Data Mining and Machine Learning

Comp 3027J

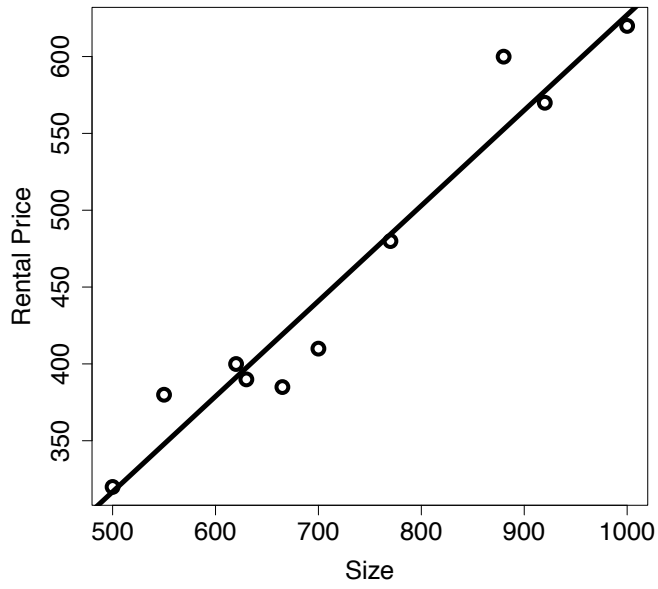
Dr Catherine Mooney
Assistant Professor

catherine.mooney@ucd.ie

Lectures and Text

- **Core Text:**
  Fundamentals of Machine Learning for Predictive Data Analytics
  By John D. Kelleher, Brian Mac Namee and Aoife D'Arcy
- Last lecture we covered Chapter 7, sections 7.2 and 7.3 (Error-based Learning – Linear Regression and Gradient Descent).
- This week we will cover Chapter 7, sections 7.4.4, 7.4.6 and 7.4.7 (Error-based Learning – Logistic Regression, Multinomial Logistic Regression, and Support Vector Machines).
- Please read these sections of the book.

Review – Simple Linear Regression



- A scatter plot of the SIZE and RENTAL PRICE features from the office rentals dataset with a simple linear model added to capture the relationship.
- This model is:

$$\begin{aligned} \text{RENTAL PRICE} &= 0.62 \times \text{SIZE} + 6.47 \\ y &= mx + b \end{aligned}$$

This kind of model is known as a **simple linear regression model**. This approach to modeling the relationships between features is extremely common in both machine learning and statistics.

We can rewrite the simple linear regression model as

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

- the parameters $\mathbf{w}[0]$ and $\mathbf{w}[1]$ are referred to as weights
- $\mathbf{d}$ is an instance defined by a single descriptive feature $\mathbf{d}[1]$
- $\mathbb{M}_{\mathbf{w}}(d)$ is the prediction output by the model for the instance $\mathbf{d}$

The error function

- The key to using simple linear regression models is determining the optimal values for the weights in the model.
- The optimal weights allow the model to capture the relationship between the descriptive features and a target feature.
- They are said to fit the training data.
- We need some way to measure how well a model fits a training dataset.
- We do this by defining an error function.
- An **error function** captures the error between the predictions made by a model and the actual values in a training dataset.

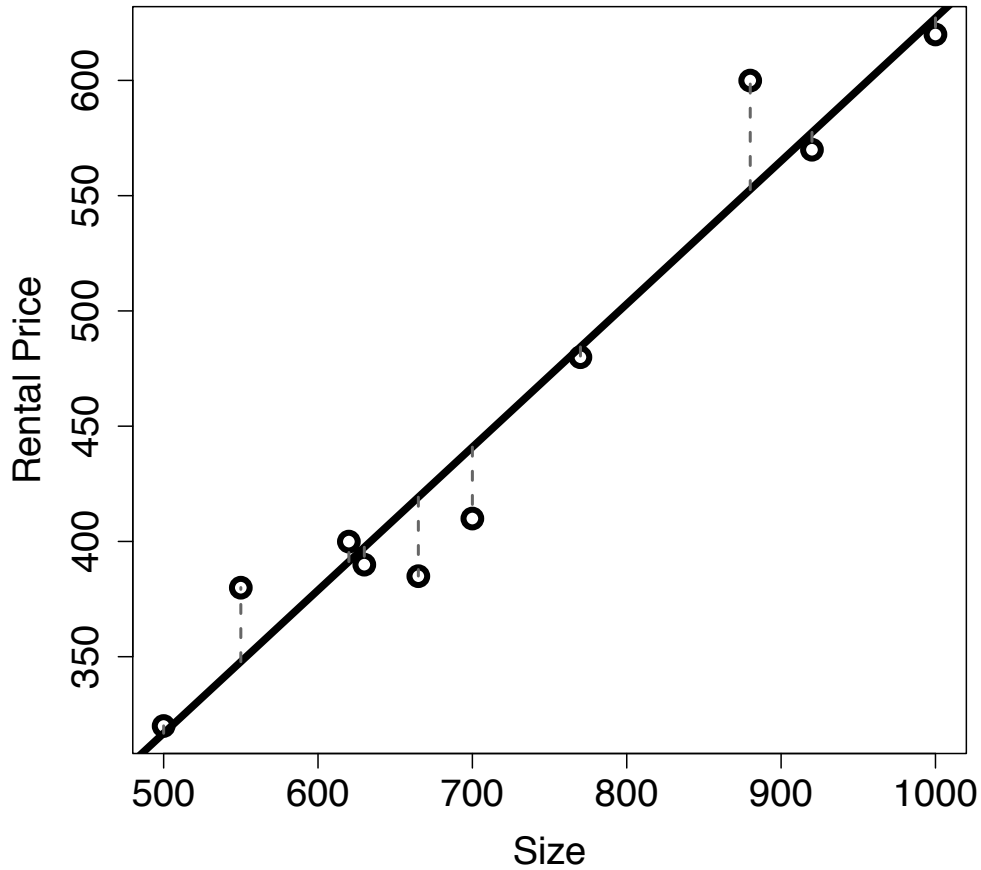


Figure: A scatter plot of the SIZE and RENTAL PRICE features from the office rentals dataset showing a candidate prediction model (with $\mathbf{w}[0] = 6.47$ and $\mathbf{w}[1] = 0.62$) and the resulting errors.

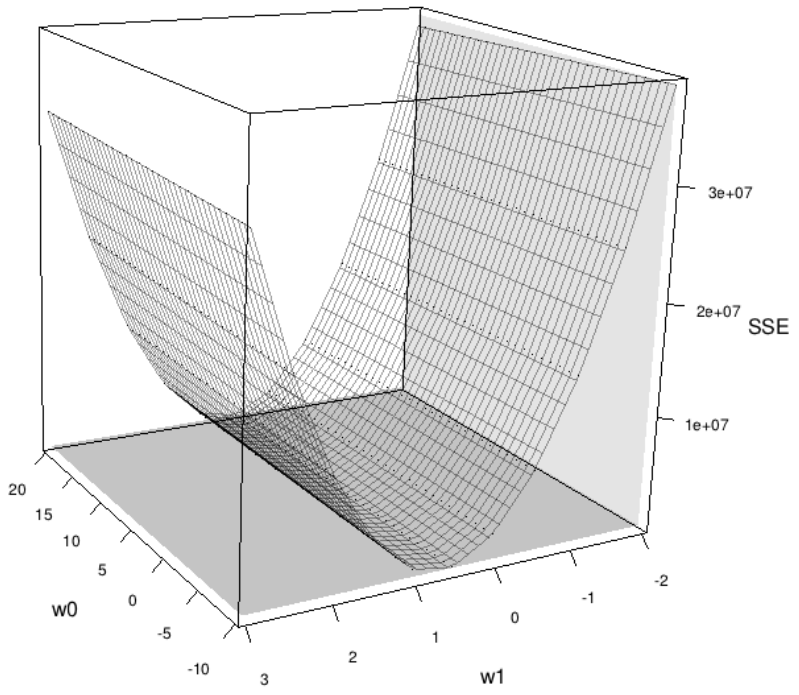
The sum of squared errors error function

- There are many different kinds of error functions
- For measuring the fit of simple linear regression models, the most commonly used is the sum of squared errors error function, or SSE.
- To calculate SSE we make a prediction for each member of the training dataset and then calculate the error between these predictions and the actual target feature values in the training set.

The sum of squared errors error function, SSE, is formally defined as

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

- 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.
- We can think about all these error values joined to make a surface defined by the weight combinations



A 3D surface plot of the error surface generated by plotting the sum of squared errors value for the office rentals training set for each possible combination of values for $\mathbf{w}[0]$ (from the range $[-10, 20]$) and $\mathbf{w}[1]$ (from the range $[-2, 3]$).

- The *x*-*y* plane is known as a **weight space** and the surface is known as an **error surface**.
- The model that best fits the training data is the model corresponding to the lowest point on the error surface.

- Fortunately, these error surfaces have two properties that help us find the optimal combination of weights
  1. they are convex
  2. they have a global minimum
- If we can find the global minimum of the error surface, we can find the set of weights defining the model that best fits the training dataset.
- This approach to finding weights is known as least squares optimization.

- We can formally define this point on the error surface as the point at which:

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

and

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

- There are a number of different ways to find this point.
- The most common approach is known as the **gradient descent** algorithm.

Handling Categorical Target Features: Logistic Regression

The generators dataset

This dataset contains measurements of:

- The revolutions per minute (RPM) that power station generators are running at
- The amount of vibration in the generators (VIBRATION)
- An indicator to show whether the generators proved to be working or faulty the day after these measurements were taken
- If power station administrators could predict upcoming generator failures before the generators actually fail, they could improve power station safety and save money on maintenance

Table: A dataset listing features for a number of generators.

ID	RPM	VIBRATION	STATUS
1	568	585	good
2	586	565	good
3	609	536	good
4	616	492	good
5	632	465	good
6	652	528	good
7	655	496	good
8	660	471	good
9	688	408	good
10	696	399	good
11	708	387	good
12	701	434	good
13	715	506	good
14	732	485	good
15	731	395	good
16	749	398	good
17	759	512	good
18	773	431	good
19	782	456	good
20	797	476	good
21	794	421	good
22	824	452	good
23	835	441	good
24	862	372	good
25	879	340	good
26	892	370	good
27	913	373	good
28	933	330	good

ID	RPM	VIBRATION	STATUS
29	562	309	faulty
30	578	346	faulty
31	593	357	faulty
32	626	341	faulty
33	635	252	faulty
34	658	235	faulty
35	663	299	faulty
36	677	223	faulty
37	685	303	faulty
38	698	197	faulty
39	699	311	faulty
40	712	257	faulty
41	722	193	faulty
42	735	259	faulty
43	738	314	faulty
44	753	113	faulty
45	767	286	faulty
46	771	264	faulty
47	780	137	faulty
48	784	131	faulty
49	798	132	faulty
50	820	152	faulty
51	834	157	faulty
52	858	163	faulty
53	888	91	faulty
54	891	156	faulty
55	911	79	faulty
56	939	99	faulty

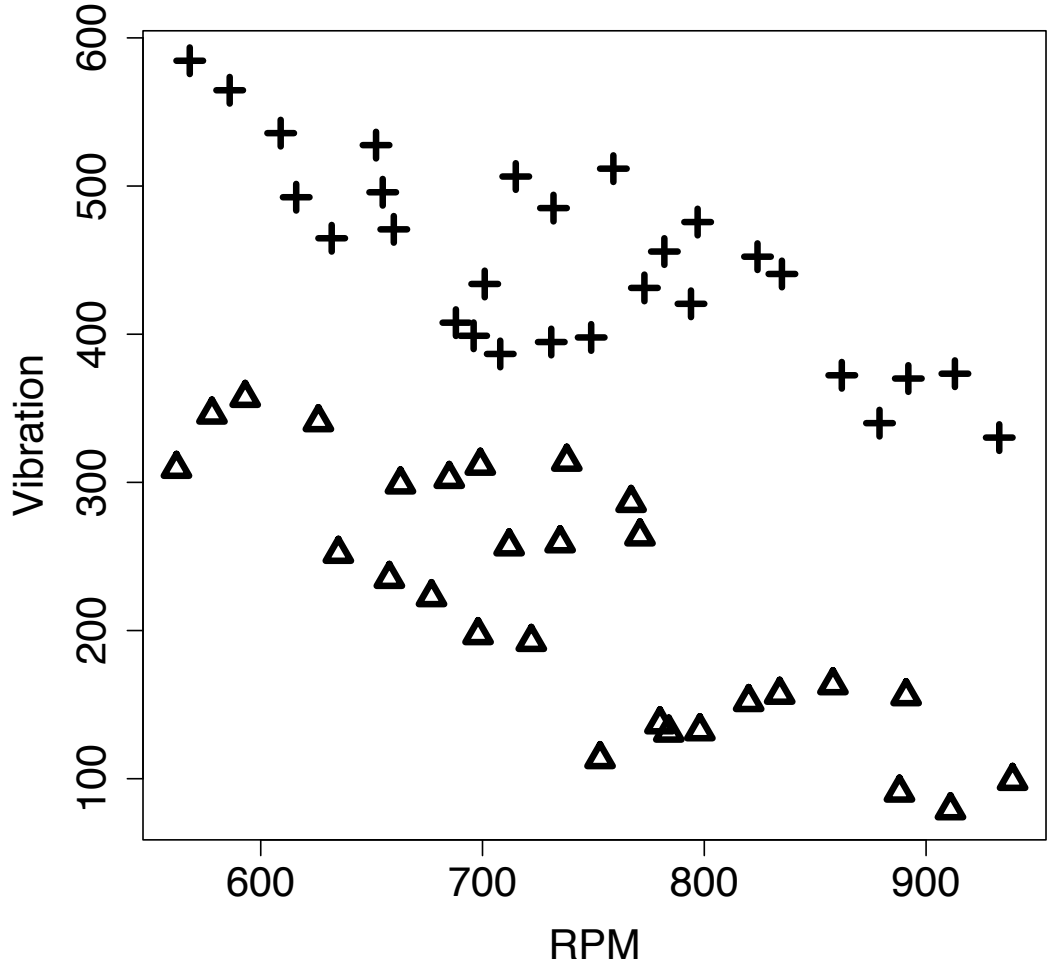


Figure: A scatter plot of the RPM and VIBRATION descriptive features from the generators dataset where *'good'* generators are shown as crosses and *'faulty'* generators are shown as triangles.

- We can draw a straight line across the scatter plot that perfectly separates the good generators from the faulty ones
- This line is known as a **decision boundary**
- Because we can draw this line, this dataset is said to be linearly separable in terms of the two descriptive features used

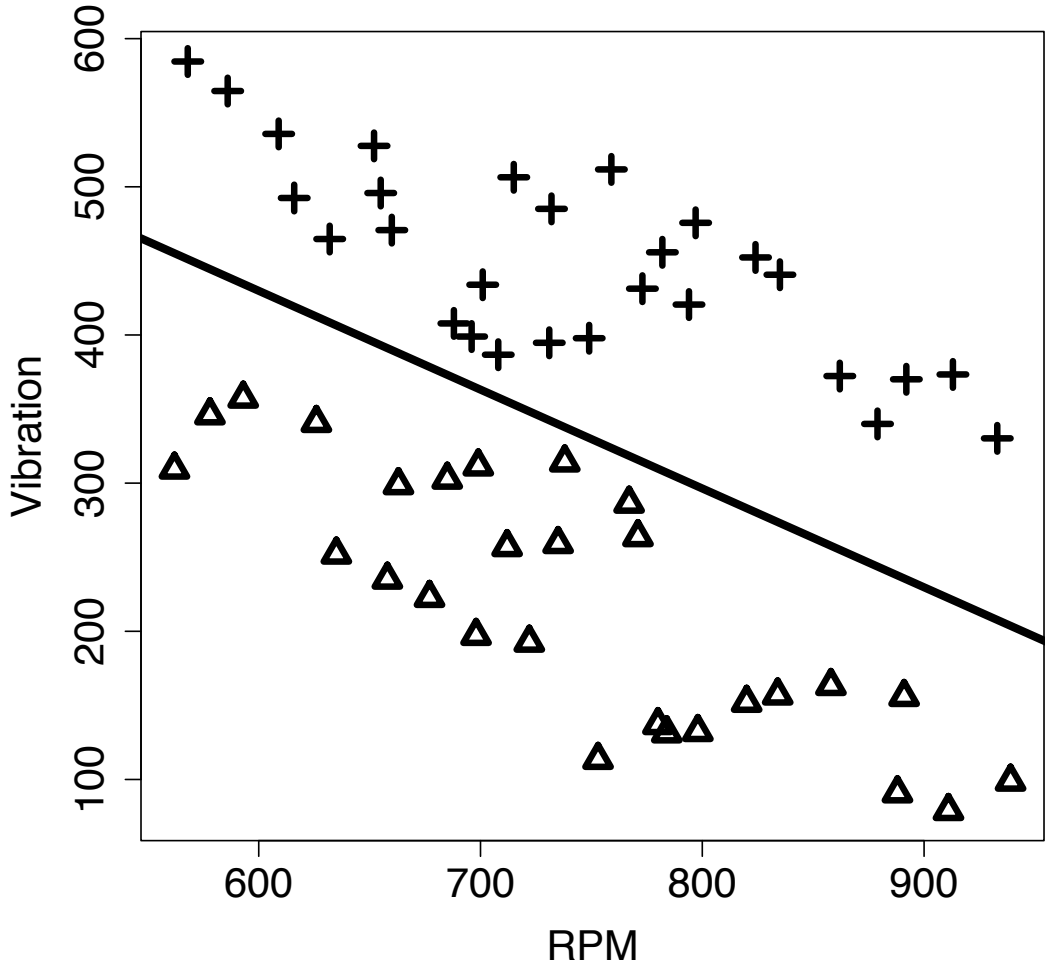


Figure: A scatter plot of the RPM and VIBRATION descriptive features from the generators dataset. A decision boundary separating *'good'* generators (crosses) from *'faulty'* generators (triangles) is also shown.

- As the decision boundary is a **linear separator** it can be defined using the equation of the line as:

$$\text{VIBRATION} = 830 - 0.667 \times \text{RPM} \tag{1}$$

or

$$830 - 0.667 \times \text{RPM} - \text{VIBRATION} = 0 \tag{2}$$

- Applying Equation 2 to the instance RPM $= 810$, VIBRATION $= 495$, which is be above the decision boundary, gives the following result:

$$830 - 0.667 \times 810 - 495 = -205.27$$

- By contrast, if we apply Equation 2 to the instance RPM $= 650$ and VIBRATION $= 240$, which is below the decision boundary, we get

$$830 - 0.667 \times 650 - 240 = 156.45$$

- All the data points above the decision boundary will result in a negative value when plugged into the decision boundary equation, while all data points below the decision boundary will result in a positive value.

- Because the values of this equation are so well behaved, we can use it to predict a categorical target feature
- Reverting to our previous notation we have:

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & \textit{otherwise} \end{cases} \tag{3}$$

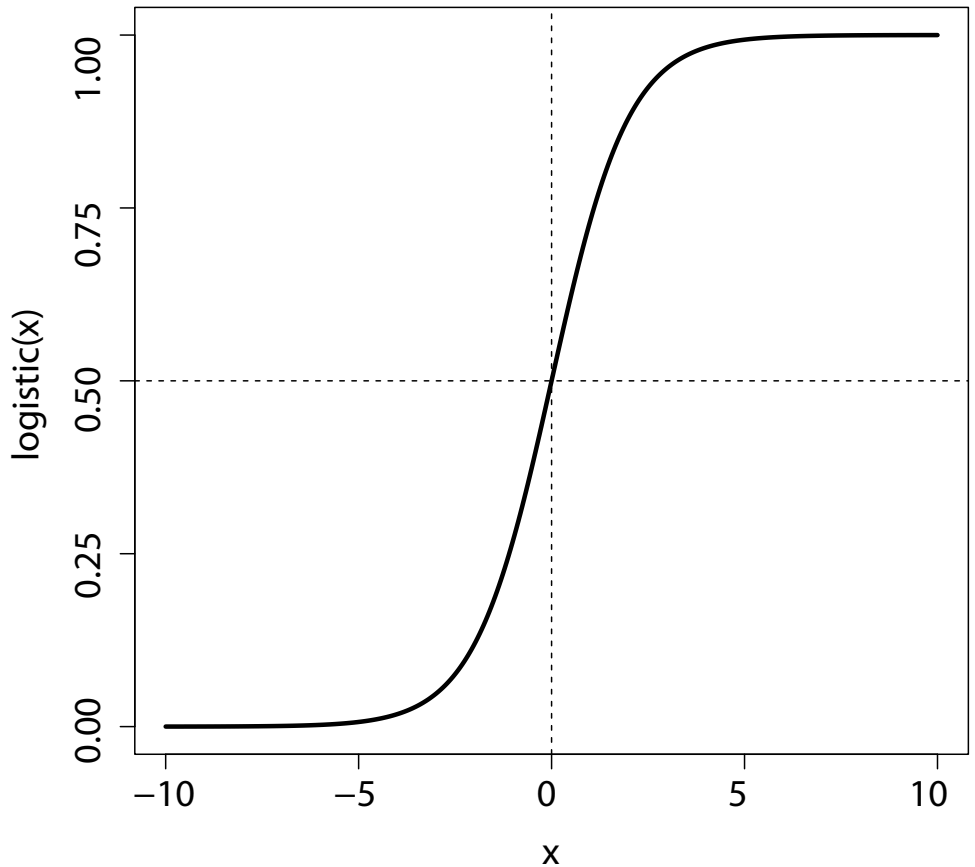
  where **d** is a set of descriptive features for an instance, **w** is the set of weights in the model, and the GOOD and FAULTY generator target feature levels are represented as 0 and 1 respectively
- The surface defined by this rule is known as a **decision surface**.

- The hard decision boundary given in Equation 3 is **discontinuous** so is not differentiable and so we can't calculate the gradient of the error surface.
- Furthermore, the model always makes completely confident predictions of 0 or 1, whereas a little more subtlety is desirable.
- We address these issues by using a more sophisticated threshold function that is continuous, and therefore differentiable, and that allows for the subtlety desired: the **logistic function**

logistic function

$$Logistic(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

where x is a numeric value and e is **Euler's number** and is approximately equal to 2.7183.



The logistic function is a threshold function that pushes values above zero to 1 and values below zero to 0. This is very similar to the hard threshold function, except that it has a soft boundary.

- To build a logistic regression model, we simply pass the output of the basic linear regression model through the logistic function

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

A note on training logistic regression models:

1. Before we train a logistic regression model we map the binary target feature levels to 0 or 1.
2. The error of the model on each instance is then the difference between the target feature (0 or 1) and the value of the prediction $[0, 1]$.

Example

$$\mathbb{M}_{\mathbf{w}}(\langle \text{RPM}, \text{VIBRATION} \rangle) = \frac{1}{1 + e^{-(-0.4077 + 4.1697 \times \text{RPM} + 6.0460 \times \text{VIBRATION})}}$$

- To repurpose the gradient descent algorithm for training logistic regression models the only change that needs to be made is in the weight update rule.
- See pg. 360 in your book for details of how to derive the new weight update rule.
- The new weight update rule is:

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

- For logistic regression models we recommend that descriptive feature values always be normalized.
- In this example, before the training process begins, both descriptive features are normalized to the range $[-1, 1]$.

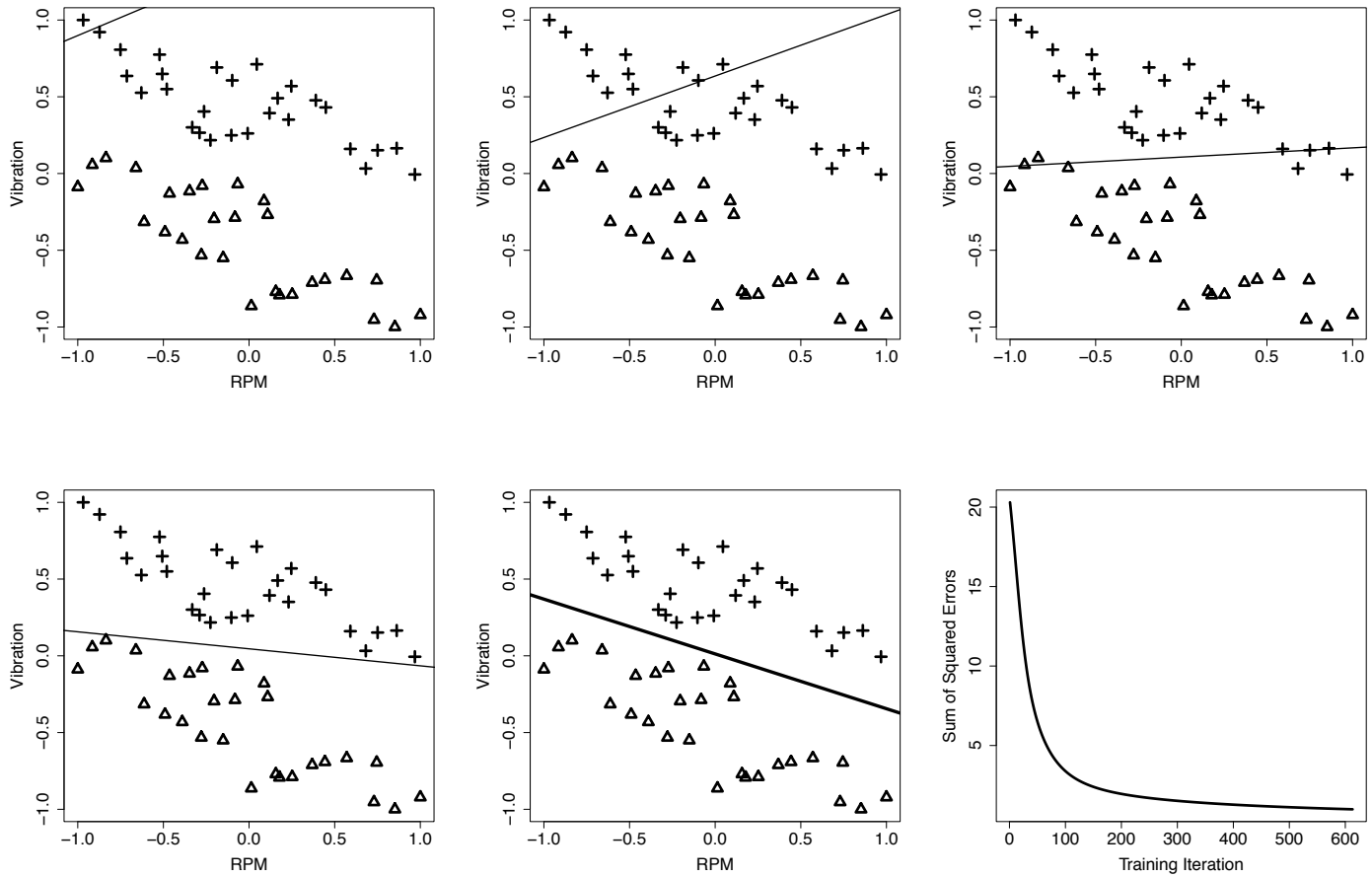


Figure: A selection of the logistic regression models developed during the gradient descent process for the machinery dataset. The bottom-right panel shows the sum of squared error values generated during the gradient descent process.

Multinomial Logistic Regression

Multinomial Logistic Regression

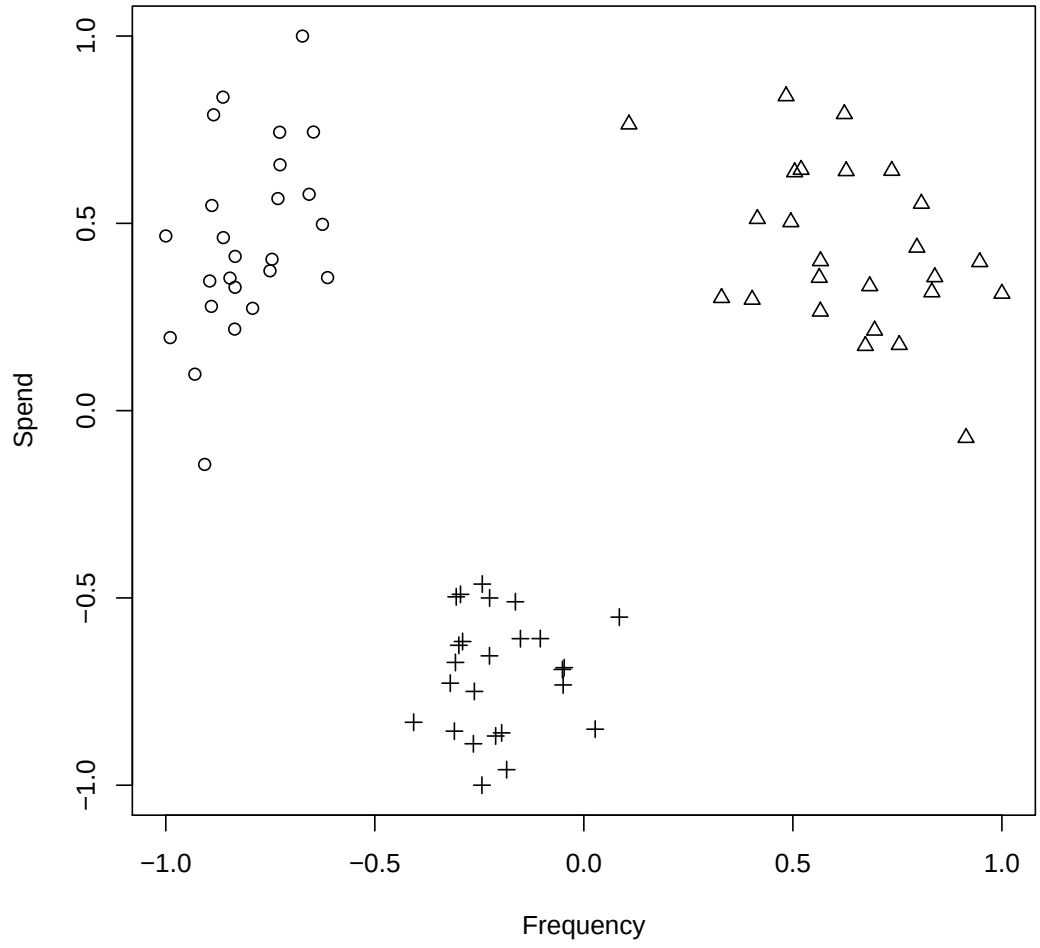
- Handles categorical target features with more than two levels
- A good way to build multinomial logistic regression models is use a set of **one-versus-all** models
- If we have *r* target levels, we create *r* one-versus-all logistic regression models
- A one-versus-all model distinguishes between one level of the target feature and all the others

The Customer Type Dataset – details of customers' shopping habits with a large national retail chain (e.g. Wal-Mart)

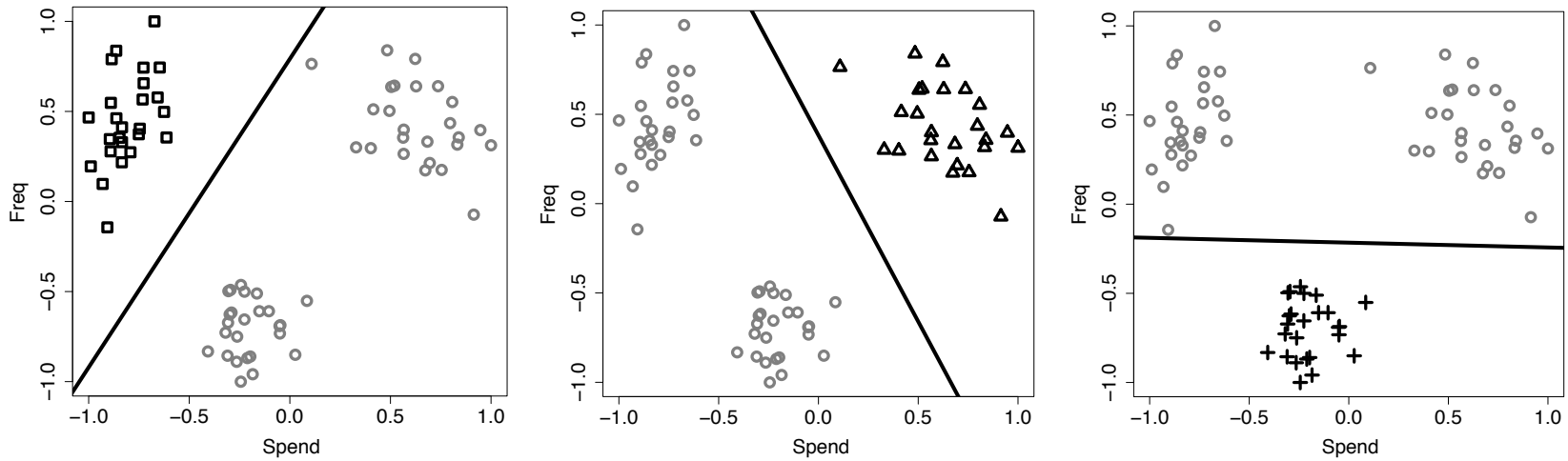
- Each customer's average weekly spending with the chain store, SPEND
- Average number of visits per week to the chain store, FREQ
- The TYPE of customer: single, business, or family

Table: A dataset of customers of a large national retail chain.

ID	SPEND	FREQ	TYPE
1	21.6	5.4	single
2	25.7	7.1	single
3	18.9	5.6	single
4	25.7	6.8	single
		⋮	
26	107.9	5.8	business
27	92.9	5.5	business
28	122.6	6.0	business
29	107.7	5.7	business
		⋮	
47	53.2	2.6	family
48	52.4	2.0	family
49	46.1	1.4	family
50	65.3	2.2	family



The customer type dataset with three target levels: single (circles), business (triangles), and family (crosses) (after the data had been range normalized to [-1, 1]).



(a) (b) (c)

Figure: An illustration of three different **one-versus-all** prediction models for the customer type dataset that has three target levels *'single'* (squares), *'business'* (triangles) and *'family'* (crosses).

- For r target feature levels, we build r separate logistic regression models $\mathbb{M}_{\mathbf{w_1}}$ to $\mathbb{M}_{\mathbf{w_r}}$:

$$\begin{aligned}
\mathbb{M}_{\mathbf{w_1}}(\mathbf{d}) &= \mathit{logistic}(\mathbf{w_1} \cdot \mathbf{d}) \\
\mathbb{M}_{\mathbf{w_2}}(\mathbf{d}) &= \mathit{logistic}(\mathbf{w_2} \cdot \mathbf{d}) \\
&\vdots \\
\mathbb{M}_{\mathbf{w_r}}(\mathbf{d}) &= \mathit{logistic}(\mathbf{w_r} \cdot \mathbf{d})
\end{aligned}$$

  where $\mathbb{M}_{\mathbf{w_1}}$ to $\mathbb{M}_{\mathbf{w_r}}$ are r different one-versus-all logistic regression models, and $\mathbf{w_1}$ to $\mathbf{w_r}$ are r different sets of weights.

- To combine the outputs of these different models, we normalize their results using:

$$\mathbb{M}'_{\mathbf{w_k}}(\mathbf{d}) = \frac{\mathbb{M}_{\mathbf{w_k}}(\mathbf{d})}{\sum_{l \in levels(t)} \mathbb{M}_{\mathbf{w}_l}(\mathbf{d})}$$

where $\mathbb{M}'_{\mathbf{w_k}}(\mathbf{d})$ is a revised, normalized prediction for the one-versus-all model for the target level k.

- The r one-versus-all logistic regression models used are trained in parallel, and the revised model outputs, $\mathbb{M}'_{\mathbf{w_k}}(\mathbf{d})$, are used when calculating the sum of squared errors for each model during the training process.
- This means that the sum of squared errors function is changed slightly to

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

- The revised predictions are also used when making predictions for query instances. The predicted level for a query, **q**, is the level associated with the one-versus-all model that outputs the highest result after normalization.
- We can write this as

$$\mathbb{M}(\mathbf{q}) = \underset{l \in levels(t)}{\text{argmax}}\, \mathbb{M}'_{\mathbf{w}_l}(\mathbf{q})$$

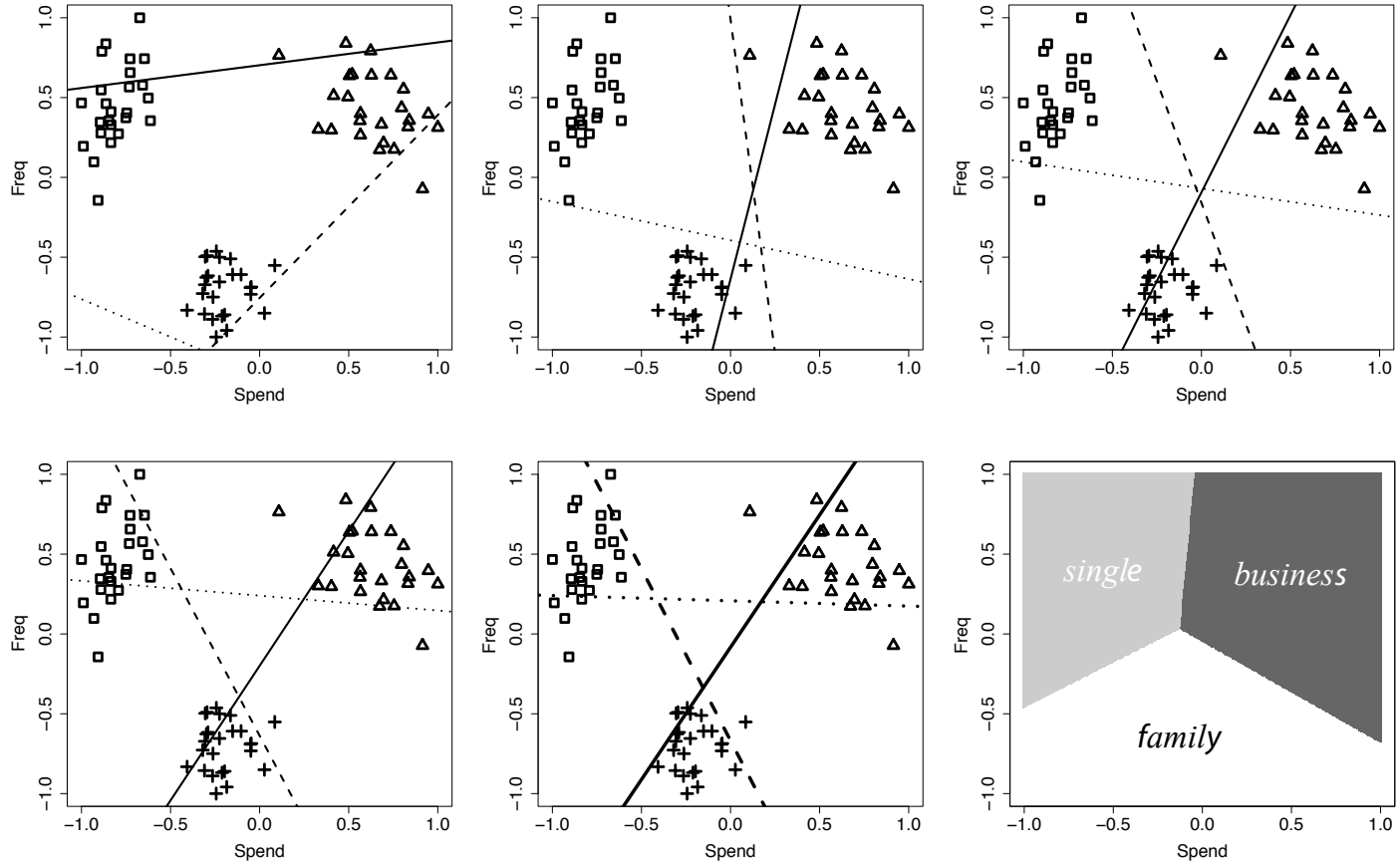


Figure: A selection of the models developed during the gradient descent process for the customer group dataset from Table 2 [38]. Squares represent instances with the *'single'* target level, triangles the *'business'* level and crosses the *'family'* level. (f) illustrates the overall decision boundaries that are learned between the three target levels.

Support Vector Machines

- Support vector machines (SVM) are another approach to predictive modeling that is based on error-based learning.
- Training a support vector machine involves searching for the decision boundary, or **separating hyperplane**, that leads to the maximum margin as this will best separate the levels of the target feature.

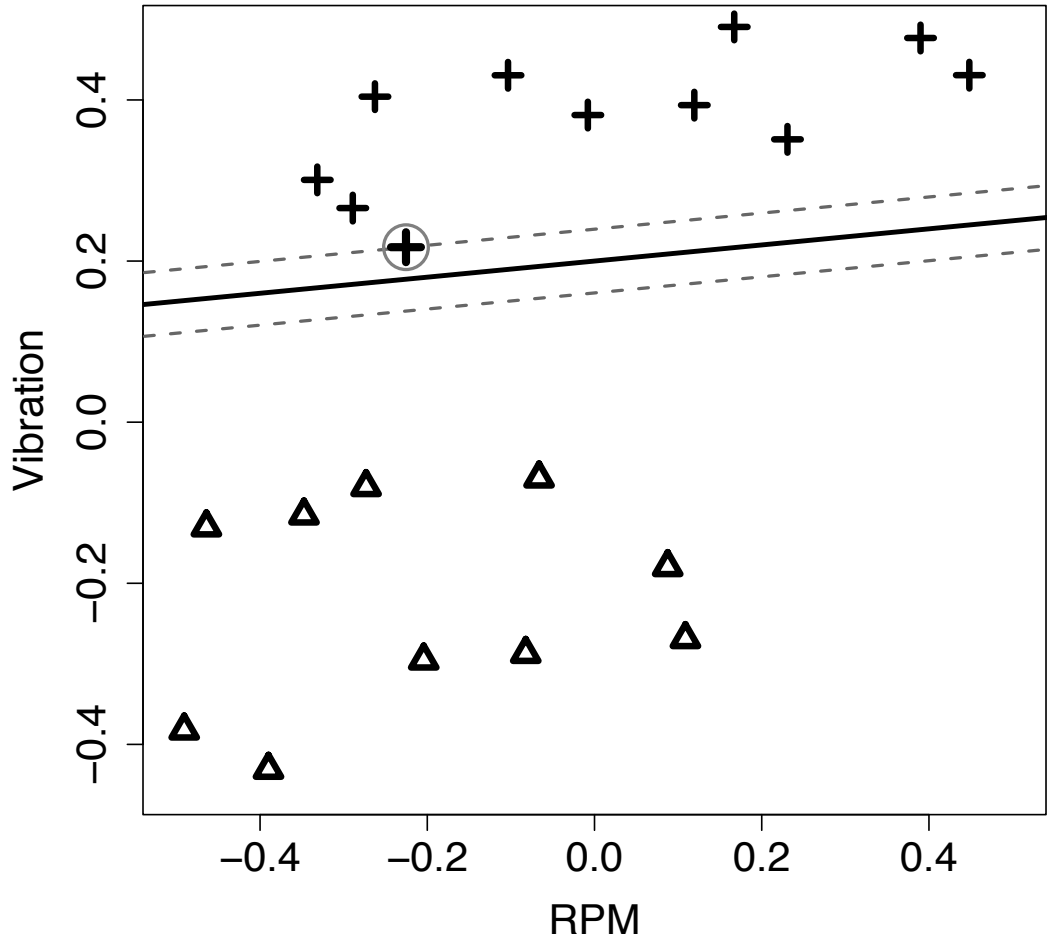


Figure: A small sample of the generators dataset with two features, RPM and VIBRATION, and two target levels, *'good'* (shown as crosses) and *'bad'* (shown as triangles). A decision boundary with a very small margin.

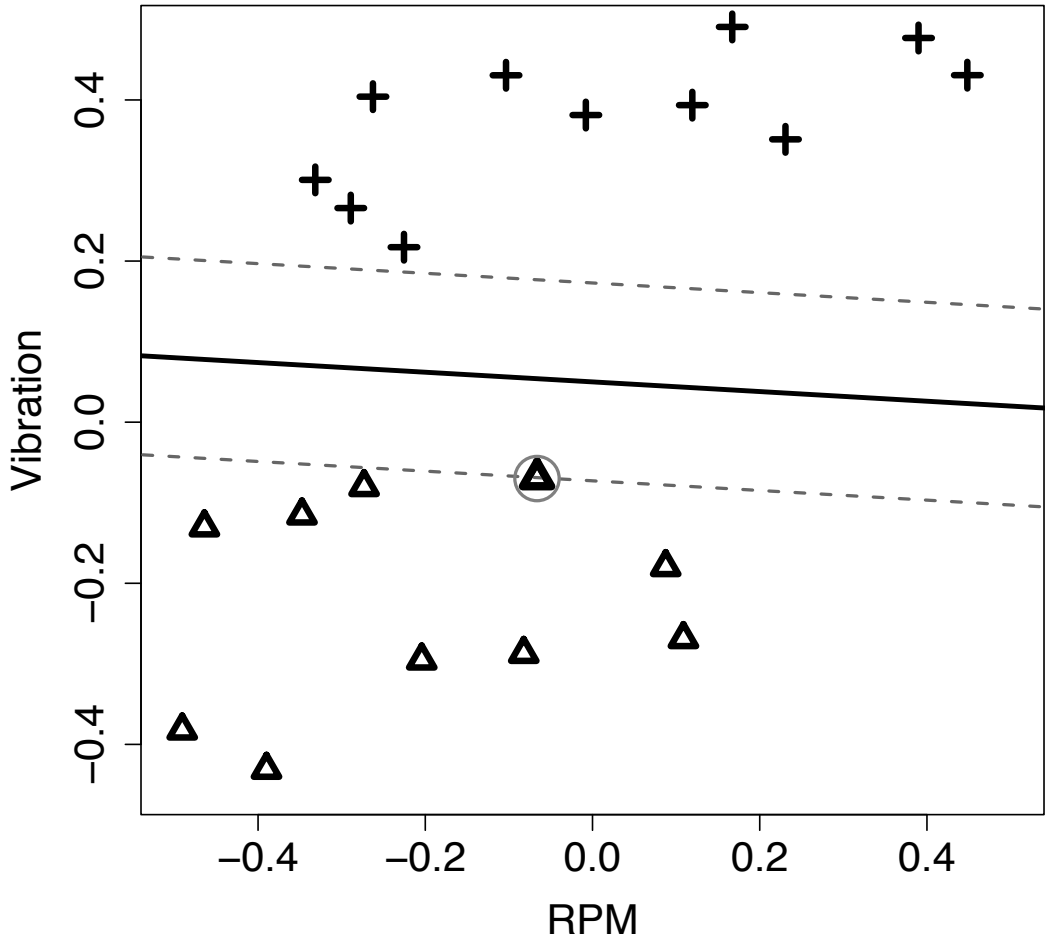


Figure: A small sample of the generators dataset with two features, RPM and VIBRATION, and two target levels, *'good'* (shown as crosses) and *'bad'* (shown as triangles). A decision boundary with a large margin.

- The intuition behind support vector machines is that this second decision boundary should distinguish between the two target levels much more reliably than the first.
- The instances in a training dataset that fall along the margin extents, and so define the margins, are known as the **support vectors** and define the decision boundary.

- We define the separating hyperplane as follows:

$$w_0 + \mathbf{w} \cdot \mathbf{d} = 0$$

- For instances above a separating hyperplane

$$w_0 + \mathbf{w} \cdot \mathbf{d} > 0$$

and for instances below a separating hyperplane

$$w_0 + \mathbf{w} \cdot \mathbf{d} < 0$$

- We build a support vector machine prediction model so that instances with the negative target level result in the model outputting ≤ -1 and instances with the positive target level result in the model outputting $\geq +1$.
- The space between the outputs of -1 and $+1$ allows for the margin.

- Training a support vector machine is framed as a **constrained quadratic optimization problem**
- This type of problem is defined in terms of:
  1. a set of constraints
  2. an optimization criterion.

- The **constraints** required by the training process are

$$w_0 + \mathbf{w} \cdot \mathbf{d} \leq -1 \text{ for } t_i = -1$$

and:

$$w_0 + \mathbf{w} \cdot \mathbf{d} \geq +1 \text{ for } t_i = +1$$

- We can combine these two constraints into a single constraint (remember t_i is always equal to either -1 or $+1$):

$$t_i \times (w_0 + \mathbf{w} \cdot \mathbf{d}) \geq 1$$

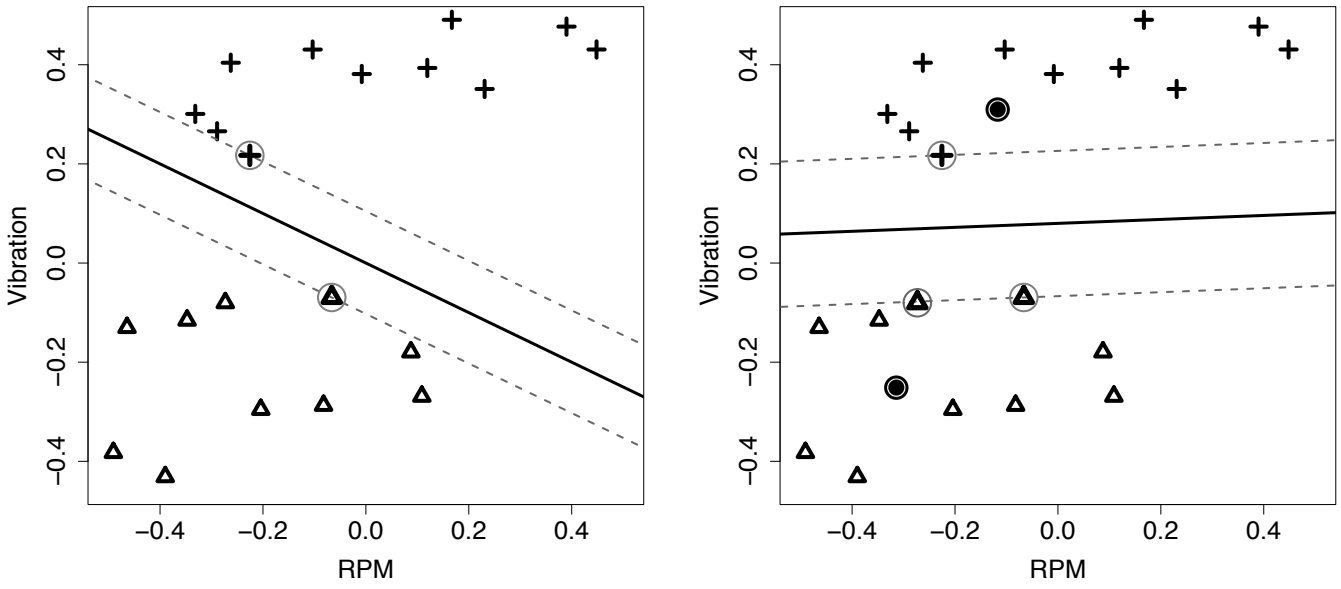


Figure: Different margins that satisfy the constraint. The instances that define the margin are highlighted in each case. (b) shows the maximum margin and also shows two query instances represented as black dots.

- The **optimization** criterion used is defined in terms of the perpendicular distance from any instance to the decision boundary and is given by

$$dist(\mathbf{d}) = \frac{w_0 + abs(\mathbf{w} \cdot \mathbf{d})}{||\mathbf{w}||}$$

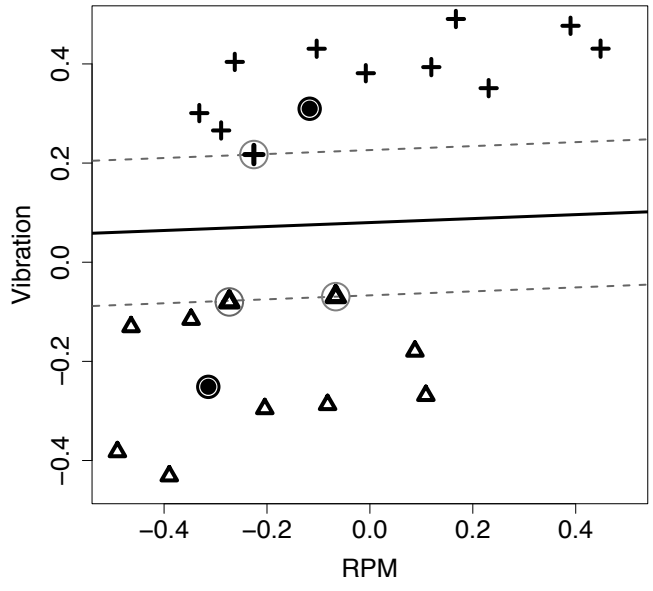
  where $||\mathbf{w}||$ is known as the **Euclidean norm** of $\mathbf{w}$ and is calculated as

$$||\mathbf{w}|| = \sqrt{\mathbf{w}[1]^2 + \mathbf{w}[2]^2 + \ldots + \mathbf{w}[m]^2}$$

- For instances along the **margin extents**, $abs(\mathbf{w} \cdot \mathbf{d} + w_0) = 1$.
- So, the distance from any instance along the margin extents to the decision boundary is $\frac{1}{||\mathbf{w}||}$, and because the margin is symmetrical to either side of the decision boundary, the size of the margin is $\frac{2}{||\mathbf{w}||}$.

- The goal when training a support vector machine is
  1. maximize $\frac{2}{||\mathbf{w}||}$
  2. subject to the constraint

$$t_i \times (w_0 + \mathbf{w} \cdot \mathbf{d}) \geq 1$$



- The optimal decision boundary and associated support vectors for the example we have been following
- In this case *'good'* is the positive level and set to $+1$, and *'faulty'* is the negative level and set to -1.

- **Kernel functions** can be used with support vector machines to handle data that is not **linearly separable**
- A wide range of standard kernel functions can be used including:

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	$kernel(\mathbf{d}, \mathbf{q}) = \exp(-\gamma \|\|\mathbf{d} - \mathbf{q}\|\|^2)$ where γ is a manually chosen tuning parameter

- The appropriate kernel function for a particular prediction model should be selected by experimenting with different options.
- It is best to start with a simple linear or low-degree polynomial kernel function and move to more complex kernel functions only if good performance cannot be achieved.

- So far we have assumed that it is possible to separate the instances with the two different target feature levels with a linear hyperplane
- Sometimes this is not possible, even after using a kernel function to move the data to a higher dimensional feature space
- In these instances, a margin cannot be defined
- An extension of the standard support vector machine approach that allows a **soft margin**
- This and allows overlap between instances with target features of the two different levels.

- Another extension allows support vector machines to handle multinomial target features using a one-versus-all approach similar to that described for Multinomial Logistic Regression
- There are also extensions to handle categorical descriptive features and continuous target features.

- Support vector machines have become a very popular approach to building predictive models
- They can be quickly trained
- They are not overly susceptible to overfitting
- They work well for high-dimensional data
- However, in contrast to logistic regression models, they are not very easy to interpret
- It is very difficult to understand why a particular prediction has been made – Especially when kernel functions are used.

Recommended Reading

- **Core Text:**
  Fundamentals of Machine Learning for Predictive Data Analytics
  By John D. Kelleher, Brian Mac Namee and Aoife D'Arcy
- This week we covered Chapter 7, sections 7.4.4, 7.4.6 and 7.4.7 (Error-based Learning – Logistic Regression, Multinomial Logistic Regression, and Support Vector Machines).
- I would suggest that you would read over these sections again.
- Email me if you have any questions and I will cover them at the beginning of class next week.

Review of Lab 8

Review of Lab 8

- Biggest problem p-values?
- I have put a recent paper on Moodle – Tatonetti_2019.pdf
- “Translational medicine in the Age of Big Data”
- Discusses *p*-values
- Also *Invalid validation*
- “Once these models are trained, it is essential that an honest evaluation of their performance is conducted. There are many tools for this evaluation including cross, hold-out, leave-one-out and out-of-bag validation depending on the model being used. The accuracy of these methods at estimating the performance is dependent on an assumption of the independence of the training examples, which is commonly violated in large biomedical data sets.”

Preview of Lab 9

Preview of Lab 9

- In Lab 9 we will be using R for Error-based Learning
- We will use a Support Vector Machine to predict upcoming generator failures
- We will also use a number of different Performance Measures to evaluate your Support Vector Machine
- You should download the following packages in advance:
  - install.packages("class")
  - install.packages("caret")
- Also download the generators dataset on Moodle

Questions?