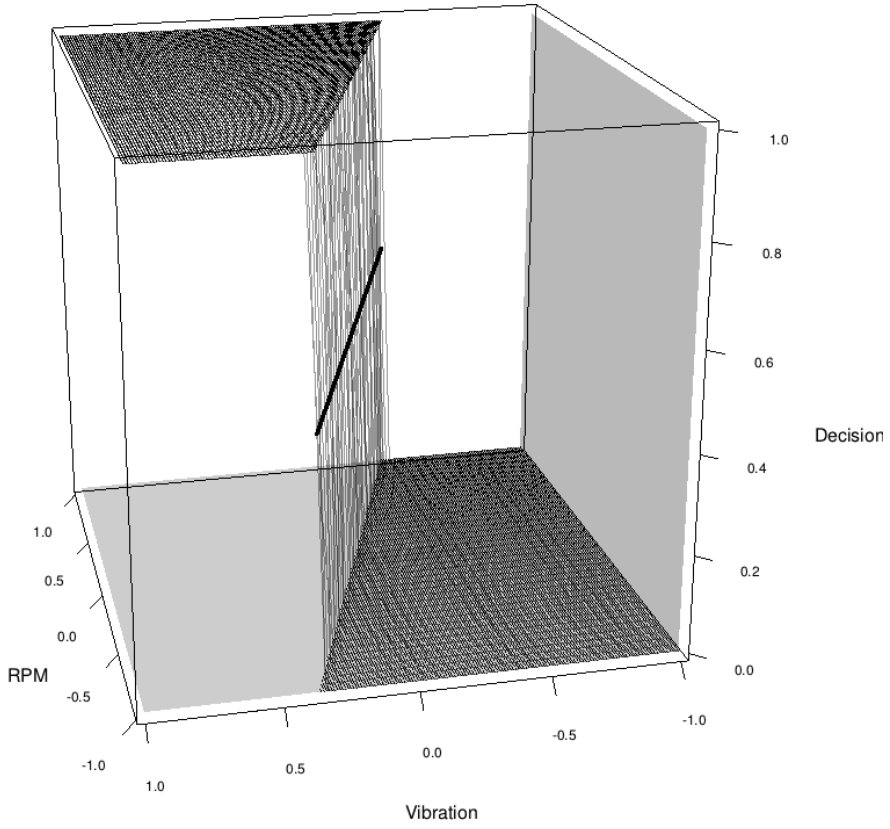




Handling Categorical Target Features

- The data is linearly separable and the decision boundary in our example is a line
- Any instance on the decision boundary will satisfy the decision boundary equation but anything above the decision boundary will be positive and anything below it will be negative

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & otherwise \end{cases}$$

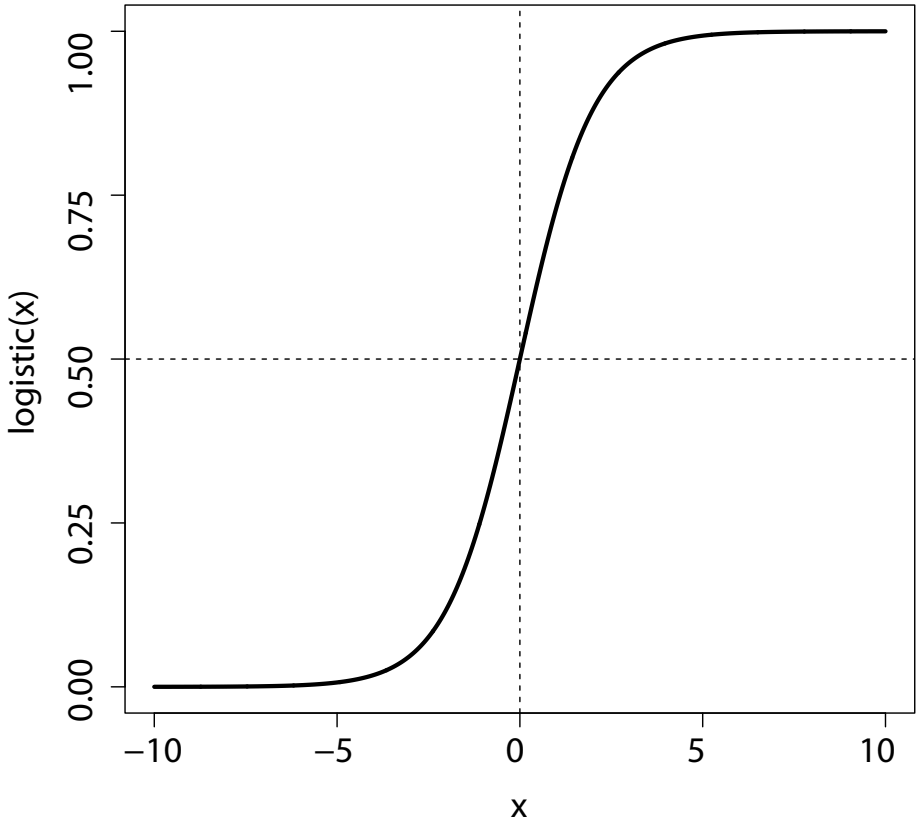


Problems with this decision surface:

- not differentiable so cannot calculate gradient
- hard threshold - outputs of 1 or 0 only

Logistic Function

- Logistic Function solves these problems
  - Gentle transition from 0 to 1 predictions
  - It is differentiable ⇒ we can use gradient descent



$$Logistic(x) = \frac{1}{1 + e^{-x}}$$

Logistic Regression

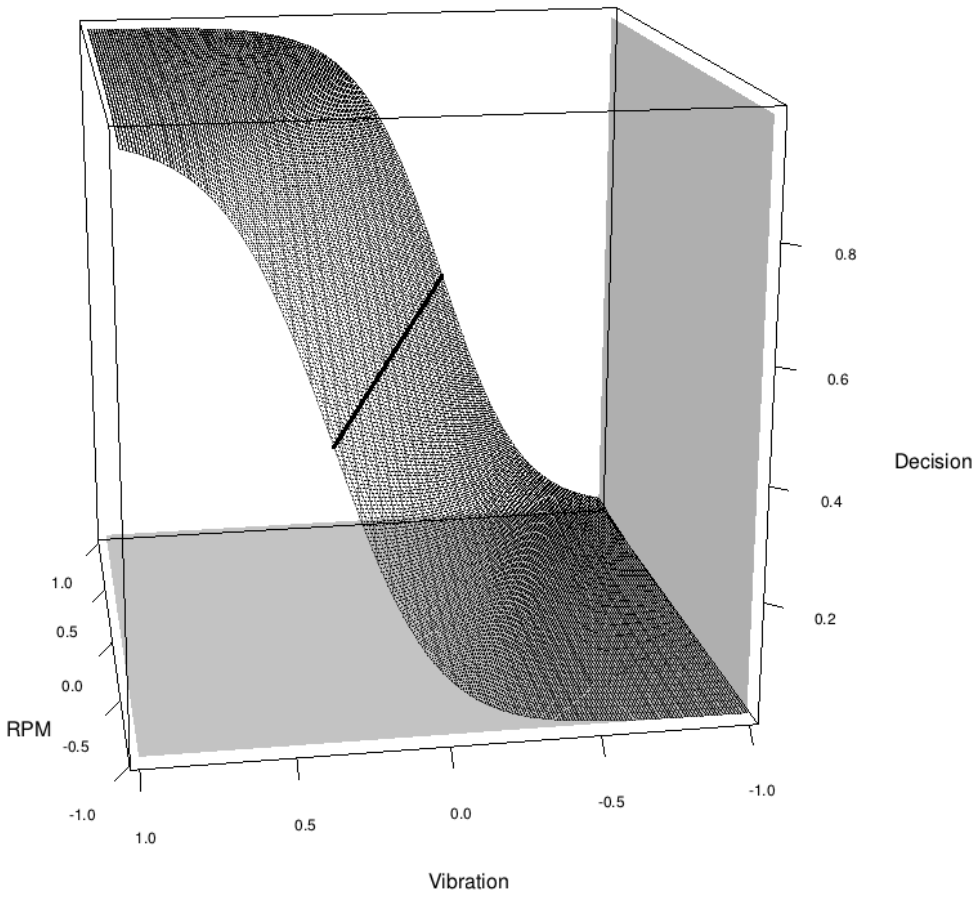
- For a logistic regression model the output of the basic linear regression model is passed through a logistic function

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = Logistic(\mathbf{w} \cdot \mathbf{d})$$
$$= \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{d}}}$$

- Logistic Regression model outputs can be interpreted as probabilities

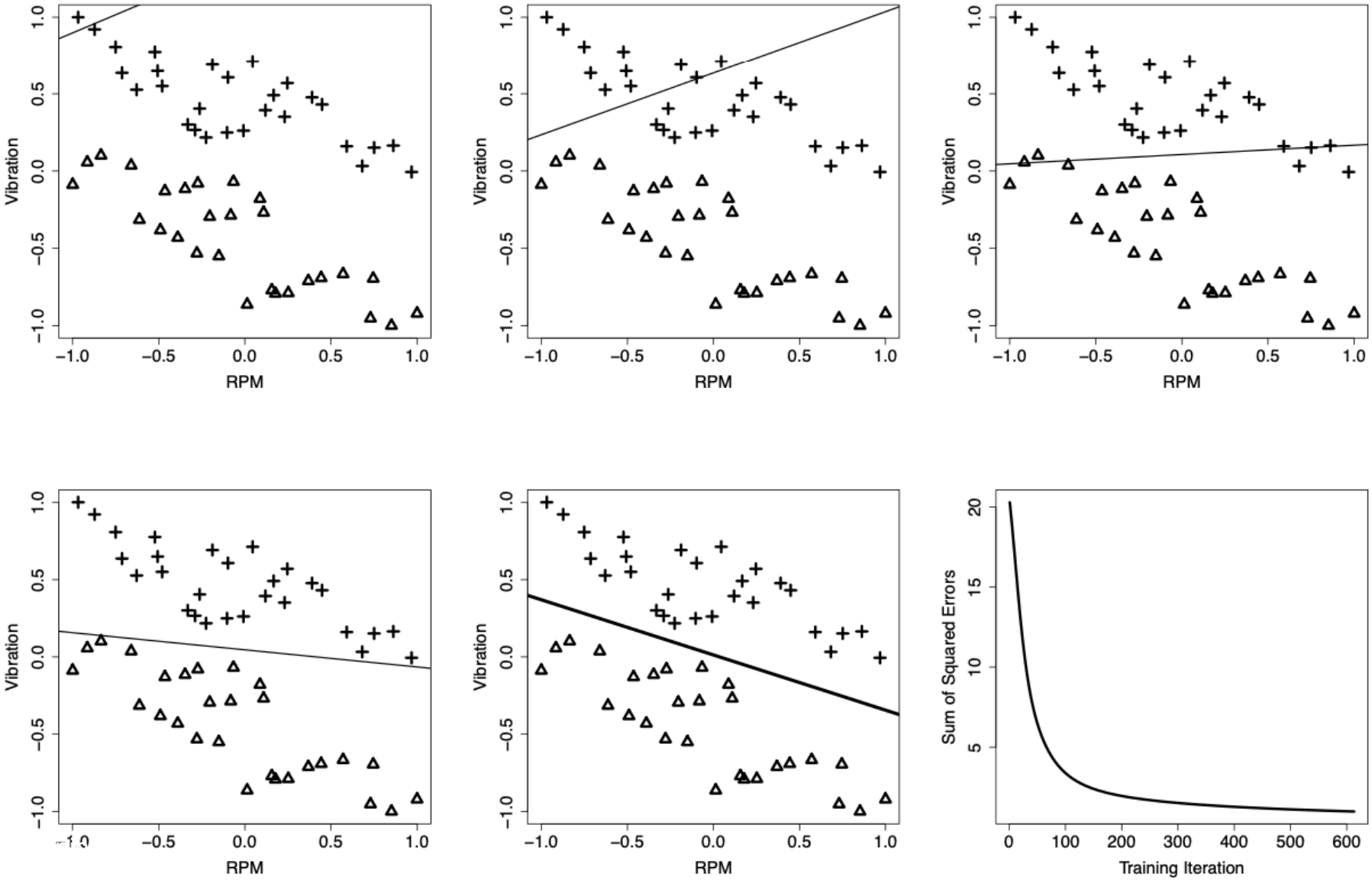
$$P(t = \text{faulty}|\mathbf{d}) = \mathbb{M}_{\mathbf{w}}(\mathbf{d})$$

$$P(t = \text{good}|\mathbf{d}) = 1 - \mathbb{M}_{\mathbf{w}}(\mathbf{d})$$



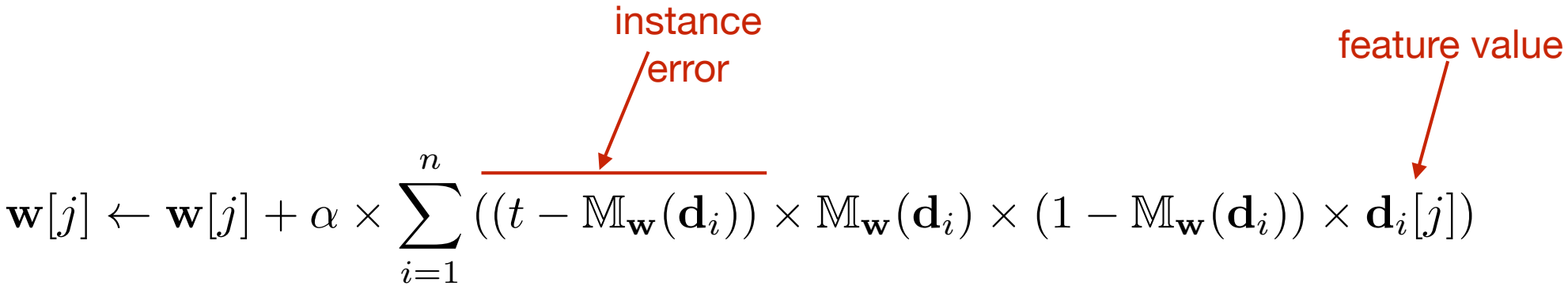
Logistic Regression

- A selection of logistic regression models during the training process, using gradient descent, of our example problem



Logistic Regression

- The only change to the gradient descent algorithm is the update rule for the weights



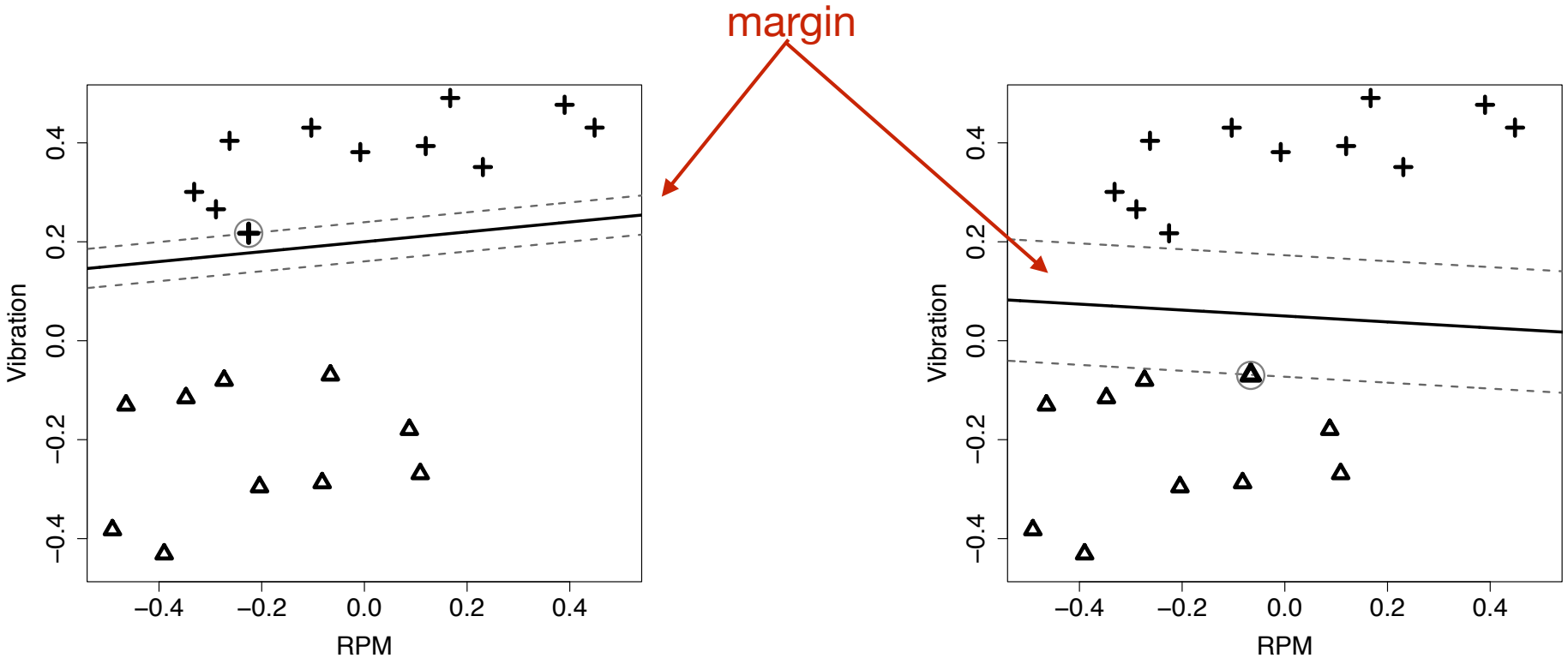
where $\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{d}}}$

Logistic Regression

- In its basic form can only handle binary target features.
- To handle categorical target features with more than two classes (multinomial prediction or multi-class classification) use a one-versus-all ensemble approach
  - Results in an explosion in the number of weights required so other approaches are often favoured

Support Vector Machines (SVM)

- Another approach to predictive modeling based on error based learning
- Training an SVM involves searching for the decision boundary (separating hyperplane) that leads to the maximal margin
- The training instances that fall on the margin are known as the support vectors and these define the decision boundary



SVM

- The separating hyperplane is defined as $w_0 + \mathbf{w} \cdot \mathbf{d} = 0$
- For instances above the separating hyperplane $w_0 + \mathbf{w} \cdot \mathbf{d} > 0$
- For instances below the separating hyperplane $w_0 + \mathbf{w} \cdot \mathbf{d} < 0$
- An SWM model is built so that instances with a negative target class result in the model outputting ≤ -1 and instances with the positive class output ≥ +1

- An SVM model is defined as

$$\mathbb{M}(\mathbf{q}) = \sum_{i=1}^{s} (t_i \times \boldsymbol{\alpha}[i] \times (\mathbf{d}_i \cdot \mathbf{q}) + w_0)$$

where $(\mathbf{d}_1, t_1), \ldots, (\mathbf{d}_s, t_s)$ are the support vectors,

Lagrange multipliers

$\boldsymbol{\alpha}$ is a set of parameters determined during the training process, one for each support vector $\boldsymbol{\alpha}[1], \ldots, \boldsymbol{\alpha}[s]$

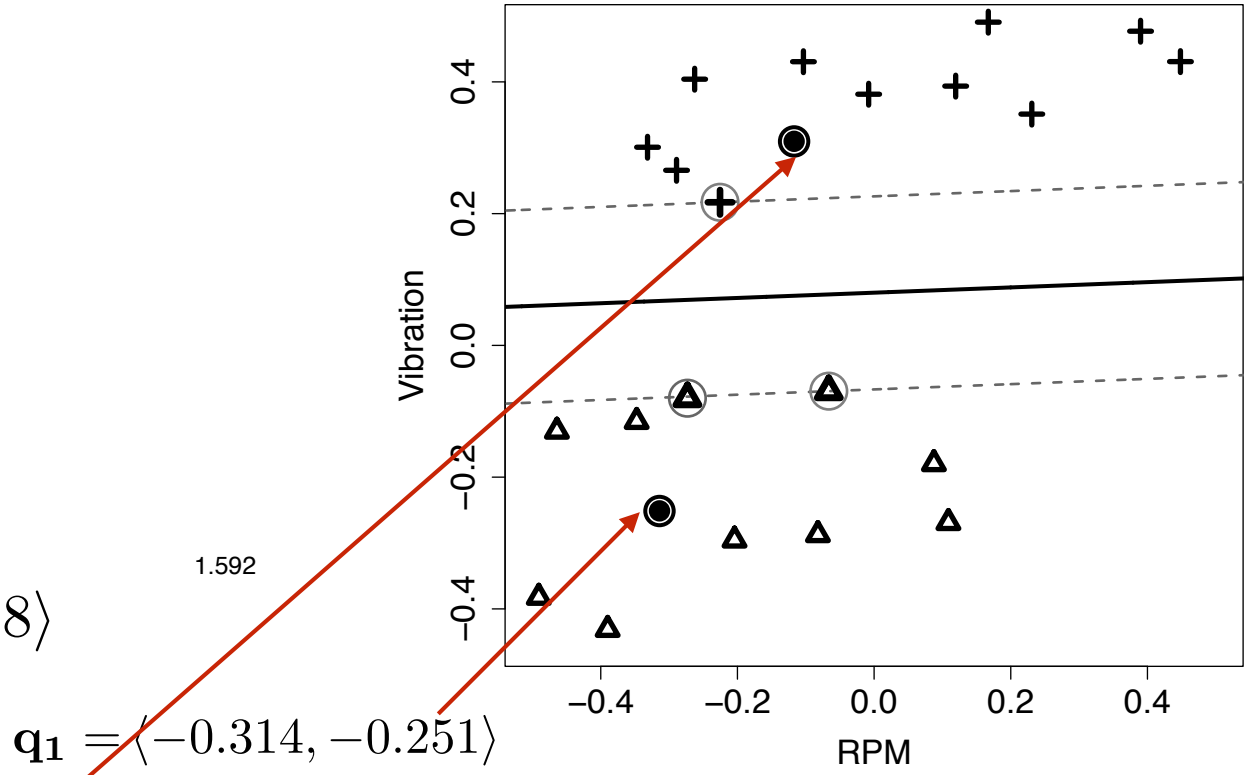
SVM example

- Support vectors are:

$$(\langle -0.225, 0.217 \rangle, +1)$$
$$(\langle -0.066, -0.069 \rangle, -1)$$
$$(\langle -0.273, -0.080 \rangle, -1)$$

- w0 is -0.1838
- α parameters

$$\langle 23.056, 6.998, 16.058 \rangle$$



$$\mathbf{q_1} = \langle -0.314, -0.251 \rangle$$
$$\mathbf{q_2} = \langle -0.117, 0.31 \rangle$$

$$\begin{aligned}\mathbb{M}(\mathbf{q_1}) = &(1 \times 23.056 \times ((-0.225 \times -0.314) + (0.217 \times -0.251)) - 0.1838) \\ &+ (-1 \times 6.998 \times ((-0.066 \times -0.314) + (-0.069 \times -0.251)) - 0.1838) \\ &+ (-1 \times 16.058 \times ((-0.273 \times -0.314) + (-0.080 \times -0.251)) - 0.1838) \\ = &-2.145 \quad \Rightarrow \text{predict negative}\end{aligned}$$

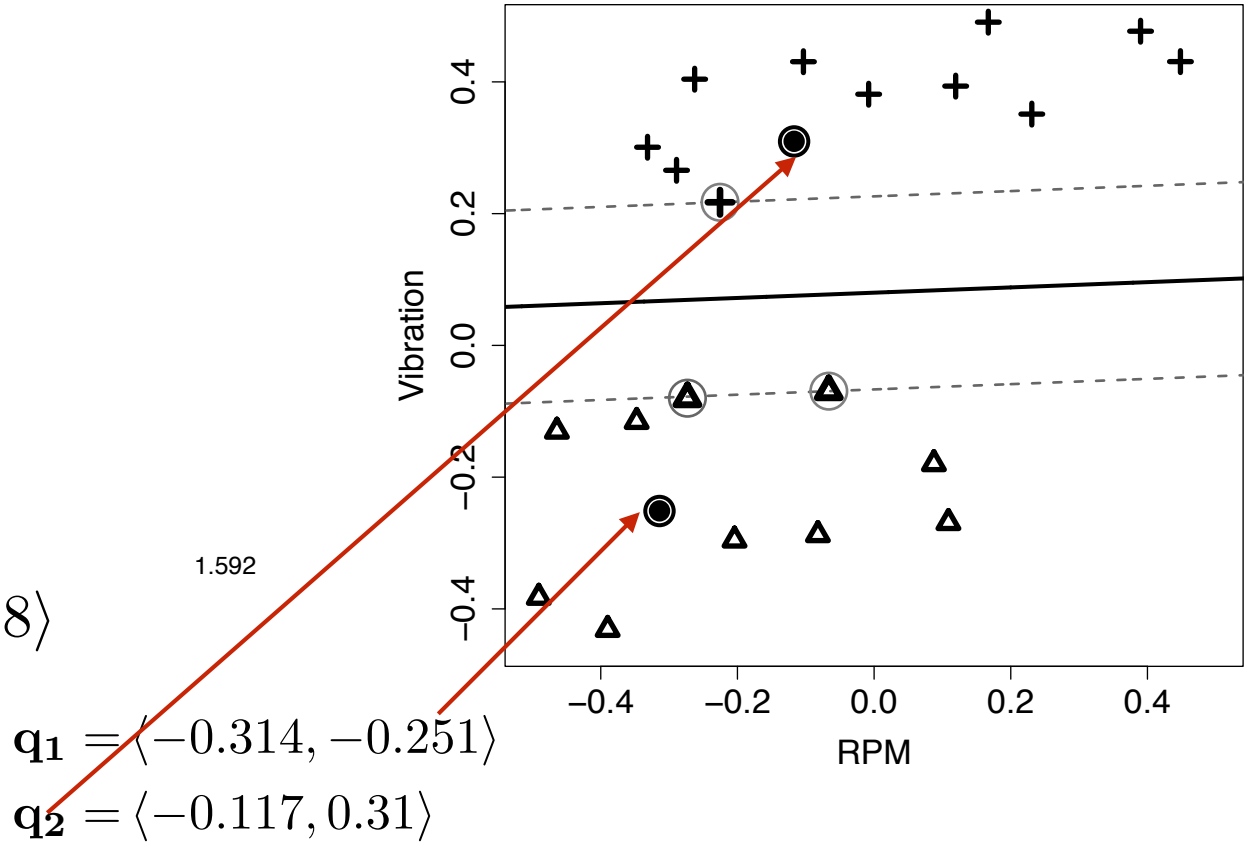
SVM example

- Support vectors are:

$$(\langle -0.225, 0.217 \rangle, +1)$$
$$(\langle -0.066, -0.069 \rangle, -1)$$
$$(\langle -0.273, -0.080 \rangle, -1)$$

- w0 is -0.1838
- α parameters

$$\langle 23.056, 6.998, 16.058 \rangle$$



$\mathbb{M}(\mathbf{q_2}) = 1.592 \quad \Rightarrow$ predict positive

SVM kernel functions

- To handle training data that is not linearly separable SVM use a kernel function

$$\mathbb{M}(\mathbf{q}) == \sum_{i=1}^{s} (t_i \times \boldsymbol{\alpha}[i] \times kernel\,(\mathbf{d}_i, \mathbf{q}) + w_0)$$

- Popular options for the kernel function include:

Linear kernel	$kernel(\mathbf{d}, \mathbf{q}) = \mathbf{d} \cdot \mathbf{q} + c$ where c is an optional constant
Polynomial kernel	$kernel(\mathbf{d}, \mathbf{q}) = (\mathbf{d} \cdot \mathbf{q} + 1)^p$ where p is the degree of a polynomial function
Gaussian radial basis kernel	$kernel(\mathbf{d}, \mathbf{q}) = \exp(-\gamma \lvert\lvert \mathbf{d} - \mathbf{q} \rvert\rvert^2)$ where γ is a manually chosen tuning parameter

- Selected by experimentation with different options…

SVMs

- A very popular approach to building predictive models in recent times

+ Quickly trained,

+ Not overly susceptible to overfitting,

+ Work well for high dimensional data

- Binary models, need an one-versus-all ensemble approach for multi-class classification

- Not very interpretable, particularly when kernel functions are used