Computer Vision

Philip O. Ogunbona

Advanced Multimedia Research Lab
EIS University of Wollongong

Introduction to Classifiers for Computer Vision[1]
Spring Session 2018

[1] The materials on Bayes' Classifier, KNN and SVM are based on the slides from the book by Kellleher et al. (2015)

Outline

Why are classifiers useful in Computer Vision?

Computer Vision can be described as follows (British Machine Vision Association [http://www.bmva.org/visionoverview]):

- Humans use their eyes and their brains to see and visually sense the world around them.
- Computer vision is the science that aims to give a similar, if not better, capability to a machine or computer.
- Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images.
- It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding.

Table 1: Some Applications of Computer Vision

Agriculture	Augmented Reality	Autonomous Vehicles
Biometrics	Character Recognition	Forensics
Industrial Quality Inspection	Face Recognition	Gesture Recognition
Geosciences	Pollution Monitoring	Transport
Medical Image Analysis	Image Restoration	Remote Sensing
Process Control	Security and Surveillance	Robotics

Some of these applications will involve solving classification problems.

Bayesian Classifiers - Introduction

Table 2: A simple dataset for MENINGITIS diagnosis with descriptive features that describe the presence or absence of three common symptoms of the disease: HEADACHE, FEVER, and VOMITING.

ID	HEADACHE	FEVER	VOMITING	MENINGITIS
1	true	true	false	false
2	false	true	false	false
3	true	false	true	false
4	true	false	true	false
5	false	true	false	true
6	true	false	true	false
7	true	false	true	false
8	true	false	true	true
9	false	true	false	false
10	true	false	true	true

Bayesian Classifiers - Introduction

- A **probability function**, $P()$, returns the probability of a feature taking a specific value.
- A **joint probability** refers to the probability of an assignment of specific values to multiple different features.
- A **conditional probability** refers to the probability of one feature taking a specific value given that we already know the value of a different feature
- A **probability distribution** is a data structure that describes the probability of each possible value a feature can take. The sum of a probability distribution must equal 1.0.
- A **joint probability distribution** is a probability distribution over more than one feature assignment and is written as a multi-dimensional matrix in which each cell lists the probability of a particular combination of feature values being assigned.
- The sum of all the cells in a joint probability distribution must be 1.0.

Bayesian Classifiers - Introduction

$$\mathbf{P}(H,F,V,M) = \begin{bmatrix} P(h,f,v,m), & P(\neg h,f,v,m) \\ P(h,f,v,\neg m), & P(\neg h,f,v,\neg m) \\ P(h,f,\neg v,m), & P(\neg h,f,\neg v,m) \\ P(h,f,\neg v,\neg m), & P(\neg h,f,\neg v,\neg m) \\ P(h,\neg f,v,m), & P(\neg h,\neg f,v,m) \\ P(h,\neg f,v,\neg m), & P(\neg h,\neg f,v,\neg m) \\ P(h,\neg f,\neg v,m), & P(\neg h,\neg f,\neg v,m) \\ P(h,\neg f,\neg v,\neg m), & P(\neg h,\neg f,\neg v,\neg m) \end{bmatrix}$$

Bayesian Classifiers - Introduction

- Given a joint probability distribution, we can compute the probability of any event in the domain that it covers by summing over the cells in the distribution where that event is true.
- Calculating probabilities in this way is known as **summing out**.

Bayesian Classifiers - Introduction

Bayes' Theorem

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Bayesian Classifiers - Introduction

Example 1

After a yearly checkup, a doctor informs their patient that he has both bad news and good news. The bad news is that the patient has tested positive for a serious disease and that the test that the doctor has used is 99% accurate (i.e., the probability of testing positive when a patient has the disease is 0.99, as is the probability of testing negative when a patient does not have the disease). The good news, however, is that the disease is extremely rare, striking only 1 in 10,000 people.

- What is the actual probability that the patient has the disease?
- Why is the rarity of the disease good news given that the patient has tested positive for it?

Bayesian Classifiers - Introduction

$$P(d|t) = \frac{P(t|d)P(d)}{P(t)}$$

$$\begin{aligned} P(t) &= P(t|d)P(d) + P(t|\neg d)P(\neg d) \\ &= (0.99 \times 0.0001) + (0.01 \times 0.9999) = 0.0101 \end{aligned}$$

$$\begin{aligned} P(d|t) &= \frac{0.99 \times 0.0001}{0.0101} \\ &= 0.0098 \end{aligned}$$

Bayesian Classifiers - Introduction

Deriving Bayes theorem

$$P(Y|X)P(X) = P(X|Y)P(Y)$$

$$\frac{P(X|Y)P(Y)}{P(Y)} = \frac{P(Y|X)P(X)}{P(Y)}$$

$$\frac{P(X|Y)\cancel{P(Y)}}{\cancel{P(Y)}} = \frac{P(Y|X)P(X)}{P(Y)}$$

$$\Rightarrow P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- The divisor is the prior probability of the evidence
- This division functions as a normalization constant.

$$0 \leq P(X|Y) \leq 1$$

$$\sum_i P(X_i|Y) = 1.0$$

- We can calculate this divisor directly from the dataset.

$$P(Y) = \frac{|\{\text{rows where Y is the case}\}|}{|\{\text{rows in the dataset}\}|}$$

- Or, we can use the **Theorem of Total Probability** to calculate this divisor.

$$P(Y) = \sum_i P(Y|X_i)P(X_i) \tag{1}$$

Generalized Bayes' Theorem

$$P(t = l|\mathbf{q}[1], \ldots, \mathbf{q}[m]) = \frac{P(\mathbf{q}[1], \ldots, \mathbf{q}[m]|t = l)P(t = l)}{P(\mathbf{q}[1], \ldots, \mathbf{q}[m])}$$

Chain Rule

$$P(\mathbf{q}[1],\ldots,\mathbf{q}[m]) = P(\mathbf{q}[1]) \times P(\mathbf{q}[2]|\mathbf{q}[1]) \times \cdots \times P(\mathbf{q}[m]|\mathbf{q}[m-1],\ldots,\mathbf{q}[2],\mathbf{q}[1])$$

- To apply the chain rule to a conditional probability we just add the conditioning term to each term in the expression:

$$P(\mathbf{q}[1],\ldots,\mathbf{q}[m]|t=l) = P(\mathbf{q}[1]|t=l) \times P(\mathbf{q}[2]|\mathbf{q}[1],t=l) \times \ldots \cdots \times P(\mathbf{q}[m]|\mathbf{q}[m-1],\ldots,\mathbf{q}[3],\mathbf{q}[2],\mathbf{q}[1],t=l)$$

ID	HEADACHE	FEVER	VOMITING	MENINGITIS
1	true	true	false	false
2	false	true	false	false
3	true	false	true	false
4	true	false	true	false
5	false	true	false	true
6	true	false	true	false
7	true	false	true	false
8	true	false	true	true
9	false	true	false	false
10	true	false	true	true

HEADACHE	FEVER	VOMITING	MENINGITIS
true	false	true	?

$$P(M|h, \neg f, v) = ?$$

- In the terms of Bayes' Theorem this problem can be stated as:

$$P(M|h, \neg f, v) = \frac{P(h, \neg f, v|M) \times P(M)}{P(h, \neg f, v)}$$

- There are two values in the domain of the MENINGITIS feature, *'true'* and *'false'*, so we have to do this calculation twice.

- We will do the calculation for m first
- To carry out this calculation we need to know the following probabilities: $P(m)$, $P(h, \neg f, v)$ and $P(h, \neg f, v \mid m)$.

ID	HEADACHE	FEVER	VOMITING	MENINGITIS
1	true	true	false	false
2	false	true	false	false
3	true	false	true	false
4	true	false	true	false
5	false	true	false	true
6	true	false	true	false
7	true	false	true	false
8	true	false	true	true
9	false	true	false	false
10	true	false	true	true

- We can calculate the required probabilities directly from the data. For example, we can calculate $P(m)$ and $P(h, \neg f, v)$ as follows:

$$P(m) = \frac{|\{\mathbf{d}_5, \mathbf{d}_8, \mathbf{d}_{10}\}|}{|\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_7, \mathbf{d}_8, \mathbf{d}_9, \mathbf{d}_{10}\}|} = \frac{3}{10} = 0.3$$

$$P(h, \neg f, v) = \frac{|\{\mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_6, \mathbf{d}_7, \mathbf{d}_8, \mathbf{d}_{10}\}|}{|\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_7, \mathbf{d}_8, \mathbf{d}_9, \mathbf{d}_{10}\}|} = \frac{6}{10} = 0.6$$

- However, as an exercise we will use the chain rule calculate:

$$P(h, \neg f, v \mid m) = ?$$

ID	HEADACHE	FEVER	VOMITING	MENINGITIS
1	true	true	false	false
2	false	true	false	false
3	true	false	true	false
4	true	false	true	false
5	false	true	false	true
6	true	false	true	false
7	true	false	true	false
8	true	false	true	true
9	false	true	false	false
10	true	false	true	true

- Using the chain rule calculate:

$$
\begin{aligned}
P(h, \neg f, v \mid m) &= P(h \mid m) \times P(\neg f \mid h, m) \times P(v \mid \neg f, h, m) \\
&= \frac{|\{\mathbf{d}_8, \mathbf{d}_{10}\}|}{|\{\mathbf{d}_5, \mathbf{d}_8, \mathbf{d}_{10}\}|} \times \frac{|\{\mathbf{d}_8, \mathbf{d}_{10}\}|}{|\{\mathbf{d}_8, \mathbf{d}_{10}\}|} \times \frac{|\{\mathbf{d}_8, \mathbf{d}_{10}\}|}{|\{\mathbf{d}_8, \mathbf{d}_{10}\}|} \\
&= \frac{2}{3} \times \frac{2}{2} \times \frac{2}{2} = 0.6666
\end{aligned}
$$

- So the calculation of $P(m|h, \neg f, v)$ is:

$$P(m|h, \neg f, v) = \frac{\left(\begin{array}{l} P(h|m) \times P(\neg f|h, m) \\ \quad \times P(v|\neg f, h, m) \times P(m) \end{array} \right)}{P(h, \neg f, v)}$$
$$= \frac{0.6666 \times 0.3}{0.6} = 0.3333$$

- The corresponding calculation for $P(\neg m|h,\neg f,v)$ is:

$$
\begin{aligned}
P(\neg m \mid h,\neg f,v) &= \frac{P(h,\neg f,v \mid \neg m)\times P(\neg m)}{P(h,\neg f,v)} \\
&= \frac{\left(\begin{array}{l} P(h|\neg m)\times P(\neg f \mid h,\neg m) \\ \quad \times P(v|\neg f,h,\neg m)\times P(\neg m)\end{array}\right)}{P(h,\neg f,v)} \\
&= \frac{0.7143\times 0.8\times 1.0\times 0.7}{0.6} = 0.6667
\end{aligned}
$$

$$P(m|h, \neg f, v) = 0.3333$$

$$P(\neg m|h, \neg f, v) = 0.6667$$

- These calculations tell us that it is twice as probable that the patient does not have meningitis than it is that they do even though the patient is suffering from a headache and is vomiting!

Bayesian MAP Prediction Model

$$\begin{aligned}\mathbb{M}_{MAP}(\mathbf{q}) &= \underset{l \in levels(t)}{\text{argmax}}\ P(t = l \mid \mathbf{q}[1], \ldots, \mathbf{q}[m]) \\ &= \underset{l \in levels(t)}{\text{argmax}}\ \frac{P(\mathbf{q}[1], \ldots, \mathbf{q}[m] \mid t = l) \times P(t = l)}{P(\mathbf{q}[1], \ldots, \mathbf{q}[m])}\end{aligned}$$

Bayesian MAP Prediction Model (without normalization)

$$\mathbb{M}_{MAP}(\mathbf{q}) = \underset{l \in levels(t)}{\text{argmax}}\ P(\mathbf{q}[1], \ldots, \mathbf{q}[m] \mid t = l) \times P(t = l)$$

ID	HEADACHE	FEVER	VOMITING	MENINGITIS
1	true	true	false	false
2	false	true	false	false
3	true	false	true	false
4	true	false	true	false
5	false	true	false	true
6	true	false	true	false
7	true	false	true	false
8	true	false	true	true
9	false	true	false	false
10	true	false	true	true

HEADACHE	FEVER	VOMITING	MENINGITIS
true	true	false	?

ID	HEADACHE	FEVER	VOMITING	MENINGITIS
1	true	true	false	false
2	false	true	false	false
3	true	false	true	false
4	true	false	true	false
5	false	true	false	true
6	true	false	true	false
7	true	false	true	false
8	true	false	true	true
9	false	true	false	false
10	true	false	true	true

$$P(m \mid h, f, \neg v) = ?$$

$$P(\neg m \mid h, f, \neg v) = ?$$

$$P(m \mid h,f,\neg v) = \frac{\left(\begin{array}{l} P(h|m) \times P(f \mid h,m) \\ \quad \times P(\neg v \mid f,h,m) \times P(m) \end{array} \right)}{P(h,f,\neg v)}$$
$$= \frac{0.6666 \times 0 \times 0 \times 0.3}{0.1} = 0$$

$$P(\neg m \mid h, f, \neg v) = \frac{\left(\begin{array}{l} P(h|\neg m) \times P(f \mid h, \neg m) \\ \quad \times P(\neg v \mid f, h, \neg m) \times P(\neg m) \end{array} \right)}{P(h, f, \neg v)}$$

$$= \frac{0.7143 \times 0.2 \times 1.0 \times 0.7}{0.1} = 1.0$$

$$P(m \mid h, f, \neg v) = 0$$

$$P(\neg m \mid h, f, \neg v) = 1.0$$

- There is something odd about these results!

Curse of Dimensionality

As the number of descriptive features grows the number of potential conditioning events grows. Consequently, an exponential increase is required in the size of the dataset as each new descriptive feature is added to ensure that for any conditional probability there are enough instances in the training dataset matching the conditions so that the resulting probability is reasonable.

- The probability of a patient who has a headache and a fever having meningitis should be greater than zero!
- Our dataset is not large enough $\rightarrow$ our model is over-fitting to the training data.
- The concepts of conditional independence and factorization can help us overcome this flaw of our current approach.

- If knowledge of one event has no effect on the probability of another event, and *vice versa*, then the two events are **independent** of each other.
- If two events X and Y are independent then:

$$P(X|Y) = P(X)$$
$$P(X,Y) = P(X) \times P(Y)$$

- Recall, that when two event are dependent these rules are:

$$P(X|Y) = \frac{P(X,Y)}{P(Y)}$$
$$P(X,Y) = P(X|Y) \times P(Y) = P(Y|X) \times P(X)$$

- Full independence between events is quite rare.
- A more common phenomenon is that two, or more, events may be independent if we know that a third event has happened.
- This is known as conditional independence.

- For two events, X and Y, that are conditionally independent given knowledge of a third events, here Z, the definition of the probability of a joint event and conditional probability are:

$$P(X|Y,Z) = P(X|Z)$$
$$P(X,Y|Z) = P(X|Z) \times P(Y|Z)$$

$$P(X|Y) = \frac{P(X,Y)}{P(Y)}$$
$$P(X,Y) = P(X|Y) \times P(Y)$$
$$= P(Y|X) \times P(X)$$

X and Y are dependent

$$P(X|Y) = P(X)$$
$$P(X,Y) = P(X) \times P(Y)$$

X and Y are independent

- If the event $t = l$ causes the events $\mathbf{q}[1], \ldots, \mathbf{q}[m]$ to happen then the events $\mathbf{q}[1], \ldots, \mathbf{q}[m]$ are conditionally independent of each other given knowledge of $t = l$ and the chain rule definition can be simplified as follows:

$$
\begin{aligned}
&P(\mathbf{q}[1], \ldots, \mathbf{q}[m] \mid t = l) \\
&\qquad = P(\mathbf{q}[1] \mid t = l) \times P(\mathbf{q}[2] \mid t = l) \times \cdots \times P(\mathbf{q}[m] \mid t = l) \\
&\qquad = \prod_{i=1}^{m} P(\mathbf{q}[i] \mid t = l)
\end{aligned}
$$

- Using this we can simplify the calculations in Bayes' Theorem, under the assumption of conditional independence between the descriptive features given the level l of the target feature:

$$P(t = l \mid \mathbf{q}[1], \ldots, \mathbf{q}[m]) = \frac{\left(\prod_{i=1}^{m} P(\mathbf{q}[i] \mid t = l) \right) \times P(t = l)}{P(\mathbf{q}[1], \ldots, \mathbf{q}[m])}$$

Without conditional independence

$$P(X, Y, Z|W) = P(X|W) \times P(Y|X, W) \times P(Z|Y, X, W) \times P(W)$$

With conditional independence

$$P(X, Y, Z|W) = \underbrace{P(X|W)}_{Factor1} \times \underbrace{P(Y|W)}_{Factor2} \times \underbrace{P(Z|W)}_{Factor3} \times \underbrace{P(W)}_{Factor4}$$

- The joint probability distribution for the meningitis dataset.

$$\mathbf{P}(H,F,V,M) = \begin{bmatrix} P(h,f,v,m), & P(\neg h,f,v,m) \\ P(h,f,v,\neg m), & P(\neg h,f,v,\neg m) \\ P(h,f,\neg v,m), & P(\neg h,f,\neg v,m) \\ P(h,f,\neg v,\neg m), & P(\neg h,f,\neg v,\neg m) \\ P(h,\neg f,v,m), & P(\neg h,\neg f,v,m) \\ P(h,\neg f,v,\neg m), & P(\neg h,\neg f,v,\neg m) \\ P(h,\neg f,\neg v,m), & P(\neg h,\neg f,\neg v,m) \\ P(h,\neg f,\neg v,\neg m), & P(\neg h,\neg f,\neg v,\neg m) \end{bmatrix}$$

- Assuming the descriptive features are conditionally independent of each other given MENINGITIS we only need to store four factors:

$$Factor_1 : < P(M) >$$
$$Factor_2 : < P(h|m), P(h|\neg m) >$$
$$Factor_3 : < P(f|m), P(f|\neg m) >$$
$$Factor_4 : < P(v|m), P(v|\neg m) >$$

$$P(H, F, V, M) = P(M) \times P(H|M) \times P(F|M) \times P(V|M)$$

ID	HEADACHE	FEVER	VOMITING	MENINGITIS
1	true	true	false	false
2	false	true	false	false
3	true	false	true	false
4	true	false	true	false
5	false	true	false	true
6	true	false	true	false
7	true	false	true	false
8	true	false	true	true
9	false	true	false	false
10	true	false	true	true

- Calculate the factors from the data.

$$Factor_1 : < P(M) >$$
$$Factor_2 : < P(h|m), P(h|\neg m) >$$
$$Factor_3 : < P(f|m), P(f|\neg m) >$$
$$Factor_4 : < P(v|m), P(v|\neg m) >$$

$$Factor_1 : < P(m) = 0.3 >$$
$$Factor_2 : < P(h|m) = 0.6666, P(h|\neg m) = 0.7413 >$$
$$Factor_3 : < P(f|m) = 0.3333, P(f|\neg m) = 0.4286 >$$
$$Factor_4 : < P(v|m) = 0.6666, P(v|\neg m) = 0.5714 >$$

$$Factor_1 : < P(m) = 0.3 >$$
$$Factor_2 : < P(h|m) = 0.6666, P(h|\neg m) = 0.7413 >$$
$$Factor_3 : < P(f|m) = 0.3333, P(f|\neg m) = 0.4286 >$$
$$Factor_4 : < P(v|m) = 0.6666, P(v|\neg m) = 0.5714 >$$

- Using the factors above calculate the probability of MENINGITIS=*'true'* for the following query.

HEADACHE	FEVER	VOMITING	MENINGITIS
true	true	false	?

$$P(m|h,f,\neg v) = \frac{P(h|m) \times P(f|m) \times P(\neg v|m) \times P(m)}{\sum_i P(h|M_i) \times P(f|M_i) \times P(\neg v|M_i) \times P(M_i)} =$$

$$\frac{0.6666 \times 0.3333 \times 0.3333 \times 0.3}{(0.6666 \times 0.3333 \times 0.3333 \times 0.3) + (0.7143 \times 0.4286 \times 0.4286 \times 0.7)} = 0.1948$$

$$Factor_1 : < P(m) = 0.3 >$$
$$Factor_2 : < P(h|m) = 0.6666, P(h|\neg m) = 0.7413 >$$
$$Factor_3 : < P(f|m) = 0.3333, P(f|\neg m) = 0.4286 >$$
$$Factor_4 : < P(v|m) = 0.6666, P(v|\neg m) = 0.5714 >$$

- Using the factors above calculate the probability of MENINGITIS=*'false'* for the same query.

HEADACHE	FEVER	VOMITING	MENINGITIS
true	true	false	?

$$P(\neg m|h,f,\neg v) = \frac{P(h|\neg m) \times P(f|\neg m) \times P(\neg v|\neg m) \times P(\neg m)}{\sum_i P(h|M_i) \times P(f|M_i) \times P(\neg v|M_i) \times P(M_i)} =$$

$$\frac{0.7143 \times 0.4286 \times 0.4286 \times 0.7}{(0.6666 \times 0.3333 \times 0.3333 \times 0.3) + (0.7143 \times 0.4286 \times 0.4286 \times 0.7)} = 0.8052$$

$$P(m|h,f,\neg v) = 0.1948$$

$$P(\neg m|h,f,\neg v) = 0.8052$$

- As before, the MAP prediction would be MENINGITIS = *'false'*
- The posterior probabilities are not as extreme!

Naive Bayes' Classifier

$$\mathbb{M}(\mathbf{q}) = \underset{l \in levels(t)}{\operatorname{argmax}} \left(\prod_{i=1}^{m} P(\mathbf{q}[i] \mid t = l) \right) \times P(t = l)$$

Naive Bayes' is simple to train!

1. Calculate the priors for each of the target levels
2. Calculate the conditional probabilities for each feature given each target level.

Table 3: A dataset from a loan application fraud detection domain.

ID	CREDIT HISTORY	GUARANTOR/ COAPPLICANT	ACCOMODATION	FRAUD
1	current	none	own	true
2	paid	none	own	false
3	paid	none	own	false
4	paid	guarantor	rent	true
5	arrears	none	own	false
6	arrears	none	own	true
7	current	none	own	false
8	arrears	none	own	false
9	current	none	rent	false
10	none	none	own	true
11	current	coapplicant	own	false
12	current	none	own	true
13	current	none	rent	true
14	paid	none	own	false
15	arrears	none	own	false
16	current	none	own	false
17	arrears	coapplicant	rent	false
18	arrears	none	free	false
19	arrears	none	own	false
20	paid	none	own	false

$P(fr)$	$=$	0.3	$P(\neg fr)$	$=$	0.7
$P(\text{CH} = \textit{'none'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'none'} \mid \neg fr)$	$=$	0
$P(\text{CH} = \textit{'paid'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'paid'} \mid \neg fr)$	$=$	0.2857
$P(\text{CH} = \textit{'current'} \mid fr)$	$=$	0.5	$P(\text{CH} = \textit{'current'} \mid \neg fr)$	$=$	0.2857
$P(\text{CH} = \textit{'arrears'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'arrears'} \mid \neg fr)$	$=$	0.4286
$P(\text{GC} = \textit{'none'} \mid fr)$	$=$	0.8334	$P(\text{GC} = \textit{'none'} \mid \neg fr)$	$=$	0.8571
$P(\text{GC} = \textit{'guarantor'} \mid fr)$	$=$	0.1666	$P(\text{GC} = \textit{'guarantor'} \mid \neg fr)$	$=$	0
$P(\text{GC} = \textit{'coapplicant'} \mid fr)$	$=$	0	$P(\text{GC} = \textit{'coapplicant'} \mid \neg fr)$	$=$	0.1429
$P(\text{ACC} = \textit{'own'} \mid fr)$	$=$	0.6666	$P(\text{ACC} = \textit{'own'} \mid \neg fr)$	$=$	0.7857
$P(\text{ACC} = \textit{'rent'} \mid fr)$	$=$	0.3333	$P(\text{ACC} = \textit{'rent'} \mid \neg fr)$	$=$	0.1429
$P(\text{ACC} = \textit{'free'} \mid fr)$	$=$	0	$P(\text{ACC} = \textit{'free'} \mid \neg fr)$	$=$	0.0714

Table 4: The probabilities needed by a Naive Bayes prediction model calculated from the dataset. Notation key: FR=FRAUDULENT, CH=CREDIT HISTORY, GC = GUARANTOR/COAPPLICANT, ACC = ACCOMODATION, T=*'true'*, F=*'false'*.

$P(fr)$	$=$	0.3	$P(\neg fr)$	$=$	0.7
$P(\text{CH} = \textit{'none'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'none'} \mid \neg fr)$	$=$	0
$P(\text{CH} = \textit{'paid'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'paid'} \mid \neg fr)$	$=$	0.2857
$P(\text{CH} = \textit{'current'} \mid fr)$	$=$	0.5	$P(\text{CH} = \textit{'current'} \mid \neg fr)$	$=$	0.2857
$P(\text{CH} = \textit{'arrears'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'arrears'} \mid \neg fr)$	$=$	0.4286
$P(\text{GC} = \textit{'none'} \mid fr)$	$=$	0.8334	$P(\text{GC} = \textit{'none'} \mid \neg fr)$	$=$	0.8571
$P(\text{GC} = \textit{'guarantor'} \mid fr)$	$=$	0.1666	$P(\text{GC} = \textit{'guarantor'} \mid \neg fr)$	$=$	0
$P(\text{GC} = \textit{'coapplicant'} \mid fr)$	$=$	0	$P(\text{GC} = \textit{'coapplicant'} \mid \neg fr)$	$=$	0.1429
$P(\text{ACC} = \textit{'own'} \mid fr)$	$=$	0.6666	$P(\text{ACC} = \textit{'own'} \mid \neg fr)$	$=$	0.7857
$P(\text{ACC} = \textit{'rent'} \mid fr)$	$=$	0.3333	$P(\text{ACC} = \textit{'rent'} \mid \neg fr)$	$=$	0.1429
$P(\text{ACC} = \textit{'free'} \mid fr)$	$=$	0	$P(\text{ACC} = \textit{'free'} \mid \neg fr)$	$=$	0.0714

CREDIT HISTORY	GUARANTOR/COAPPLICANT	ACCOMODATION	FRAUDULENT
paid	none	rent	?

$P(fr)$	$=$	0.3	$P(\neg fr)$	$=$	0.7
$P(\text{CH} = \textit{'paid'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'paid'} \mid \neg fr)$	$=$	0.2857
$P(\text{GC} = \textit{'none'} \mid fr)$	$=$	0.8334	$P(\text{GC} = \textit{'none'} \mid \neg fr)$	$=$	0.8571
$P(\text{ACC} = \textit{'rent'} \mid fr)$	$=$	0.3333	$P(\text{ACC} = \textit{'rent'} \mid \neg fr)$	$=$	0.1429

$$\left(\prod_{k=1}^{m} P\left(\mathbf{q}\left[k\right] \mid fr\right) \right) \times P\left(fr\right) = 0.0139$$

$$\left(\prod_{k=1}^{m} P\left(\mathbf{q}\left[k\right] \mid \neg fr\right) \right) \times P(\neg fr) = 0.0245$$

CREDIT HISTORY	GUARANTOR/COAPPLICANT	ACCOMODATION	FRAUDULENT
paid	none	rent	?

$P(fr)$	$=$	0.3	$P(\neg fr)$	$=$	0.7
$P(\text{CH} = \textit{'paid'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \textit{'paid'} \mid \neg fr)$	$=$	0.2857
$P(\text{GC} = \textit{'none'} \mid fr)$	$=$	0.8334	$P(\text{GC} = \textit{'none'} \mid \neg fr)$	$=$	0.8571
$P(\text{ACC} = \textit{'rent'} \mid fr)$	$=$	0.3333	$P(\text{ACC} = \textit{'rent'} \mid \neg fr)$	$=$	0.1429

$$\left(\prod_{k=1}^{m} P\left(\mathbf{q}\left[k\right] \mid fr\right) \right) \times P\left(fr\right) = 0.0139$$

$$\left(\prod_{k=1}^{m} P\left(\mathbf{q}\left[k\right] \mid \neg fr\right) \right) \times P(\neg fr) = 0.0245$$

CREDIT HISTORY	GUARANTOR/COAPPLICANT	ACCOMODATION	FRAUDULENT
paid	none	rent	*'false'*

The model is generalizing beyond the dataset!

ID	Credit History	Guarantor/ CoApplicant	Accommodation	Fraud
1	current	none	own	true
2	paid	none	own	false
3	paid	none	own	false
4	paid	guarantor	rent	true
5	arrears	none	own	false
6	arrears	none	own	true
7	current	none	own	false
8	arrears	none	own	false
9	current	none	rent	false
10	none	none	own	true
11	current	coapplicant	own	false
12	current	none	own	true
13	current	none	rent	true
14	paid	none	own	false
15	arrears	none	own	false
16	current	none	own	false
17	arrears	coapplicant	rent	false
18	arrears	none	free	false
19	arrears	none	own	false
20	paid	none	own	false

Credit History	Guarantor/CoApplicant	Accommodation	Fraudulent
paid	none	rent	*'false'*

- The year is 1798 and you are Lieutenant-Colonel David Collins of HMS Calcutta who is exploring the region around Hawkesbury River, in New South Wales.
- After an expedition up the river one of the men tells you that he saw a strange animal near the river.
- You ask him to describe the animal to you and he explains that he didn't see it very well but that he did notice that it had webbed feet and a duck-bill snout, and that it growled at him.
- In order to plan the expedition for the next day you decide that you need to classify the animal so that you can figure out whether it is dangerous to approach it or not.

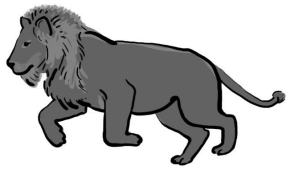
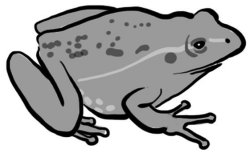
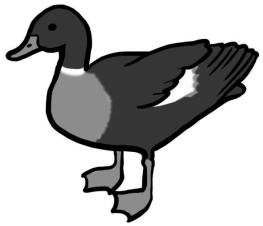
	Grrrh!			**Score**
	✓	✗	✗	1
	✗	✓	✗	1
	✗	✓	✓	2

Figure 1: Matching animals you remember to the features of the unknown animal described by the sailor. Note: The images used in this figure were created by Jan Gillbank for the English for the Australian Curriculum website (`http://www.e4ac.edu.au`) and are used under the Create Commons Attribution 3.0 Unported licence (`http://creativecommons.org/licenses/by/3.0`). The images were sourced via Wikimedia Commons.

- The process of classifying an unknown animal by matching the features of the animal against the features of animals you can remember neatly encapsulates the big idea underpinning similarity-based learning:

if you are trying to classify something then you should search your memory to find things that are similar and label it with the same class as the most similar thing in your memory

- One of the simplest and best known machine learning algorithms for this type of reasoning is called the nearest neighbor algorithm.

- The fundamentals of similarity-based learning are:
  - Feature space
  - Similarity metrics

Table 5: The speed and agility ratings for 20 college athletes labelled with the decisions for whether they were drafted or not.

ID	Speed	Agility	Draft
1	2.50	6.00	No
2	3.75	8.00	No
3	2.25	5.50	No
4	3.25	8.25	No
5	2.75	7.50	No
6	4.50	5.00	No
7	3.50	5.25	No
8	3.00	3.25	No
9	4.00	4.00	No
10	4.25	3.75	No

ID	Speed	Agility	Draft
11	2.00	2.00	No
12	5.00	2.50	No
13	8.25	8.50	No
14	5.75	8.75	Yes
15	4.75	6.25	Yes
16	5.50	6.75	Yes
17	5.25	9.50	Yes
18	7.00	4.25	Yes
19	7.50	8.00	Yes
20	7.25	5.75	Yes

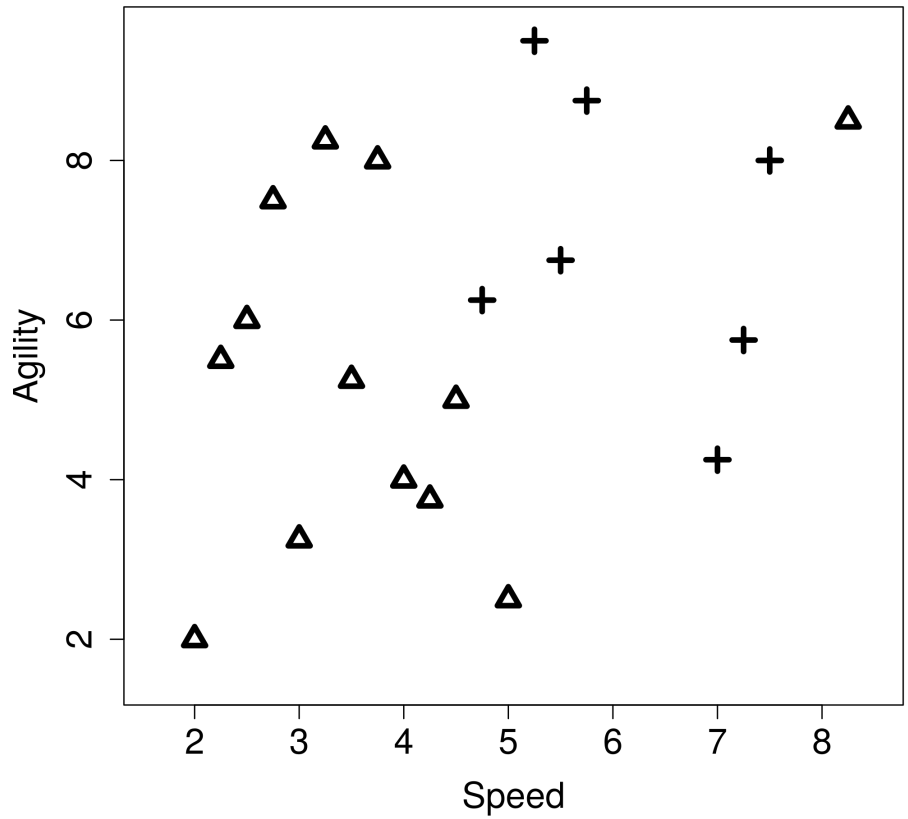


Figure 2: A feature space plot of the data in Table 5 [62]. The triangles represent *'Non-draft'* instances and the crosses represent the *'Draft'* instances.

- A feature space is an abstract n-dimensional space that is created by taking each of the descriptive features in the dataset to be the axes of a reference space and each instance in the dataset is mapped to a point in the feature space based on the values of its descriptive features.

- A similarity metric measures the similarity between two instances according to a feature space
- Mathematically, a metric must conform to the following four criteria:
  1. Non-negativity: $metric(\mathbf{a}, \mathbf{b}) \geq 0$
  2. Identity: $metric(\mathbf{a}, \mathbf{b}) = 0 \iff \mathbf{a} = \mathbf{b}$
  3. Symmetry: $metric(\mathbf{a}, \mathbf{b}) = metric(\mathbf{b}, \mathbf{a})$
  4. Triangular Inequality: $metric(\mathbf{a}, \mathbf{b}) \leq metric(\mathbf{a}, \mathbf{c}) + metric(\mathbf{b}, \mathbf{c})$

  where $metric(\mathbf{a}, \mathbf{b})$ is a function that returns the distance between two instances **a** and **b**.

- One of the best known metrics is Euclidean distance which computes the length of the straight line between two points. Euclidean distance between two instances **a** and **b** in a m-dimensional feature space is defined as:

$$Euclidean(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{m} (\mathbf{a}[i] - \mathbf{b}[i])^2} \tag{2}$$

Example 2

The Euclidean distance between instances d_{12} (SPEED$= 5.00$, AGILITY$= 2.5$) and d_5 (SPEED$= 2.75$,AGILITY$= 7.5$) in Table 5 [62] is:

Example 2

The Euclidean distance between instances d_{12} (SPEED$=5.00$, AGILITY$=2.5$) and d_5 (SPEED$=2.75$,AGILITY$=7.5$) in Table 5 [62] is:

$$\begin{aligned} Euclidean(\langle 5.00, 2.50\rangle, \langle 2.75, 7.50\rangle) &= \sqrt{(5.00-2.75)^2+(2.50-7.50)^2} \\ &= \sqrt{30.0625} = 5.4829 \end{aligned}$$

- Another, less well known, distance measure is the Manhattan distance or taxi-cab distance.
- The Manhattan distance between two instances **a** and **b** in a feature space with m dimensions is:[2]

$$Manhattan(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{m} abs(\mathbf{a}[i] - \mathbf{b}[i]) \tag{3}$$

[2] The $abs()$ function surrounding the subtraction term indicates that we use the absolute value, i.e. non-negative value, when we are summing the differences; this makes sense because distances can't be negative.

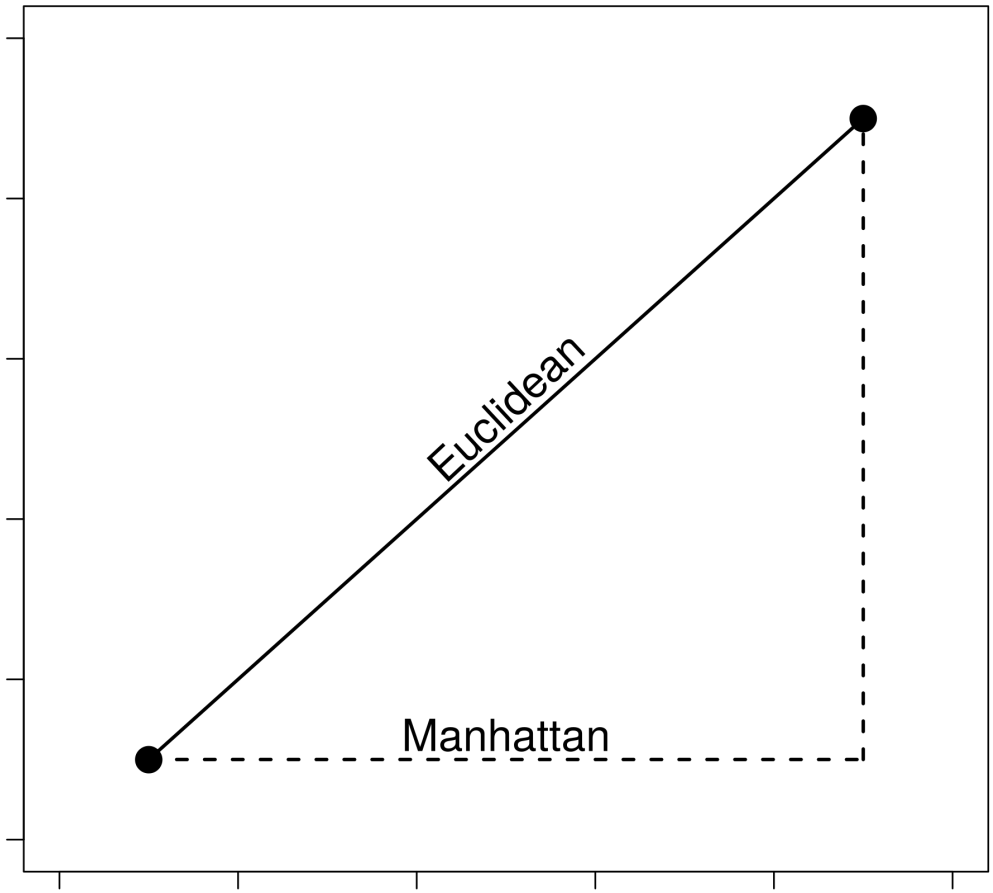


Figure 3: The Manhattan and Euclidean distances between two points.

Example 3

The Manhattan distance between instances d_{12} (SPEED$= 5.00$, AGILITY$= 2.5$) and d_5 (SPEED$= 2.75$,AGILITY$= 7.5$) in Table 5 [62] is:

Example 3

The Manhattan distance between instances d_{12} (SPEED$=5.00$, AGILITY$=2.5$) and d_5 (SPEED$=2.75$,AGILITY$=7.5$) in Table 5 [62] is:

$$\begin{aligned} Manhattan(\langle 5.00, 2.50\rangle, \langle 2.75, 7.50\rangle) &= abs(5.00 - 2.75) + abs(2.5 - 7.5) \\ &= 2.25 + 5 = 7.25 \end{aligned}$$

- The Euclidean and Manhattan distances are special cases of Minkowski distance
- The Minkowski distance between two instances **a** and **b** in a feature space with m descriptive features is:

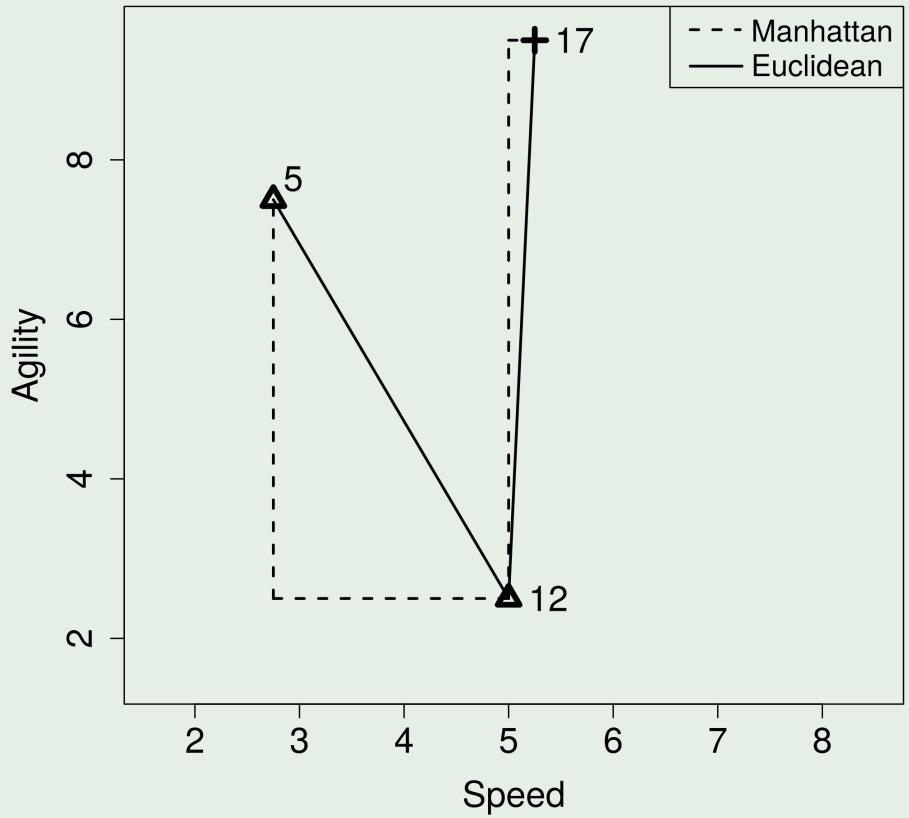
$$Minkowski(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{m} abs(\mathbf{a}[i] - \mathbf{b}[i])^p \right)^{\frac{1}{p}} \quad (4)$$

  where different values of the parameter p result in different distance metrics
- The Minkowski distance with $p = 1$ is the Manhattan distance and with $p = 2$ is the Euclidean distance.

- The larger the value of p the more emphasis is placed on the features with large differences in values because these differences are raised to the power of p.

Example 4

Instance ID	Instance ID	Manhattan (Minkowski p=1)	Euclidean (Minkowski p=2)
12	5	7.25	5.4829
12	17	7.25	8.25



The Manhattan and Euclidean distances between instances $\mathbf{d}_{12}$ (SPEED= 5.00, AGILITY= 2.5) and $\mathbf{d}_5$ (SPEED= 2.75, AGILITY= 7.5) and between instances $\mathbf{d}_{12}$ and $\mathbf{d}_{17}$ (SPEED= 5.25, AGILITY= 9.5).

The Nearest Neighbour Algorithm

Require: set of training instances
Require: a query to be classified

1: Iterate across the instances in memory and find the instance that is shortest distance from the query position in the feature space.
2: Make a prediction for the query equal to the value of the target feature of the nearest neighbor.

Table 6: The speed and agility ratings for 20 college athletes labelled with the decisions for whether they were drafted or not.

ID	Speed	Agility	Draft
1	2.50	6.00	No
2	3.75	8.00	No
3	2.25	5.50	No
4	3.25	8.25	No
5	2.75	7.50	No
6	4.50	5.00	No
7	3.50	5.25	No
8	3.00	3.25	No
9	4.00	4.00	No
10	4.25	3.75	No

ID	Speed	Agility	Draft
11	2.00	2.00	No
12	5.00	2.50	No
13	8.25	8.50	No
14	5.75	8.75	Yes
15	4.75	6.25	Yes
16	5.50	6.75	Yes
17	5.25	9.50	Yes
18	7.00	4.25	Yes
19	7.50	8.00	Yes
20	7.25	5.75	Yes

Example 5

- Should we draft an athlete with the following profile:

$$\text{SPEED} = 6.75,\ \text{AGILITY} = 3$$

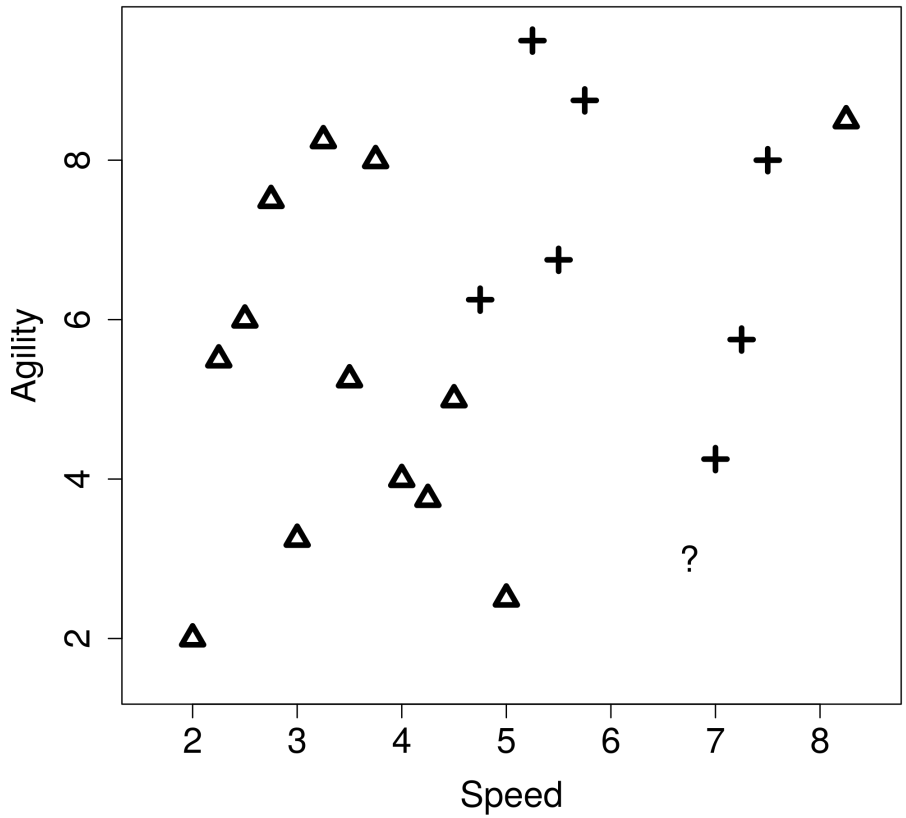
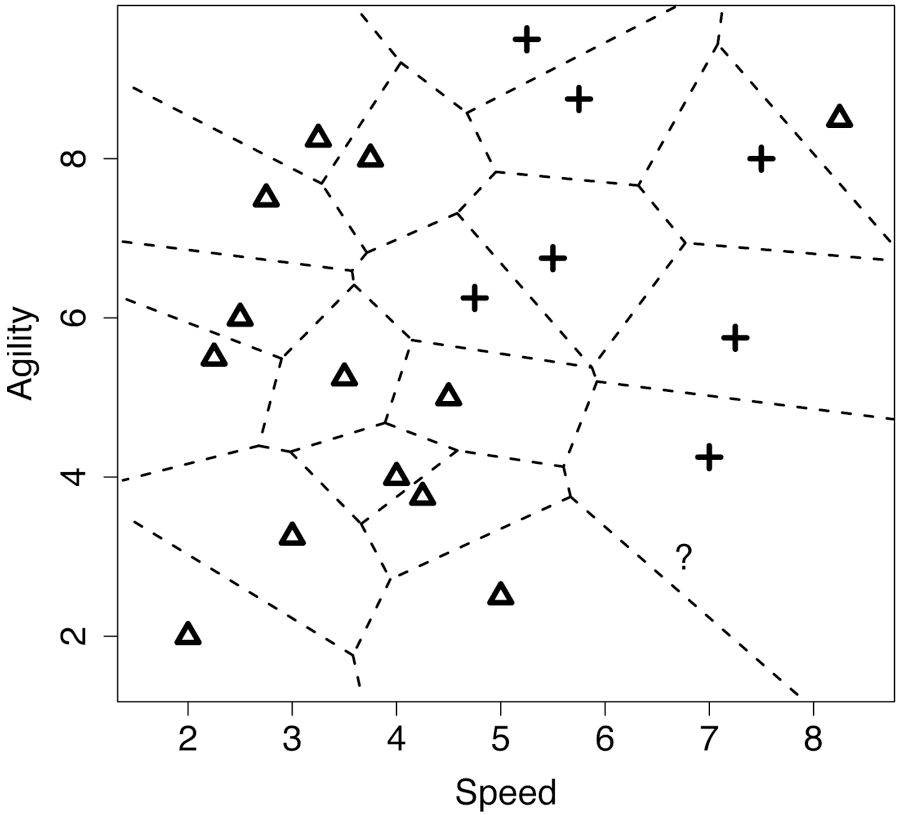


Figure 4: A feature space plot of the data in Table 6 [77] with the position in the feature space of the query represented by the ? marker. The triangles represent *'Non-draft'* instances and the crosses represent the *'Draft'* instances.

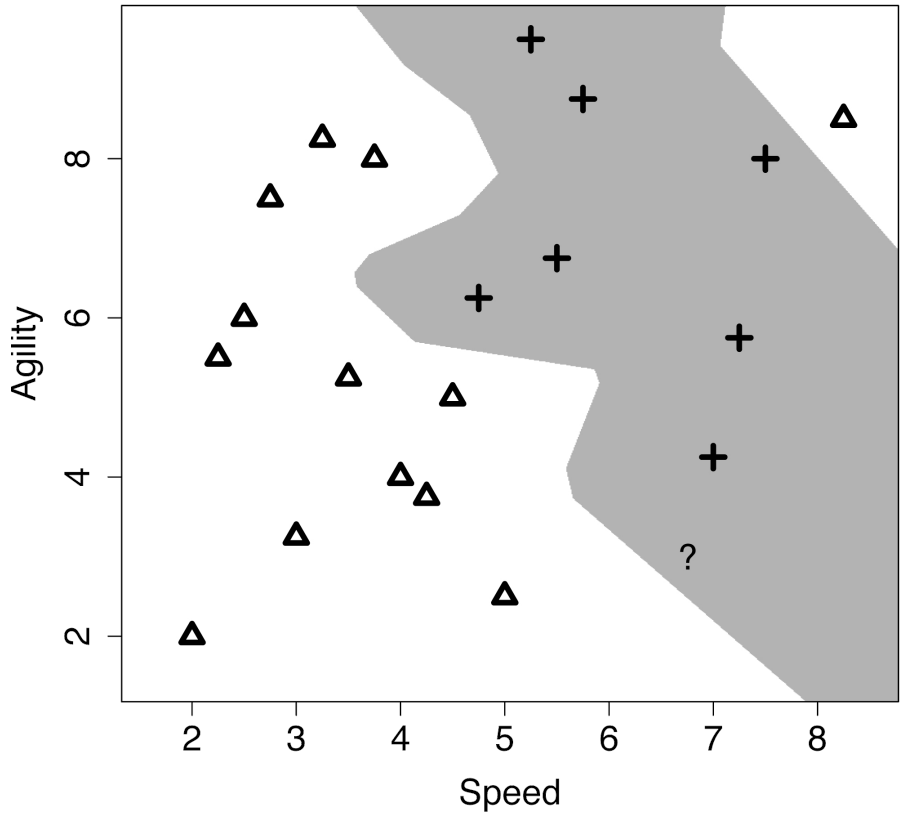
Table 7: The distances (Dist.) between the query instance with SPEED $= 6.75$ and AGILITY $= 3.00$ and each instance in Table 6 [77].

ID	SPEED	AGILITY	DRAFT	Dist.
18	7.00	4.25	yes	1.27
12	5.00	2.50	no	1.82
10	4.25	3.75	no	2.61
20	7.25	5.75	yes	2.80
9	4.00	4.00	no	2.93
6	4.50	5.00	no	3.01
8	3.00	3.25	no	3.76
15	4.75	6.25	yes	3.82
7	3.50	5.25	no	3.95
16	5.50	6.75	yes	3.95

ID	SPEED	AGILITY	DRAFT	Dist.
11	2.00	2.00	no	4.85
19	7.50	8.00	yes	5.06
3	2.25	5.50	no	5.15
1	2.50	6.00	no	5.20
13	8.25	8.50	no	5.70
2	3.75	8.00	no	5.83
14	5.75	8.75	yes	5.84
5	2.75	7.50	no	6.02
4	3.25	8.25	no	6.31
17	5.25	9.50	yes	6.67



(a) Voronoi tessellation



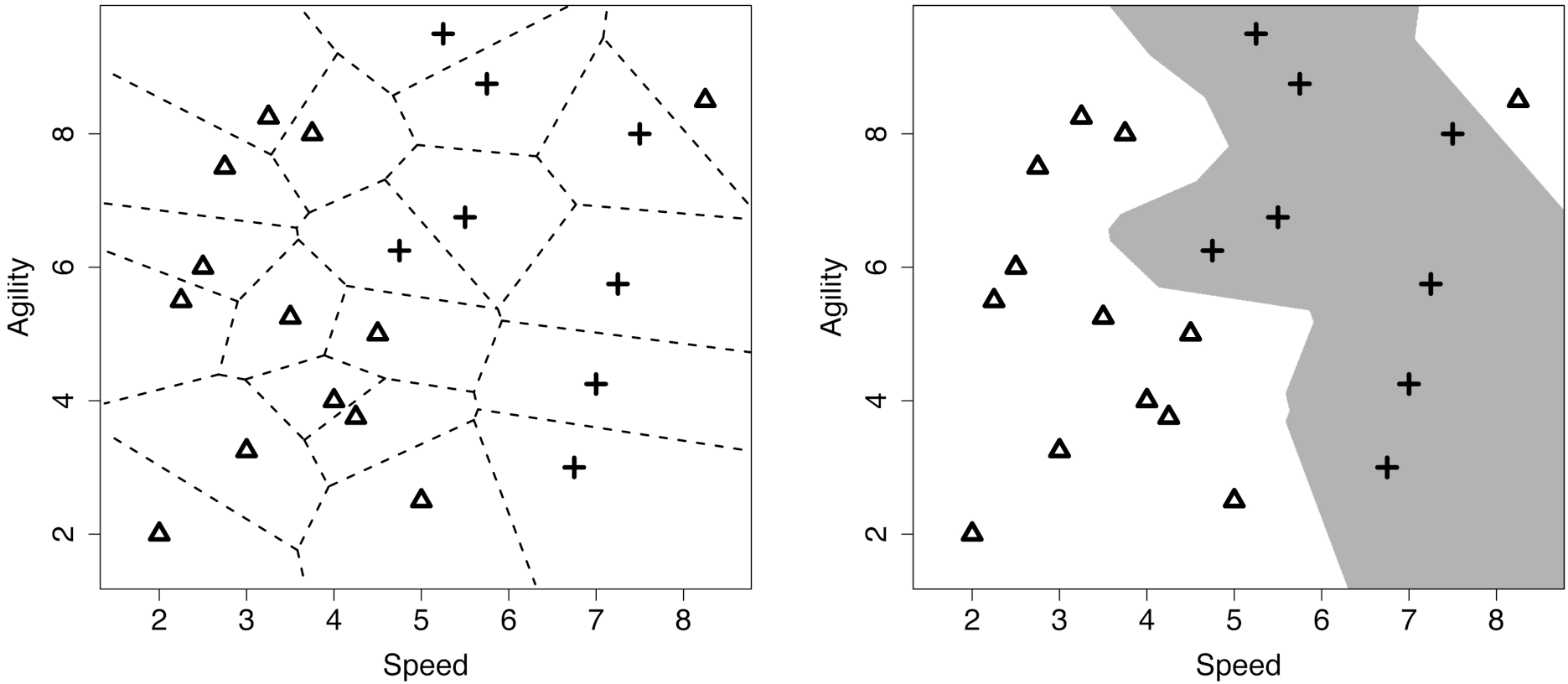
(b) Decision boundary ($k = 1$)

Figure 5: (a) The Voronoi tessellation of the feature space for the dataset in Table 6 [77] with the position of the query represented by the ? marker; (b) the decision boundary created by aggregating the neighboring Voronoi regions that belong to the same target level.

- One of the great things about nearest neighbour algorithms is that we can add in new data to update the model very easily.

Table 8: The extended version of the college athletes dataset.

ID	SPEED	AGILITY	DRAFT
1	2.50	6.00	no
2	3.75	8.00	no
3	2.25	5.50	no
4	3.25	8.25	no
5	2.75	7.50	no
6	4.50	5.00	no
7	3.50	5.25	no
8	3.00	3.25	no
9	4.00	4.00	no
10	4.25	3.75	no
11	2.00	2.00	no
12	5.00	2.50	no
13	8.25	8.50	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.50	6.75	yes
17	5.25	9.50	yes
18	7.00	4.25	yes
19	7.50	8.00	yes
20	7.25	5.75	yes
21	6.75	3.00	yes

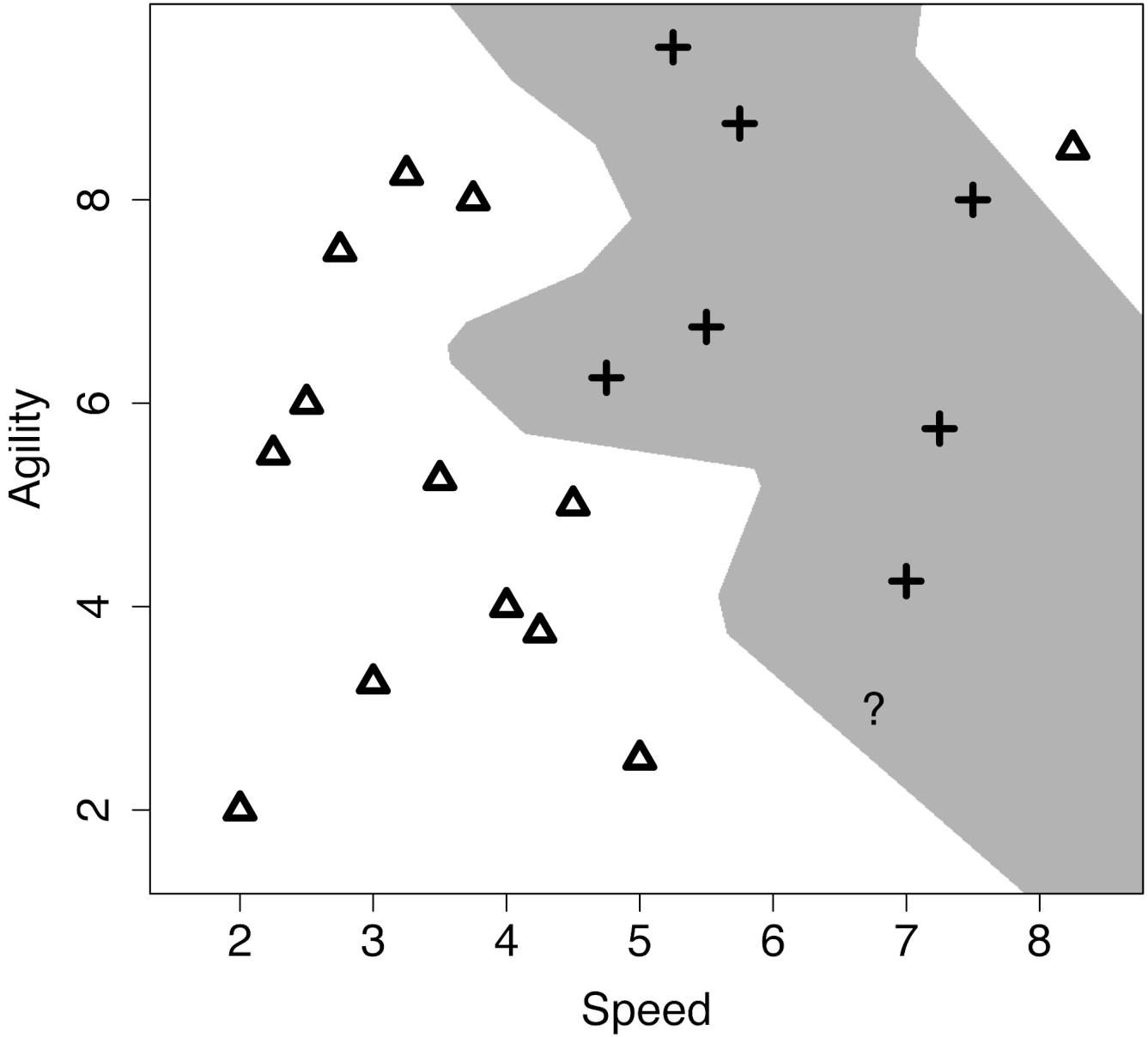


(a) Voronoi tessellation

(b) Decision boundary ($k = 1$)

Figure 6: (a) The Voronoi tessellation of the feature space when the dataset has been updated to include the query instance; (b) the updated decision boundary reflecting the addition of the query instance in the training set.

Handling Noisy Data



- The k nearest neighbors model predicts the target level with the majority vote from the set of k nearest neightbors to the query $\mathbf{q}$:

$$\mathbb{M}_k(\mathbf{q}) = \underset{l \in levels(t)}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(t_i, l) \tag{5}$$

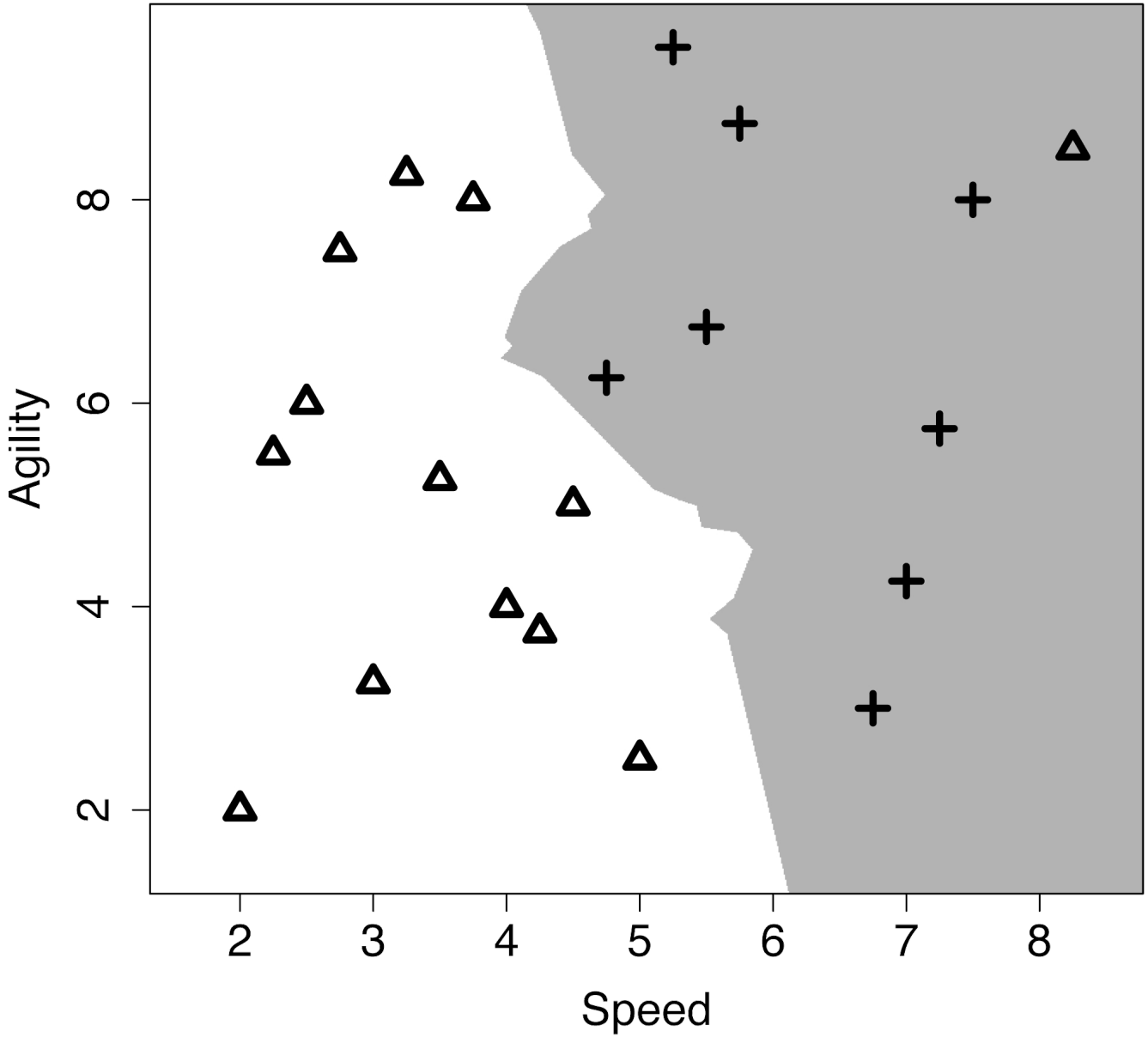


Figure 8: The decision boundary using majority classification of the nearest 3 neighbors.

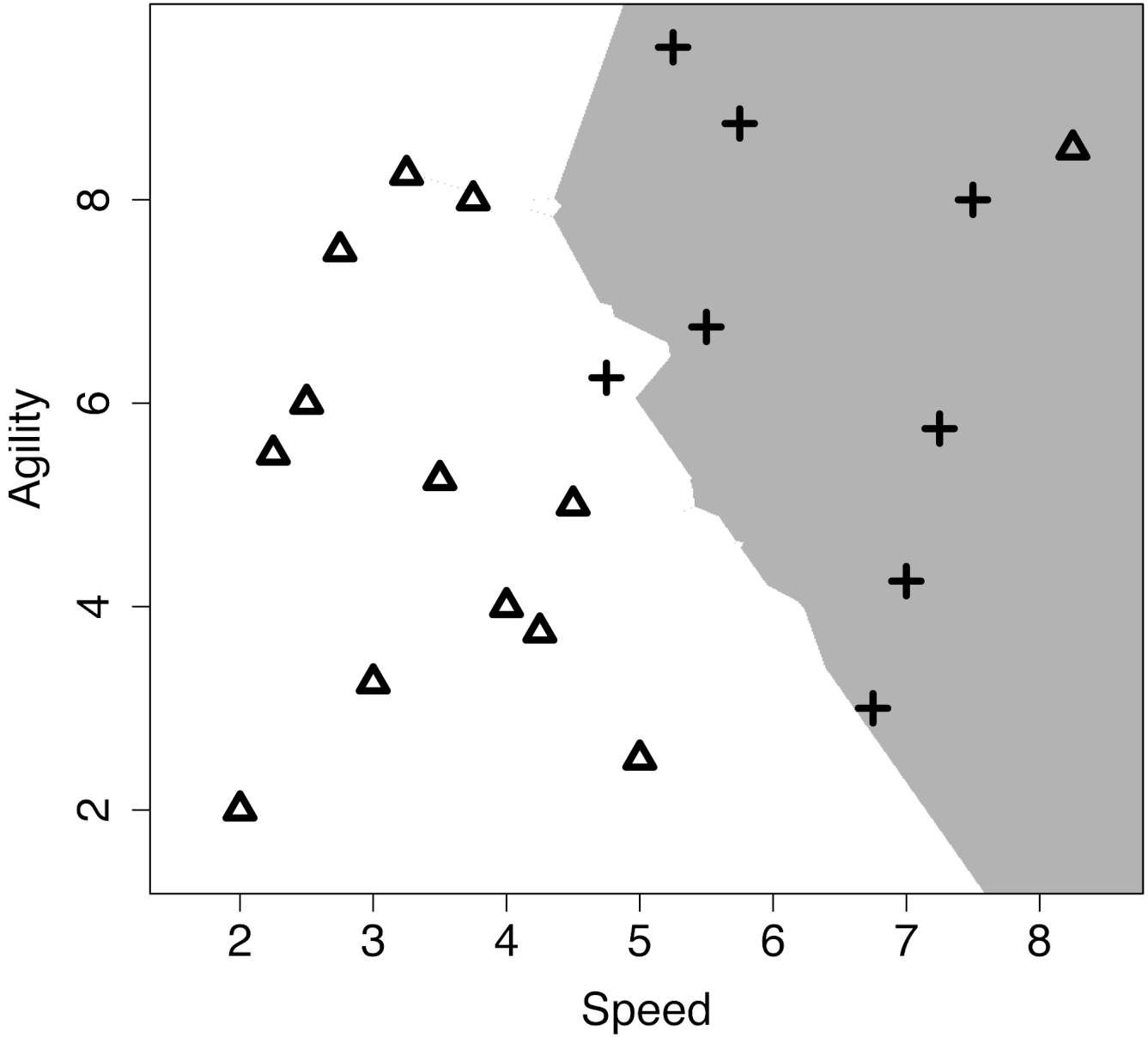


Figure 9: The decision boundary using majority classification of the nearest 5 neighbors.

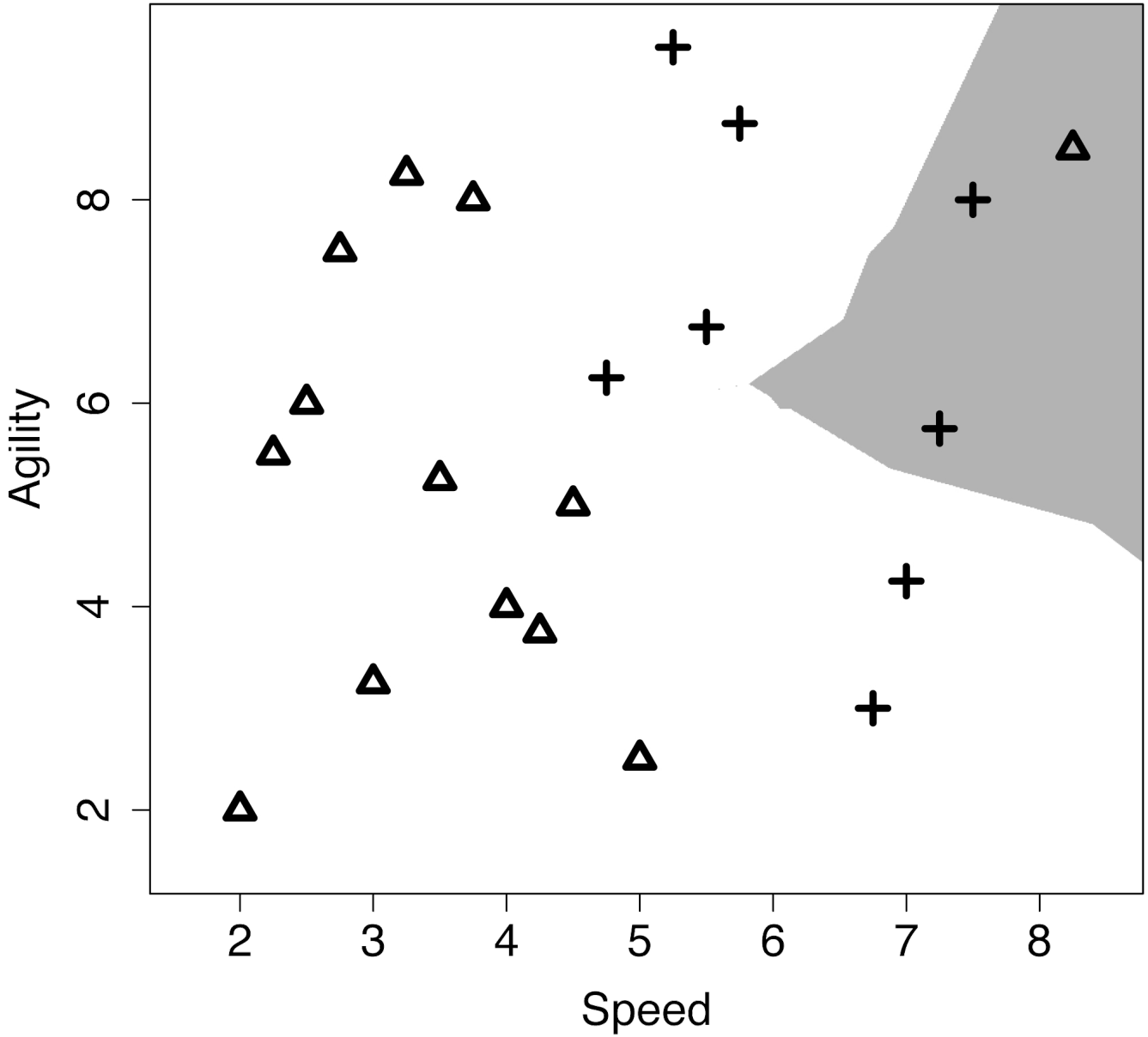


Figure 10: The decision boundary when k is set to 15.

- In a distance weighted *k* nearest neighbor algorithm the contribution of each neighbor to the classification decision is weighted by the reciprocal of the squared distance between the neighbor **d** and the query **q**:

$$\frac{1}{dist(\mathbf{q},\mathbf{d})^2} \tag{6}$$

- The weighted *k* nearest neighbor model is defined as:

$$\mathbb{M}_k(\mathbf{q}) = \underset{l \in levels(t)}{\text{argmax}} \sum_{i=1}^{k} \frac{1}{dist(\mathbf{q},\mathbf{d_i})^2} \times \delta(t_i, l) \tag{7}$$

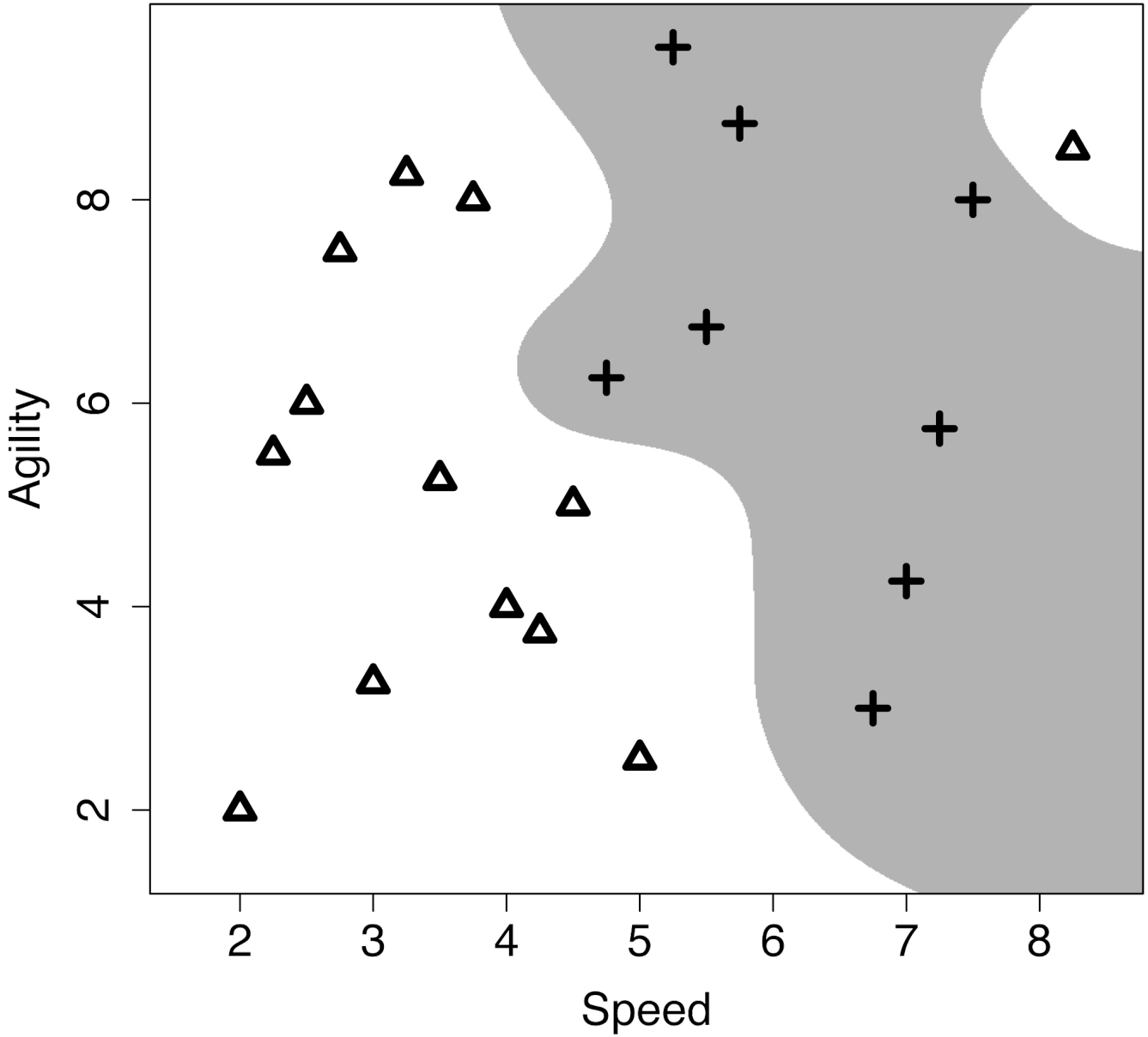


Figure 11: The weighted *k* nearest neighbor model decision boundary.

Data Normalization

Table 9: A dataset listing the salary and age information for customers and whether or not the purchased a pension plan .

ID	Salary	Age	Purchased
1	53700	41	No
2	65300	37	No
3	48900	45	Yes
4	64800	49	Yes
5	44200	30	No
6	55900	57	Yes
7	48600	26	No
8	72800	60	Yes
9	45300	34	No
10	73200	52	Yes

Data Normalization

- The marketing department wants to decide whether or not they should contact a customer with the following profile:

$$\langle \text{SALARY} = 56,000, \text{AGE} = 35 \rangle$$

Data Normalization

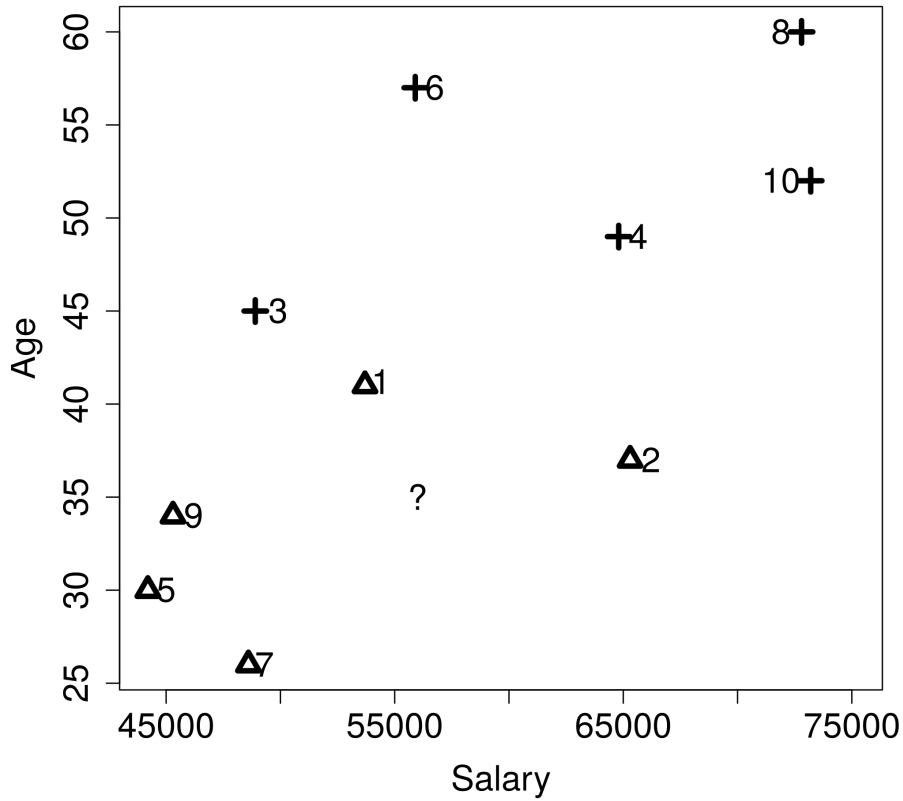


Figure 12: The salary and age feature space with the data in Table 9 [92] plotted. The instances are labelled their IDs, triangles represent the negative instances and crosses represent the positive instances. The location of the query $\langle \text{SALARY} = 56,000, \text{AGE} = 35 \rangle$ is indicated by the ?.

Data Normalization

ID	Salary	Age	Purch.	Salary and Age Dist.	Salary and Age Neigh.	Salary Only Dist.	Salary Only Neigh.	Age Only Dist.	Age Only Neigh.
1	53700	41	No	2300.0078	2	2300	2	6	4
2	65300	37	No	9300.0002	6	9300	6	2	2
3	48900	45	Yes	7100.0070	3	7100	3	10	6
4	64800	49	Yes	8800.0111	5	8800	5	14	7
5	44200	30	No	11800.0011	8	11800	8	5	5
6	55900	57	Yes	102.3914	1	100	1	22	9
7	48600	26	No	7400.0055	4	7400	4	9	3
8	72800	60	Yes	16800.0186	9	16800	9	25	10
9	45300	34	No	10700.0000	7	10700	7	1	1
10	73200	52	Yes	17200.0084	10	17200	10	17	8

Data Normalization

- This odd prediction is caused by features taking different ranges of values, this is equivalent to features having different variances.
- We can adjust for this using normalization; the equation for range normalization is:

$$a_i^{'} = \frac{a_i - min(a)}{max(a) - min(a)} \times (high - low) + low \tag{8}$$

	Normalized Dataset			Salary and Age		Salary Only		Age Only	
ID	Salary	Age	Purch.	Dist.	Neigh.	Dist.	Neigh.	Dist.	Neigh.
1	0.3276	0.4412	No	0.1935	1	0.0793	2	0.17647	4
2	0.7276	0.3235	No	0.3260	2	0.3207	6	0.05882	2
3	0.1621	0.5588	Yes	0.3827	5	0.2448	3	0.29412	6
4	0.7103	0.6765	Yes	0.5115	7	0.3034	5	0.41176	7
5	0.0000	0.1176	No	0.4327	6	0.4069	8	0.14706	3
6	0.4034	0.9118	Yes	0.6471	8	0.0034	1	0.64706	9
7	0.1517	0.0000	No	0.3677	3	0.2552	4	0.26471	5
8	0.9862	1.0000	Yes	0.9361	10	0.5793	9	0.73529	10
9	0.0379	0.2353	No	0.3701	4	0.3690	7	0.02941	1
10	1.0000	0.7647	Yes	0.7757	9	0.5931	10	0.50000	8

Data Normalization

- Normalizing the data is an important thing to do for almost all machine learning algorithms, not just nearest neighbor!

ID	RPM	VIBRATION	STATUS
1	498	604	faulty
2	517	594	faulty
3	541	574	faulty
4	555	587	faulty
5	572	537	faulty
6	600	553	faulty
7	621	482	faulty
8	632	539	faulty
9	656	476	faulty
10	653	554	faulty
11	679	516	faulty
12	688	524	faulty
13	684	450	faulty
14	699	512	faulty
15	703	505	faulty
16	717	377	faulty
17	740	377	faulty
18	749	501	faulty
19	756	492	faulty
20	752	381	faulty
21	762	508	faulty
22	781	474	faulty
23	781	480	faulty
24	804	460	faulty
25	828	346	faulty
26	830	366	faulty
27	864	344	faulty
28	882	403	faulty
29	891	338	faulty
30	921	362	faulty
31	941	301	faulty
32	965	336	faulty
33	976	297	faulty
34	994	287	faulty

ID	RPM	VIBRATION	STATUS
35	501	463	good
36	526	443	good
37	536	412	good
38	564	394	good
39	584	398	good
40	602	398	good
41	610	428	good
42	638	389	good
43	652	394	good
44	659	336	good
45	662	364	good
46	672	308	good
47	691	248	good
48	694	401	good
49	718	313	good
50	720	410	good
51	723	389	good
52	744	227	good
53	741	397	good
54	770	200	good
55	764	370	good
56	790	248	good
57	786	344	good
58	792	290	good
59	818	268	good
60	845	232	good
61	867	195	good
62	878	168	good
63	895	218	good
64	916	221	good
65	950	156	good
66	956	174	good
67	973	134	good
68	1002	121	good

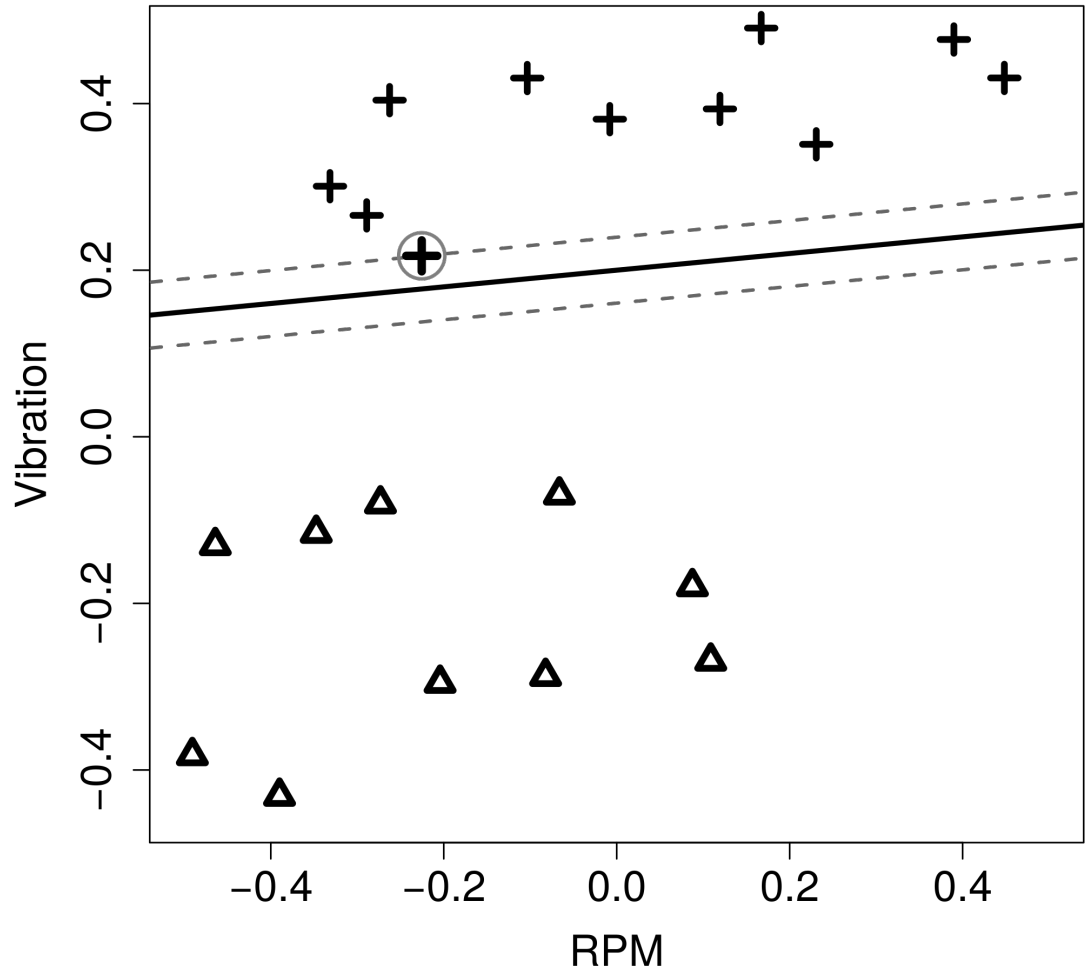


Figure 13: A small sample of the generators dataset with two features, RPM and VIBRATION, and two target levels, *'good'* (shown as crosses) and *'bad'* (shown as triangles). A decision boundary with a very small margin.

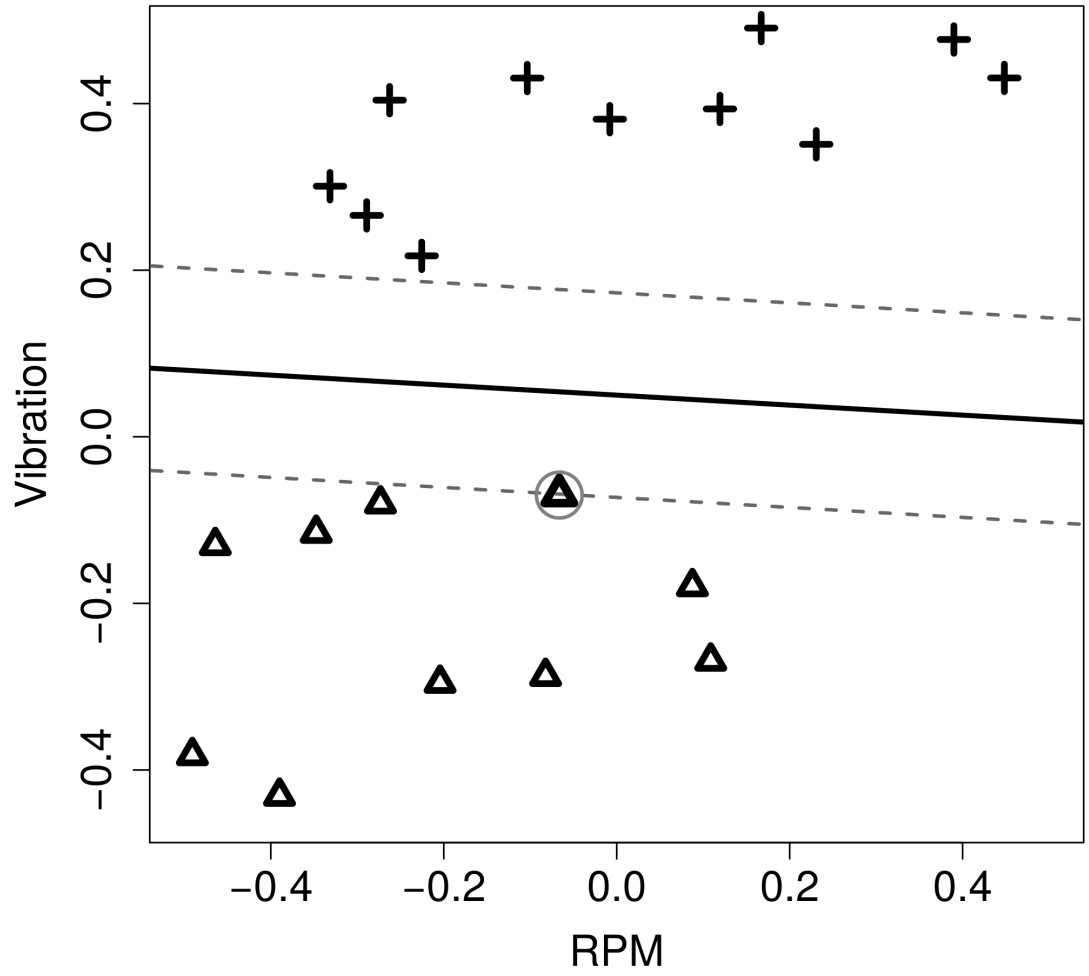


Figure 14: A small sample of the generators dataset with two features, RPM and VIBRATION, and two target levels, *'good'* (shown as crosses) and *'bad'* (shown as triangles). A decision boundary with a large margin.

- 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.
- 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:

$$w_0 + \mathbf{w} \cdot \mathbf{d} = 0 \tag{9}$$

- 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.

- A support vector machine model is defined as

$$\mathbb{M}_{\boldsymbol{\alpha},w_0}(\mathbf{q}) = \sum_{i=1}^{s} (t_i \times \boldsymbol{\alpha}[i] \times (\mathbf{d}_i \cdot \mathbf{q}) + w_0) \tag{10}$$

where

- $\mathbf{q}$ is the set of descriptive features for a query instance;
- $(\mathbf{d}_1, t_1), \ldots, (\mathbf{d}_s, t_s)$ are s support vectors (instances composed of descriptive features and a target feature);
- w_0 is the first weight of the decision boundary;
- and $\boldsymbol{\alpha}$ is a set of parameters determined during the training process (there is a parameter for each support vector $\boldsymbol{\alpha}[1], \ldots, \boldsymbol{\alpha}[s]$).[3]

[3] These parameters are formally known as **Lagrange multipliers**.

- Training a support vector machine is frames 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 required **constraints** required by the training process are

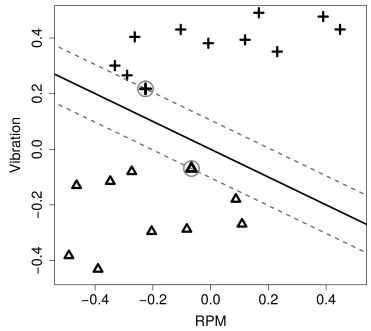
$$w_0 + \mathbf{w} \cdot \mathbf{d} \leq -1 \text{ for } t_i = -1 \tag{11}$$

and:

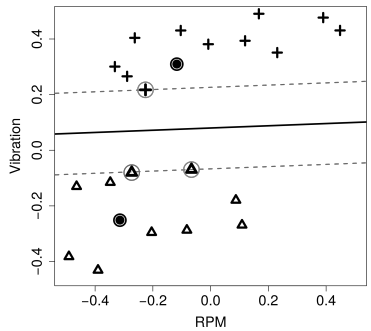
$$w_0 + \mathbf{w} \cdot \mathbf{d} \geq +1 \text{ for } t_i = +1 \tag{12}$$

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

$$t_i \times (w_0 + \mathbf{w} \cdot \mathbf{d}) \geq 1 \tag{13}$$



(a)



(b)

Figure 15: Different margins that satisfy the constraint in Equation (13)[108]. 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

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

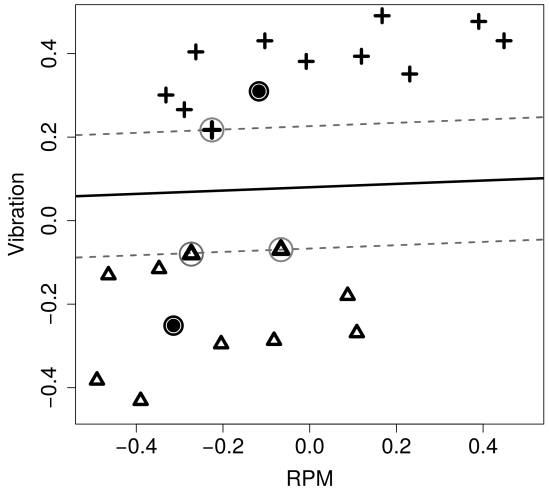
where $||\mathbf{w}||$ is known as the **Euclidean norm** of $\mathbf{w}$ and is calculated as

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

- 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

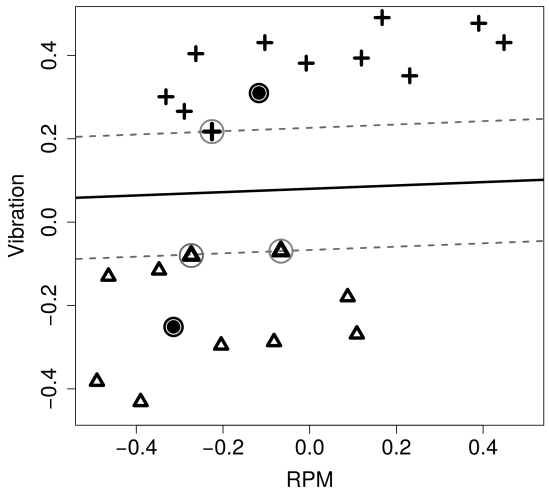
$$t_i \times (w_0 + \mathbf{w} \cdot \mathbf{d}) \geq 1$$



- 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.

- The descriptive feature values and target feature values for the support vectors in these cases are
  - $(\langle -0.225, 0.217 \rangle, +1)$,
  - $(\langle -0.066, -0.069 \rangle, -1)$,
  - $(\langle -0.273, -0.080 \rangle, -1)$.
- The value of w_0 is -0.1838,
- The values of the α parameters are

$$\langle 22.056, 6.998, 16.058 \rangle.$$



- The plot shows the position of two new query instances for this problem.
- The descriptive feature values for these querys are
  1. $\mathbf{q_1} = \langle -0.314, -0.251 \rangle$
  2. $\mathbf{q_2} = \langle -0.117, 0.31 \rangle$.

- For the first query instance, $\mathbf{q_1} = \langle -0.314, -0.251 \rangle$, the output of the support vector machine model is:

$$\begin{aligned}
&\mathbb{M}_{\boldsymbol{\alpha}, w_0}(\mathbf{q_1}) \\
&\quad = (1 \times 23.056 \times ((-0.225 \times -0.314) + (0.217 \times -0.251)) - 0.1838) \\
&\qquad + (-1 \times 6.998 \times ((-0.066 \times -0.314) + (-0.069 \times -0.251)) - 0.1838) \\
&\qquad + (-1 \times 16.058 \times ((-0.273 \times -0.314) + (-0.080 \times -0.251)) - 0.1838) \\
&\quad = -2.145
\end{aligned}$$

- The model output is less than -1, so this query is predicted to be a *'faulty'* generator.
- For the second query instance, the model output is 1.592, so this instance is predicted to be a *'good'* generator.

- **Basis functions** can be used with support vector machines to handle data that is not **linearly separable**
- To use basis functions we update Equation (13)[108] to

$$t_i \times (w_0 + \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{d})) \geq 1 \text{ for all } i \tag{14}$$

where $\boldsymbol{\phi}$ is a set of basis functions applied to the descriptive features $\mathbf{d}$, and $\mathbf{w}$ is a set of weights containing one weight for each member of $\boldsymbol{\phi}$.

- Typically, the number of basis functions in ϕ is larger than the number of descriptive features, so the application of the basis functions moves the data into a higher-dimensional space.
- The expectation is that a linear separating hyperplane will exist in this higher-dimensional space even though it does not in the original feature space.

- The prediction model in this case becomes

$$\mathbb{M}_{\boldsymbol{\alpha},\boldsymbol{\phi},w_0}(\mathbf{q}) = \sum_{i=1}^{s} (t_i \times \boldsymbol{\alpha}[i] \times (\boldsymbol{\phi}(\mathbf{d}_i) \cdot \boldsymbol{\phi}(\mathbf{q})) + w_0) \tag{15}$$

- This equation requires a dot product calculation between the result of applying the basis functions to the query instance and to each of the support vectors which is repeated multiple times during the training process.

- A dot product is a computationally expensive operation,
- We can use a clever trick is used to avoid it:
  - the same result obtained by calculating the dot product of the descriptive features of a support vector and a query instance after having applied the basis functions can be obtained by applying a much less costly **kernel function**, $kernel$, to the original descriptive feature values of the support vector and the query.

- The prediction equation becomes

$$\mathbb{M}_{\boldsymbol{\alpha},kernel,w_0}(\mathbf{q}) = \sum_{i=1}^{s} (t_i \times \boldsymbol{\alpha}[i] \times kernel(\mathbf{d}_i, \mathbf{q}) + w_0) \tag{16}$$

- A wide range of standard kernel functions can be used with support vector machines 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

Multi-layer Perceptron is a Neural Network

- A neural network is a machine designed to model the way in which the brain performs a particular task or function of interest.
- A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use (Haykin 2009).

It resembles the brain in two respects (Haykin 2009):

1. Knowledge is acquired by the network from its environment through a learning process.
2. Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Models of a neuron

- A neuron is an information-processing unit fundamental to the operation of a neural network
- Consists of:
  1. Synapse or connecting links: each characterized by a weight (ω_{kj}) or strength of its own. Note a signal x_j at the input of synapse j, connected to neuron k is multiplied by the synaptic weight ω_{kj}.
  2. Adder: sums the input signals(x_i), weighted by the respective synaptic strengths of the neuron
  3. Activation (or squashing) function: limits the amplitude of the output of a neuron; squashes permissible amplitude range of the output signal to some finite value.

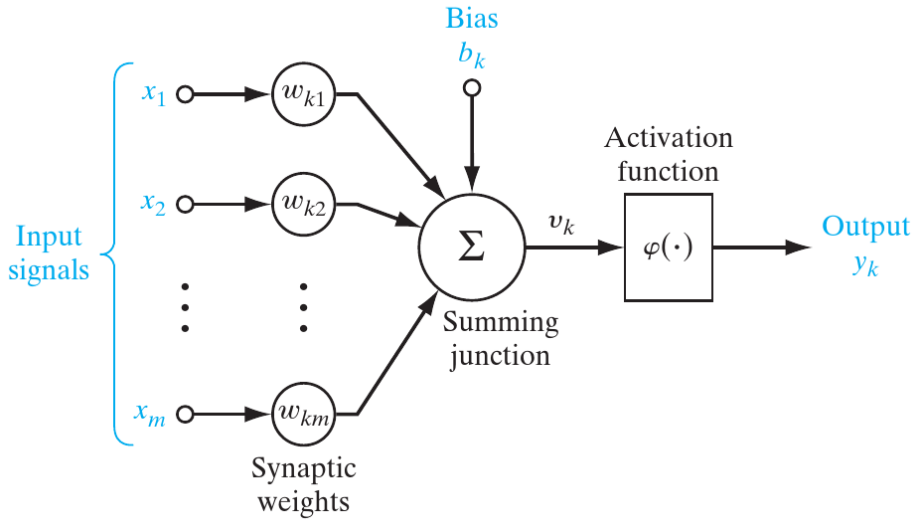


Figure 16: Model of a neuron with bias b_k which increases or lowers the net input of the activation function (Haykin 2009).

Models of a neuron

Operation of neuron in Figure (16) can be written mathematically as

$$u_k = \sum_{j=1}^{m} \omega_{kj} x_j \tag{17}$$

$$y_k = \varphi(u_k + b_k) \tag{18}$$

where

- $x_1, x_2, \ldots x_m$ are the input signals;
- $\omega_1, \omega_2, \ldots, \omega_m$ are the respective synaptic weights of neuron k;
- u_k is the linear combiner output due to the input signals
- b_k is the bias;
- $\varphi(\cdot)$ is the activation function;

Bias b_k applies an affine transformation to the output u_k of the linear combiner

$$v_k = u_k + b_k \tag{19}$$

Models of a neuron

Equations (17) - (19) can be combined into

$$v_k = \sum_{j=0}^{m} \omega_{kj} x_j \qquad (20)$$

and

$$y_k = \varphi(v_k) \qquad (21)$$

In combining the equations a new synapse has been added with input

$$x_0 = +1 \qquad (22)$$

and weight

$$\omega_{k0} = b_k \qquad (23)$$

See Figure (17).

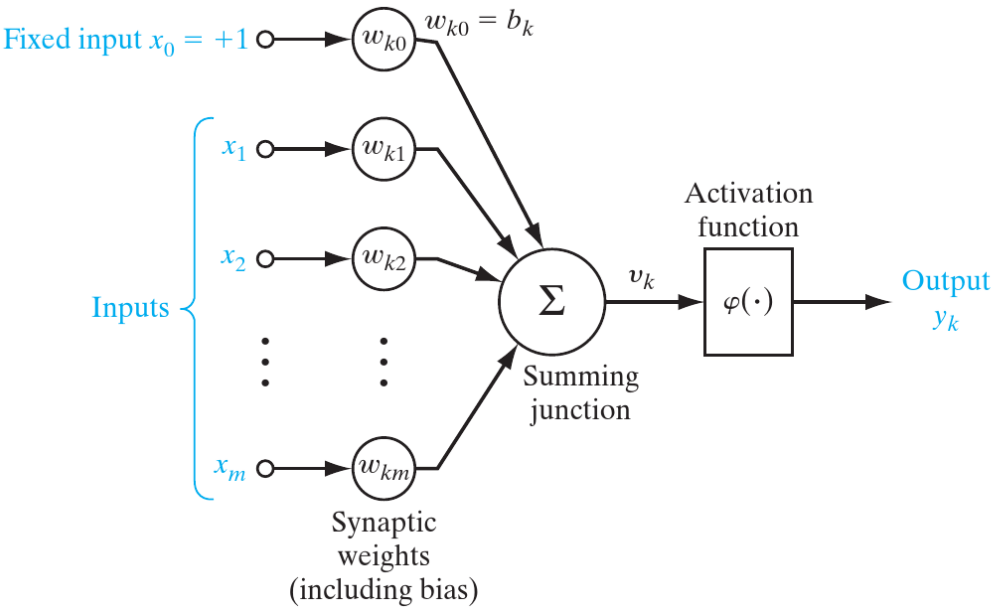


Figure 17: Model of neuron with the bias absorbed into the neuron (Haykin 2009).

Common Activation Functions

Threshold Function depicted in Figure (18) can be written as:

$$\varphi(v) = \begin{cases} 1 & \text{if} \quad v \geq 0 \\ 0 & \text{if} \quad v < 0 \end{cases} \tag{24}$$

Output of neuron, k, using threshold function is

$$y_k = \begin{cases} 1 & \text{if} \quad v_k \geq 0 \\ 0 & \text{if} \quad v_k < 0 \end{cases} \tag{25}$$

and induced local field of neuron, v_k is

$$v_k = \sum_{j=1}^{m} \omega_{kj} x_j + b_k \tag{26}$$

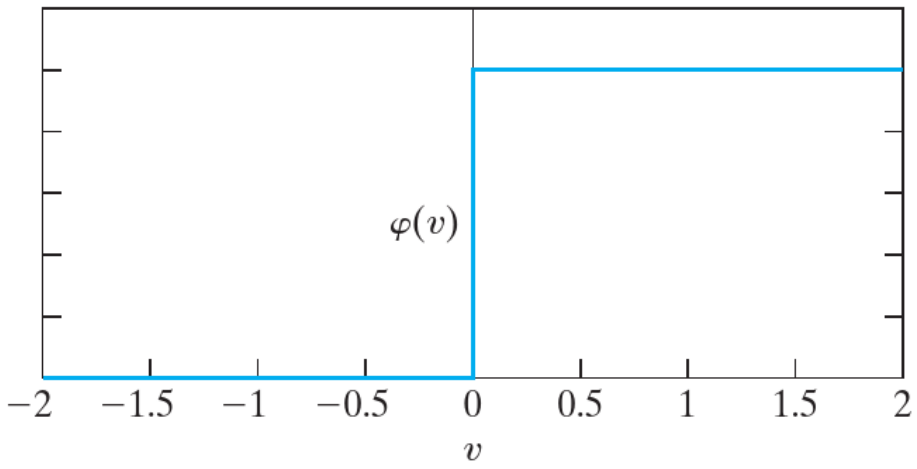


Figure 18: Threshold function (Haykin 2009).

Common Activation Functions

Logistic Function (an example of Sigmoid function) is depicted in Figure (19) and can be written as:

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \qquad (27)$$

where induced local field of neuron, v_k is

$$v_k = \sum_{j=1}^{m} \omega_{kj} x_j + b_k \qquad (28)$$

and slope parameter a determines the shape

- Note that the logistic function is differentiable while the threshold function is not

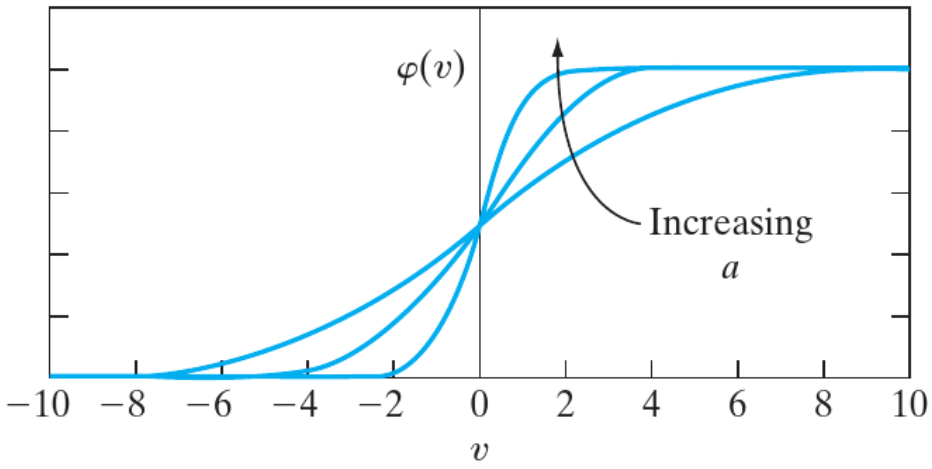


Figure 19: Sigmoid function for varying slope parameter a (Haykin 2009).

Common Activation Functions

- Rectified Linear Unit (ReLU) has become very popular since its introduction by Nair & Hinton (2010).
- Output is a non-linear function of the input

$$v_k = \sum_{j=1}^{m} \omega_{kj} x_j + b_k \tag{29}$$

$$y_k = \begin{cases} v_k & \text{if} \quad v_k > 0 \\ 0 & \text{if} \quad v_k < 0 \end{cases} \tag{30}$$

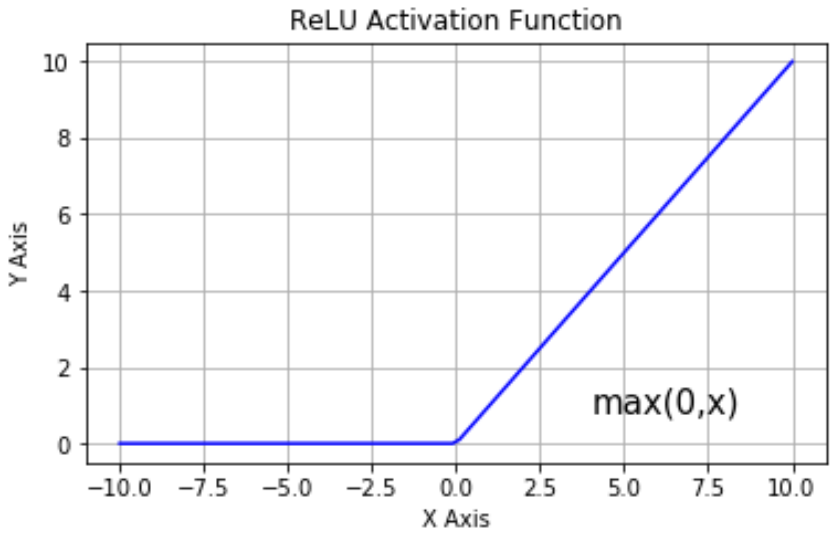


Figure 20: Rectified Linear Unit

Common Activation Functions

- Softmax activation function squashes each input to a value between 0 and 1.
- Output is equivalent to a categorical probability distribution
- Graph similar to logistic but usually applied to provide probabilistic interpretation to outputs in classification task

$$v_k = \sum_{j=1}^{m} \omega_{kj} x_j + b_k \tag{31}$$

$$y_k = \frac{\exp(v_k)}{\sum_{k=1}^{K} \exp(v_k)} \tag{32}$$

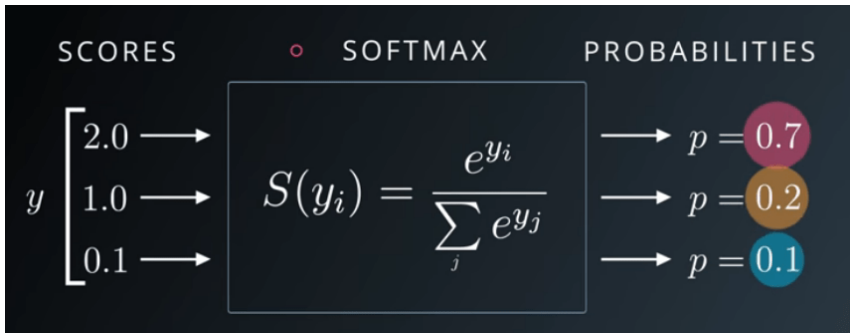


Figure 21: Softmax operation for a 3-class classification task (`https://sefiks.com/`).

Common Activation Functions

What are some nice properties of activation functions?

- Non-linear function; otherwise neural net can only solve simple problems;
- Without activation neural net is equivalent to a linear regression
- Nice derivatives makes learning easy
- Activation functions should give a bounded output for a bounded input
- Choosing the right activation function is both science and art. For further insight, see the works of Ramachandran et al. (2017) and Mhaskar & Micchelli (1994)
- Together with the right cost function, activation functions make training NN possible.

Models of a neuron

- In Figure (22) consider only 3 inputs and the bias into the neuron;
- Let the weights be $\omega_{10} = b_1 = 0.5$, $\omega_{11} = 0.4$ $\omega_{12} = 0.6$; $\omega_{13} = 0.2$
- Let the inputs be $x_0 = 1$; $x_1 = 1.2$; $x_2 = 2.0$; $x_3 = 1.8$
- Let the activation function be logistic sigmod with $a = 0.2$

$$
\begin{aligned}
v_1 &= \sum_{j=0}^{3} \omega_{1j} x_j \\
&= 1 \times 0.5 + 0.4 \times 1.2 + 0.6 \times 2.0 + 0.2 \times 1.8 \\
&= 2.54
\end{aligned}
$$

$$
\begin{aligned}
y_1 = \varphi(v_1) &= \frac{1}{1 + \exp(-a v_1)} \\
&= \frac{1}{1 + \exp(-0.2 \times 2.54)} = 0.624
\end{aligned}
$$

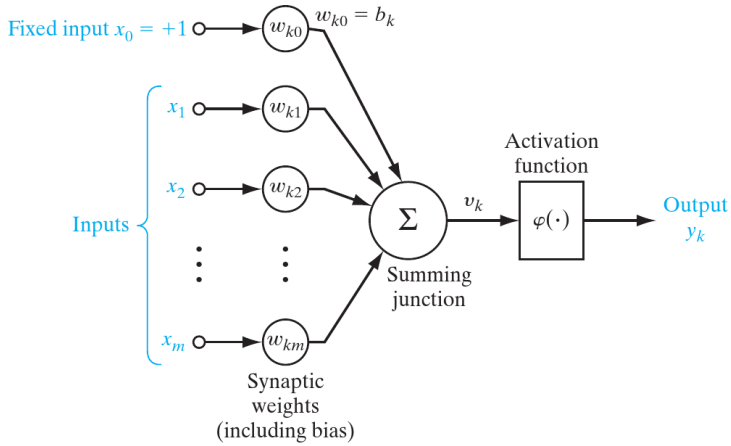


Figure 22: Model of neuron: Example computation (Haykin 2009).

Network Architecture

Single Layer Feedforward Networks

- Input layer of source nodes project directly onto an output layer of neurons

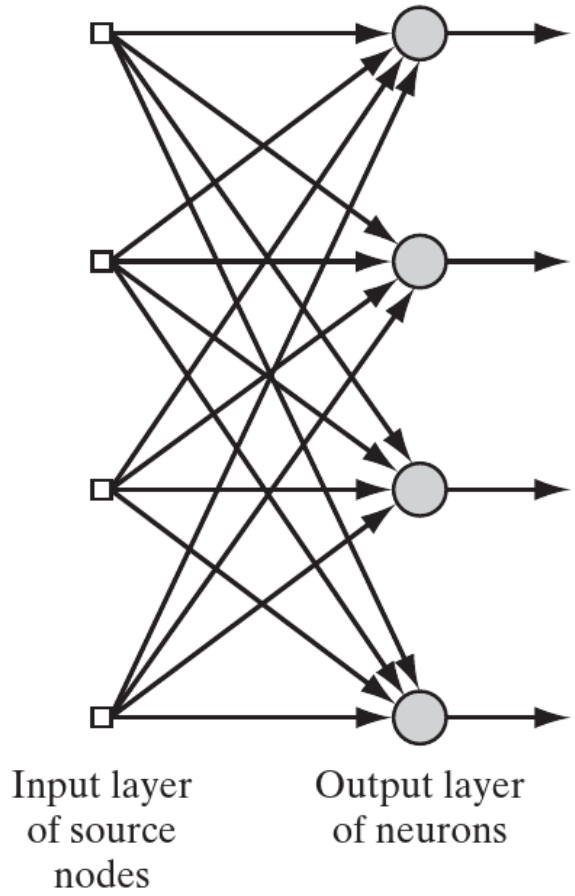


Figure 23: Single Layer Feedforward NN (Haykin (2009))

Network Architecture

Multilayer Feedforward Networks

- Input layer of source nodes project directly onto a set of neurons in a hidden layer
- There could be one or more hidden layers; output of each layer forming input to the next layer
- Adding one or more hidden layers allows network to extract higher-order statistics from the input data
- Network is fully connected if every node in each layer is connected to every node in the adjacent forward layer

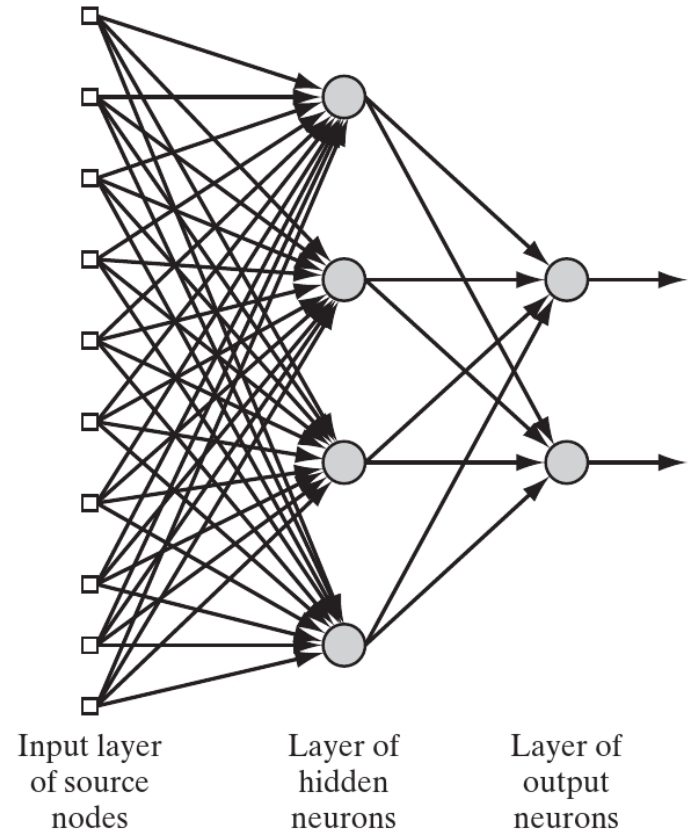


Figure 24: Multilayer Fully Connected Feedforward NN (Haykin (2009))

Learning process

Types of Learning

- Supervised learning - predict an output when given an input vector
- Reinforcement learning - select an action to maximize some defined payoff
- Unsupervised learning - discover a good internal representation of the data

Learning process

Supervised Learning

- Each training case consists of an input vector x and a target output t.
  1. Regression: The target output is a real number or a whole vector of real numbers.
  2. Classification: The target output is a class label.

Recall that in general we want to learn a mapping from input vector $\boldsymbol{x}$ to some output y through a vector of weights $\boldsymbol{\omega}$

$$y = f(\boldsymbol{\omega}, \boldsymbol{x}) \tag{33}$$

such that the error (or loss or cost function) incurred in the prediction of the actual value is minimized.

- For regression, the cost function

$$J(\boldsymbol{\omega}, b) = -\mathbb{E} \log p_{\text{model}}(y|\boldsymbol{x}) \tag{34}$$

is the expectation of negative conditional log-likelihood computed over the training data; the cross-entropy between the training data and the model distribution

Learning process

- Cost function in Equation (34) is usually minimized in an optimization process, specifically gradient descent because of the non-convexity of objective function.
- How to understand gradient-based optimization? (Goodfellow et al. 2016, p. 80)
  - Consider a function $y = f(x)$ where both x and y are real numbers
  - Derivative of $y = f(x)$, $f'(x)$, gives slope of $f(x)$ at point x
  - Importantly, it tells us how to scale a small change in the input to obtain corresponding change in output (this is due to Taylor's expansion):

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x) \tag{35}$$

$$f(x - \epsilon \text{ sign}(f'(x))) < f(x) \quad \text{for small enough} \quad \epsilon$$

So we reduce $f(x)$ by moving x in small steps with the opposite sign of the derivative

  - This technique is called gradient descent [4] and credited to Louis Augustin Cauchy, 1847 (it's also called steepest descent)

[4] For brief (mathematical) historical account see Lemarechal (2012)

Learning process

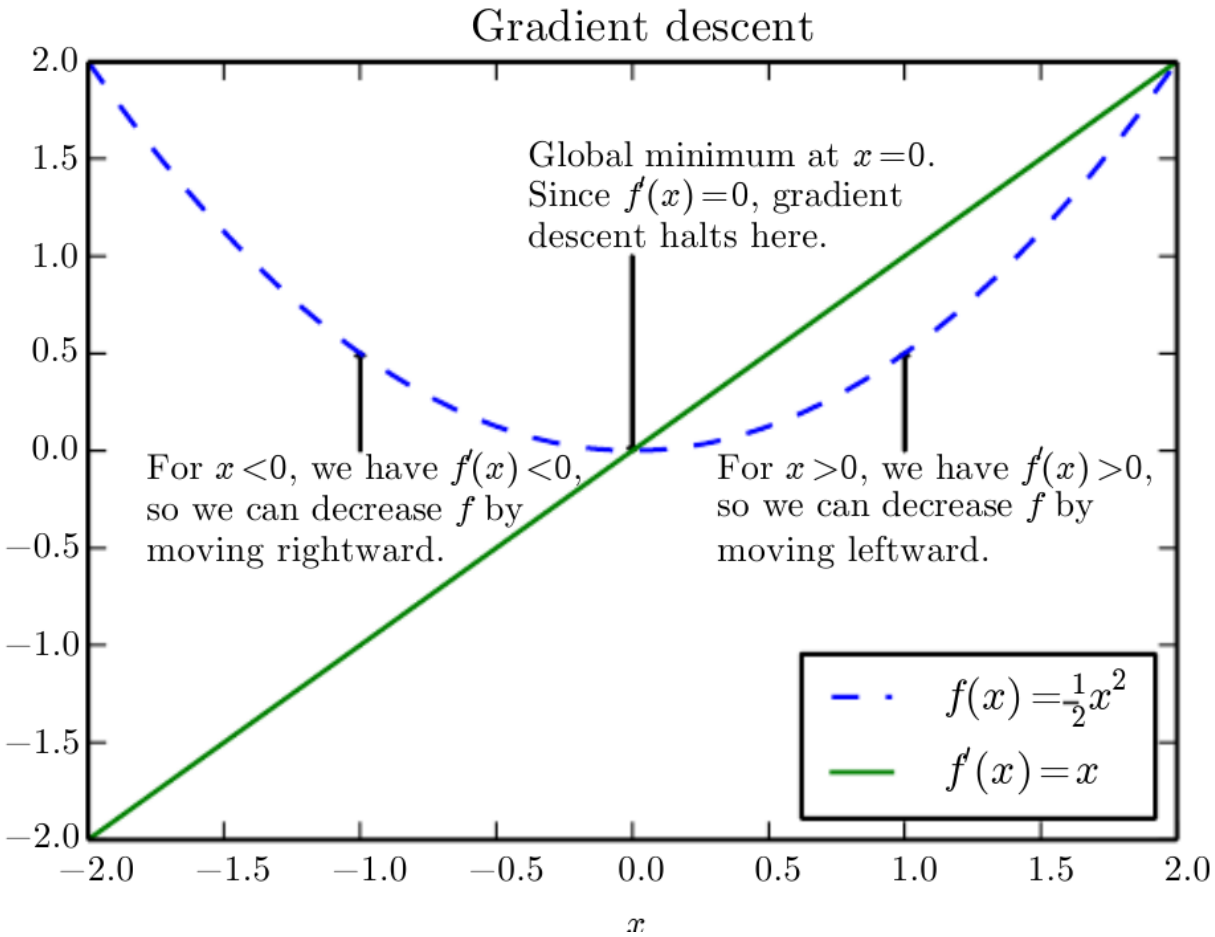


Figure 25: Illustraion of the gradient descent algorithm (Goodfellow et al. 2016, p.80)

Multilayer Perceptron

Basic features of multilayer perceptrons (Haykin 2009) (See Figure 26):

- Each neuron in the network includes a nonlinear activation function that is differentiable
- Network contains one or more layers that are hidden from both the input and output nodes
- Network exhibits a high degree of connectivity determined by synaptic weights of the network

Training method

Multilayer perceptron is usually trained using the back-propagation algorithm:

- Forward phase: Weights of the network are fixed and input signal is propagated layer-wise through the network and transformed signal appears at the output
- Backward phase: Error signal is computed by comparing generated output and desired response; error signal is propagated backward and layer-wise through the network; successive adjustments made to weights of the network

Multilayer Perceptron

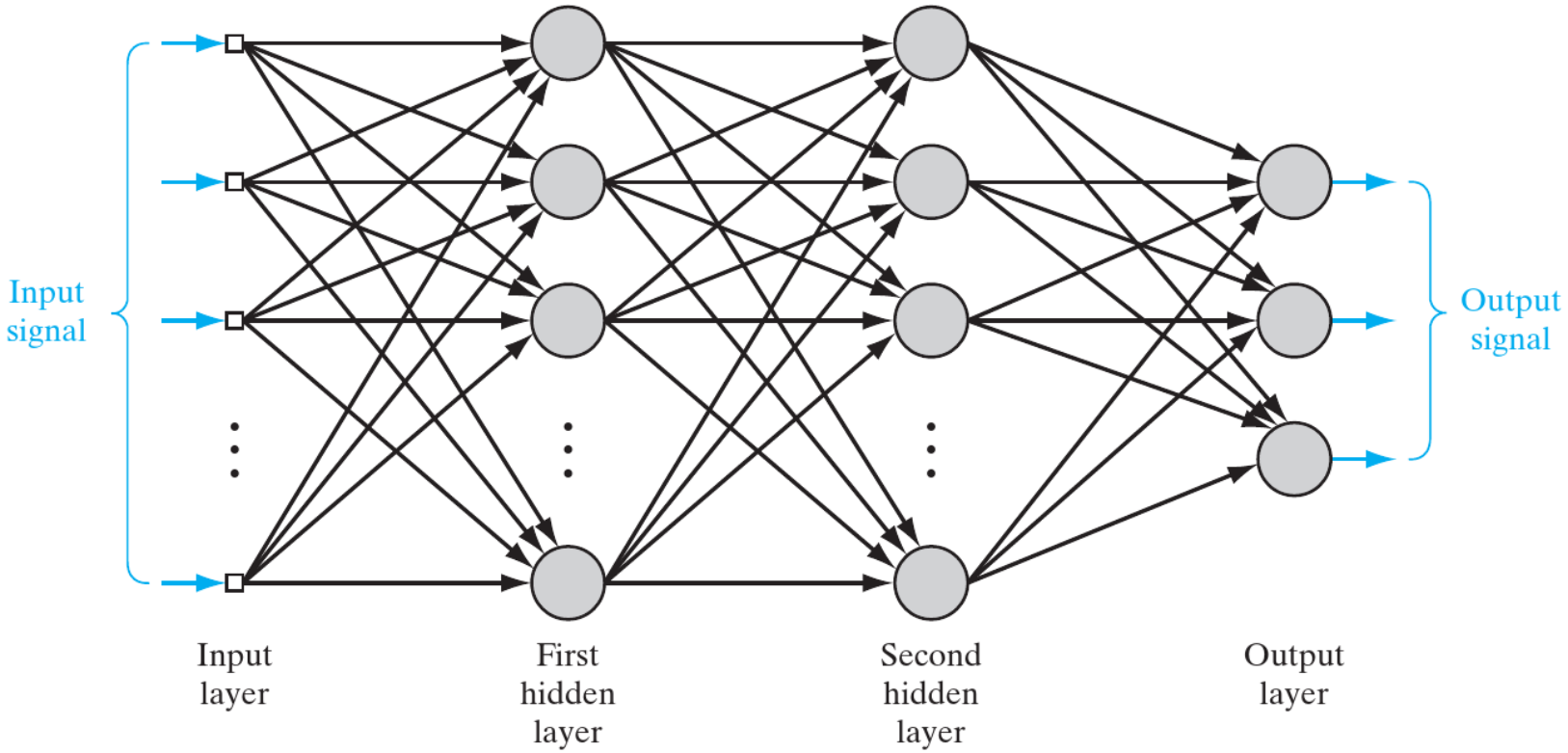


Figure 26: Architectural graph of the Multilayer Perceptron (Haykin 2009)

Multilayer Perceptron

- Each hidden or output neuron performs two computations:
  1. Output of each neuron expressed as continuous nonlinear function of input signals and associated weights
  2. Estimate of the gradient vector (gradient of error surface) required in the backward phase of the training
- Hidden neurons act as feature detectors, discovering the salient features characterising the training data;
- Hidden neurons perform nonlinear transformation on input data into a new space; feature space
- The training is a form of error-correction learning that assigns blame or credit to each of the internal neurons; this is a case of the credit assignment problem
- Back-propagation solves the credit assignment problem for the multilayer perceptron

Back-propagation Algorithm

Key points leading to overall strategy

- Multilayer perceptron is a universal function approximator
- It can be trained using error-correction learning to obtain optimum approximation
- The optimum can be obtained if we can minimize the approximation error
- This is equivalent to modifying the weights so that the network minimizes the error between desired output and response of the network
- Gradient descent algorithm can be used to find the minimum of an objective function by iteratively computing the adjustment that leads to the minimization of the objective function
- Back-propagation is an efficient implementation of the gradient descent
- Strategy is to compute the adjustment, $\Delta\omega$ to be applied to each weight, ω
- The adjustment is proportional to the gradient of the objective function; in this case ∇E (E is error signal energy) with respect to the parameters ω

Back-propagation Algorithm

- Error signal of the output neuron is given by

$$e_j(n) = d_j(n) - y_j(n) \tag{36}$$

where y_j is the output of neuron j when stimulus $\boldsymbol{x}(n)$ is applied at the input; $d_j(n)$ is the desired output

- Instantaneous error energy can be written as

$$E_j(n) = \frac{1}{2} e_j^2(n) \tag{37}$$

- Total instantaneous error (summed over all neurons in the output layer) is

$$E(n) = \sum_{j \in C} E_j(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \tag{38}$$

- Computation of the error could be in batch mode or on-line mode leading to either batch mode (presentation of all training samples) or on-line (presentation of training sample one-at-a-time) training

Back-propagation Algorithm

Consider Figure (27):

- Induced local field of neuron j at iteration n is:

$$v_j(n) = \sum_{i=0}^{m} \omega_{ji}(n) y_i(n) \tag{39}$$

m is the total number of inputs

- Function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n)) \tag{40}$$

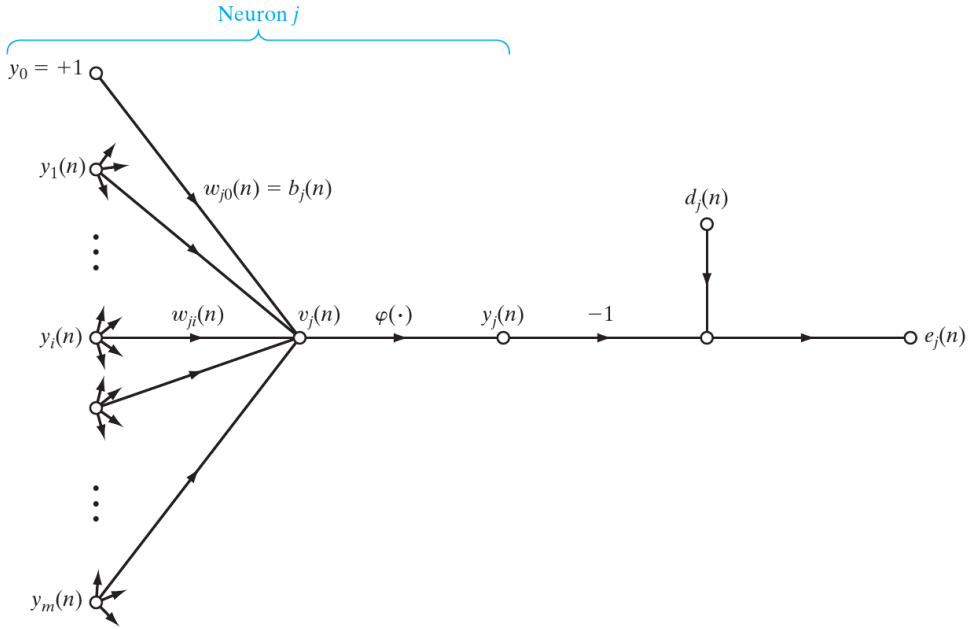


Figure 27: Signal flow highlighting neuron j being fed by the outputs from the neurons to its left; induced local field of neuron is $v_j(n)$ and this is the input to activation function $\varphi(\cdot)$ (Haykin 2009)

Back-propagation Algorithm

- We need to compute the adjustment (or correction) $\Delta\omega_{ji}(n)$ to be applied to weight $\omega_{ji}(n)$
- This is proportional to the partial derivative $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ and determines the direction of search in the weight space for ω_{ji}
- Chain rule tells us how to compute $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ from a set of known quantities

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)} \tag{41}$$

- Recall Equation (37) : $E_j(n) = \frac{1}{2} e_j^2(n)$; therefore

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{42}$$

Back-propagation Algorithm

- We need to compute the adjustment (or correction) $\Delta\omega_{ji}(n)$ to be applied to weight $\omega_{ji}(n)$
- This is proportional to the partial derivative $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ and determines the direction of search in the weight space for ω_{ji}
- Chain rule tells us how to compute $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ from a set of known quantities

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)} \tag{41}$$

- Recall Equation (37) : $E_j(n) = \frac{1}{2} e_j^2(n)$; therefore

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{42}$$

Back-propagation Algorithm

- We need to compute the adjustment (or correction) $\Delta\omega_{ji}(n)$ to be applied to weight $\omega_{ji}(n)$
- This is proportional to the partial derivative $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ and determines the direction of search in the weight space for ω_{ji}
- Chain rule tells us how to compute $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ from a set of known quantities

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)} \tag{41}$$

- Recall Equation (37) : $E_j(n) = \frac{1}{2} e_j^2(n)$; therefore

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{42}$$

Back-propagation Algorithm

- We need to compute the adjustment (or correction) $\Delta\omega_{ji}(n)$ to be applied to weight $\omega_{ji}(n)$
- This is proportional to the partial derivative $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ and determines the direction of search in the weight space for ω_{ji}
- Chain rule tells us how to compute $\frac{\partial E(n)}{\partial \omega_{ji}(n)}$ from a set of known quantities

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)} \tag{41}$$

- Recall Equation (37) : $E_j(n) = \frac{1}{2} e_j^2(n)$; therefore

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{42}$$

Back-propagation Algorithm

- Recall Equation (36): $e_j(n) = d_j(n) - y_j(n)$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{43}$$

- Recall Equation (40): $y_j(n) = \varphi_j(v_j(n))$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)); \text{ where } ()' \text{ indicates differentiation} \tag{44}$$

- Recall Equation(39): $v_j(n) = \sum_{i=0}^{m} \omega_{ji}(n) y_i(n)$

$$\frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = y_i(n) \tag{45}$$

- Equation (41) becomes (using Equations (42) - (45))

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{46}$$

Back-propagation Algorithm

- Recall Equation (36): $e_j(n) = d_j(n) - y_j(n)$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{43}$$

- Recall Equation (40): $y_j(n) = \varphi_j(v_j(n))$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)); \text{ where } ()' \text{ indicates differentiation} \tag{44}$$

- Recall Equation(39): $v_j(n) = \sum_{i=0}^{m} \omega_{ji}(n) y_i(n)$

$$\frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = y_i(n) \tag{45}$$

- Equation (41) becomes (using Equations (42) - (45))

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{46}$$

Back-propagation Algorithm

- Recall Equation (36): $e_j(n) = d_j(n) - y_j(n)$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{43}$$

- Recall Equation (40): $y_j(n) = \varphi_j(v_j(n))$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)); \text{ where } ()' \text{ indicates differentiation} \tag{44}$$

- Recall Equation(39): $v_j(n) = \sum_{i=0}^{m} \omega_{ji}(n) y_i(n)$

$$\frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = y_i(n) \tag{45}$$

- Equation (41) becomes (using Equations (42) - (45))

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{46}$$

Back-propagation Algorithm

- Recall Equation (36): $e_j(n) = d_j(n) - y_j(n)$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{43}$$

- Recall Equation (40): $y_j(n) = \varphi_j(v_j(n))$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)); \text{ where } ()' \text{ indicates differentiation} \tag{44}$$

- Recall Equation(39): $v_j(n) = \sum_{i=0}^{m} \omega_{ji}(n) y_i(n)$

$$\frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = y_i(n) \tag{45}$$

- Equation (41) becomes (using Equations (42) - (45))

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{46}$$

Back-propagation Algorithm

- Correction, $\Delta\omega_{ji}(n)$, applied to $\omega_{ji}(n)$ is defined by the delta rule

$$\begin{aligned}\Delta\omega_{ji}(n) &= -\eta\frac{\partial E(n)}{\partial\omega_{ji}(n)}; \quad \eta \text{ is the learning rate parameter} \\ &= \eta\boxed{e_j(n)\varphi_j'(v_j(n))}y_i(n) \\ &= \eta\boxed{\delta_j(n)}y_i(n) \end{aligned} \tag{47}$$

where $\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$ is defined as the local gradient for neuron j

- Local gradient for neuron j is the product of corresponding error $e_j(n)$ and the derivative of associated activation function, $\varphi_j'(v_j(n))$
- Error $e_j(n)$ is easily computed for the output neurons; we have access to $d_j(n)$ and $y_j(n)$. How to compute error for hidden neurons? These have no given $d_j(n)$.

Back-propagation Algorithm

- Correction, $\Delta\omega_{ji}(n)$, applied to $\omega_{ji}(n)$ is defined by the delta rule

$$\begin{aligned}\Delta\omega_{ji}(n) &= -\eta\frac{\partial E(n)}{\partial\omega_{ji}(n)}; \quad \eta \text{ is the learning rate parameter} \\ &= \eta\boxed{e_j(n)\varphi_j'(v_j(n))}y_i(n) \\ &= \eta\boxed{\delta_j(n)}y_i(n) \end{aligned} \tag{47}$$

  where $\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$ is defined as the local gradient for neuron j

- Local gradient for neuron j is the product of corresponding error $e_j(n)$ and the derivative of associated activation function, $\varphi_j'(v_j(n))$

- Error $e_j(n)$ is easily computed for the output neurons; we have access to $d_j(n)$ and $y_j(n)$. How to compute error for hidden neurons? These have no given $d_j(n)$.

Back-propagation Algorithm

- Correction, $\Delta\omega_{ji}(n)$, applied to $\omega_{ji}(n)$ is defined by the delta rule

$$\begin{aligned}\Delta\omega_{ji}(n) &= -\eta\frac{\partial E(n)}{\partial\omega_{ji}(n)}; \quad \eta \text{ is the learning rate parameter} \\ &= \eta\boxed{e_j(n)\varphi_j'(v_j(n))}y_i(n) \\ &= \eta\boxed{\delta_j(n)}y_i(n) \end{aligned} \tag{47}$$

  where $\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$ is defined as the local gradient for neuron j

- Local gradient for neuron j is the product of corresponding error $e_j(n)$ and the derivative of associated activation function, $\varphi_j'(v_j(n))$

- Error $e_j(n)$ is easily computed for the output neurons; we have access to $d_j(n)$ and $y_j(n)$. How to compute error for hidden neurons? These have no given $d_j(n)$.

Back-propagation Algorithm

What do we know so far?

1. Training a multilayer perceptron involves using the training data set in an error-correction learning paradigm to adjust the weights
2. The error-correction learning is essentially equivalent to solving a function minimization problem
3. The function to be minimized is the error surface corresponding to the mismatch between the response of the network and the desired response
4. This can be solved by the gradient descent algorithm
5. The back-propagation algorithm is an efficient implementation of the gradient descent algorithm for the multilayer perceptron
6. The correction (or update) to the weight at each iteration is (cf. Equation (47)):

$$\begin{aligned} \Delta\omega_{ji}(n) &= \eta \boxed{e_j(n)\varphi_j'(v_j(n))} y_i(n) \\ &= \eta \boxed{\delta_j(n)} y_i(n) \end{aligned} \tag{48}$$

This is the product of the learning rate η, local gradient of the associated neuron, $\delta_j(n)$ and the input to the neuron, $y_i(n)$. See Figure (27)

Back-propagation Algorithm

- Weights connected to the output neurons are updated as

$$\begin{aligned}\omega_{ji}^{new}(n) &= \omega_{ji}^{old}(n) + \Delta\omega_{ji}(n) \\ &= \omega_{ji}^{old}(n) + \eta \boxed{\delta_j(n)} y_i(n) \\ &= \omega_{ji}^{old}(n) + \eta \boxed{e_j(n)\varphi_j'(v_j(n))} y_i(n)\end{aligned} \tag{49}$$

- Using chain rule similarly to how we derive the update for the weight of output neurons we will show that the weight update for hidden neurons is given as

$$\begin{aligned}\omega_{ji}^{new}(n) &= \omega_{ji}^{old}(n) + \Delta\omega_{ji}(n) \\ &= \omega_{ji}^{old}(n) + \eta \boxed{\delta_j(n)} y_i(n) \\ &= \omega_{ji}^{old}(n) + \eta \boxed{\varphi_j'(v_j(n)) \sum_k \delta_k(n)\omega_{kj}(n)} y_i(n)\end{aligned} \tag{50}$$

where neuron j is hidden; $\varphi_j'(v_j(n))$ is derivative of associated activation function; $\delta_k(n)$ are associated with neurons k which are to the immediate right of neuron j and connected to it; $\omega_{kj}(n)$ are the associated weights of these connections (see Figure (28))

Back-propagation Algorithm

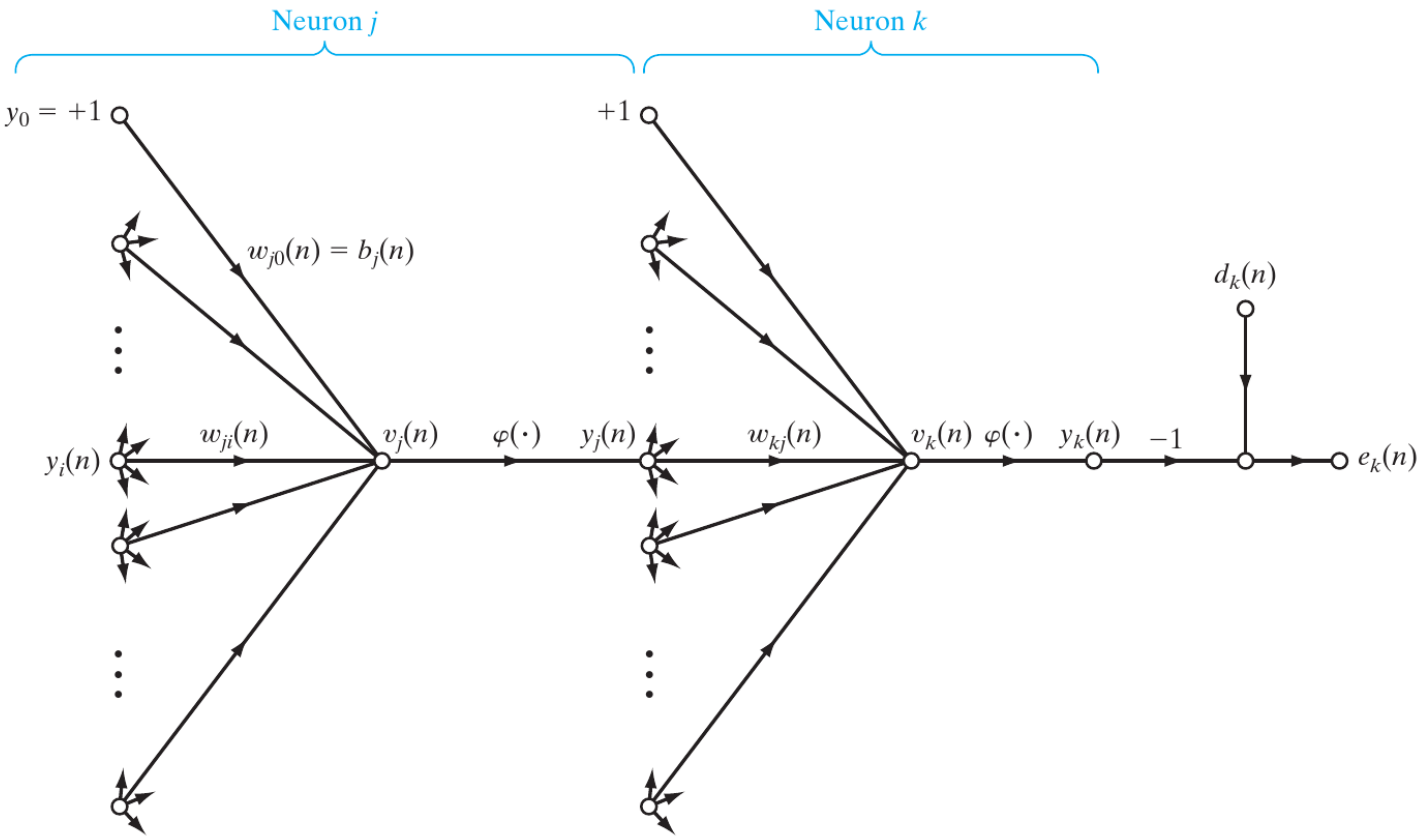


Figure 28: Signal flow showing hidden neuron j connected to an output neuron k to its immediate right; Diagram used to show the derivation of weight update for hidden neuron (Haykin 2009)

Back-propagation Algorithm

For the sake of completeness we now derive

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n)\omega_{kj}(n)$$

of Equation (50)

- Recall from Equation(41)

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \boxed{\frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)}$$

and Equation(47)

$$\begin{aligned} \Delta\omega_{ji}(n) &= \eta \boxed{e_j(n)\varphi_j'(v_j(n))} y_i(n) \\ &= \eta \boxed{\delta_j(n)} y_i(n) \end{aligned}$$

we infer that the local gradient, $\delta_j(n)$, can be written as

$$\delta_j(n) = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \tag{51}$$

Back-propagation Algorithm

- Use Figure (28) and Equation (51) to write local gradient as:

$$\begin{aligned}\delta_j(n) &= -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial E(n)}{\partial y_j(n)}\varphi_j'(v_j(n))\end{aligned} \tag{52}$$

- From Figure (28)

$$E(n) = \frac{1}{2}\sum_{k\in C} e_k^2(n);\ \text{neuron } k \text{ is an output node} \tag{53}$$

Differentiating both sides of Equation (53) with respect to y_j:

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)} \tag{54}$$

Use chain rule to write

$$\frac{\partial e_k(n)}{\partial y_j(n)} = \frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}$$

and

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)} \tag{55}$$

Back-propagation Algorithm

- Observe from Figure (28) that

$$\begin{aligned} e_k(n) &= d_k(n) - y_k(n) \\ &= d_k(n) - \varphi_k(v_k(n)); \text{ neuron } k \text{ is an output node} \end{aligned} \tag{56}$$

and we can write

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)) \tag{57}$$

- Also note that the induced local field for neuron k

$$v_k(n) = \sum_{j=0}^{m} \omega_{kj}(n) y_j(n); \; m \text{ is number of inputs applied to neuron } k \tag{58}$$

Upon differentiation we have

$$\frac{\partial v_k(n)}{\partial y_j(n)} = \omega_{kj}(n) \tag{59}$$

- Combining these component partial derivatives we obtain

$$\begin{aligned} \frac{\partial E(n)}{\partial y_j(n)} &= -\sum_k \boxed{e_k(n)\varphi_k'(v_k(n))}\, \omega_{kj}(n) \\ &= -\sum_k \delta_k(n)\omega_{kj}(n) \end{aligned} \tag{60}$$

Back-propagation Algorithm

- Substituting Equation (60) into Equation (52) to obtain

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n)\omega_{kj}(n) \tag{61}$$

and when combined with Equation (47) we can write the correction as

$$\begin{aligned} \Delta\omega_{ji}(n) &= \eta\delta_j(n)y_i(n) \\ &= \eta\varphi_j'(v_j(n)) \sum_k \delta_k(n)\omega_{kj}(n)y_i(n) \end{aligned} \tag{62}$$

and the update rule as

$$\begin{aligned} \omega_{ji}^{\text{new}}(n) &= \omega_{ji}^{\text{old}}(n) + \Delta\omega_{ji}(n) \\ &= \eta\delta_j(n)y_i(n) \\ &= \omega_{ji}^{\text{old}}(n) + \eta\varphi_j'(v_j(n)) \sum_k \delta_k(n)\omega_{kj}(n)y_i(n) \end{aligned} \tag{63}$$

which is the same expression we provided in Equation (50)

Back-propagation

Summary of Back-propagation Algorithm for Multilayer Perceptron

1. Training could be Online (weight update after presentation of each sample) or Batch (weight update after presentation of all samples)
2. Back-propagation comprises two phases namely Forward pass and Backward pass
3. Forward pass: Weights of the network are fixed and input signal is propagated layer-wise through the network and transformed signal appears at the output; each neuron computes (see Figure (27))

$$v_j(n) = \sum_{j=0}^{m} \omega_{ji}(n) y_i(n); \quad y_j(n) = \varphi_j(v_j(n)) \tag{64}$$

4. In the Backward pass error is propagated backward through the network to compute weight updates (see Figure (28) and Equation (60)):

$$\omega_{ji}^{\text{new}}(n) = \omega_{ji}^{\text{old}}(n) + \begin{cases} \eta \boxed{e_j(n)\varphi_j'(v_j(n))} y_i(n) & \text{for output neurons} \\ \eta \boxed{\varphi_j'(v_j(n)) \sum_k \delta_k(n)\omega_{kj}(n)} y_i(n) & \text{for hidden neurons} \end{cases} \tag{65}$$

Bibliography I

Alpaydin, E. (2010), *Introduction to Machine Learning*, second edn, The MIT Press, Cambridge Massachusetts.

Duda, R. O., Hart, P. E. & Stork, D. G. (2001), *Pattern Classification*, Second edn, John Wiley and Sons.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press.

Hastie, T., Tibshirani, R. & Friedman, J. (2001), *The Elements of Statistical Learning - Data Mining, Inference and Prediction*, Springer Science+Business Media LLC.

Haykin, S. (2009), *Neural Networks and Learning Machines*, Third edn, Pearson Education.

Izenman, A. J. (2008), *Modern Multivariate Statistical Techniques - Regression, Classification and Manifold Learning*, Springer Science+Business Media LLC.

Kellleher, J. D., Namee, B. M. & D'Arcy, A. (2015), *Fundamentals of Machine Learning for Predictive Data Analytics - Algorithms, Worked Examples and Case Studies*, The MIT Press, Cambridge Massachusetts.

Lemarechal, C. (2012), 'Cauchy and the gradient method', *Documenta Mathematica* **Extra Volume ISMP**, 251–254.

Mhaskar, H. N. & Micchelli, C. A. (1994), How to choose an activation function, *in* J. D. Cowan, G. Tesauro & J. Alspector, eds, 'Advances in Neural Information Processing Systems 6', Morgan-Kaufmann, pp. 319–326.
URL: *http://papers.nips.cc/paper/874-how-to-choose-an-activation-function.pdf*

Mitchell, T. M. (1997), *Machine Learning*, WCB McGraw-Hill.

Bibliography II

Mohri, M., Rostamizadeh, A. & Talwalkar, A. (2012), *Foundations of Machine Learning*, MIT Press.

Nair, V. & Hinton, G. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Proceedings of 27th International Conference on Machine Learning', Haifa, Israel.

Ramachandran, P., Zoph, B. & Le, Q. V. (2017), Searching for activation functions, Technical Report arXiv:1710.05941v2 [cs.NE], ArXiV.
URL: *https://arxiv.org/pdf/1710.05941.pdf*

Webb, A. (2002), *Statistical Pattern Recognition*, Second edn, John Wiley and Sons.