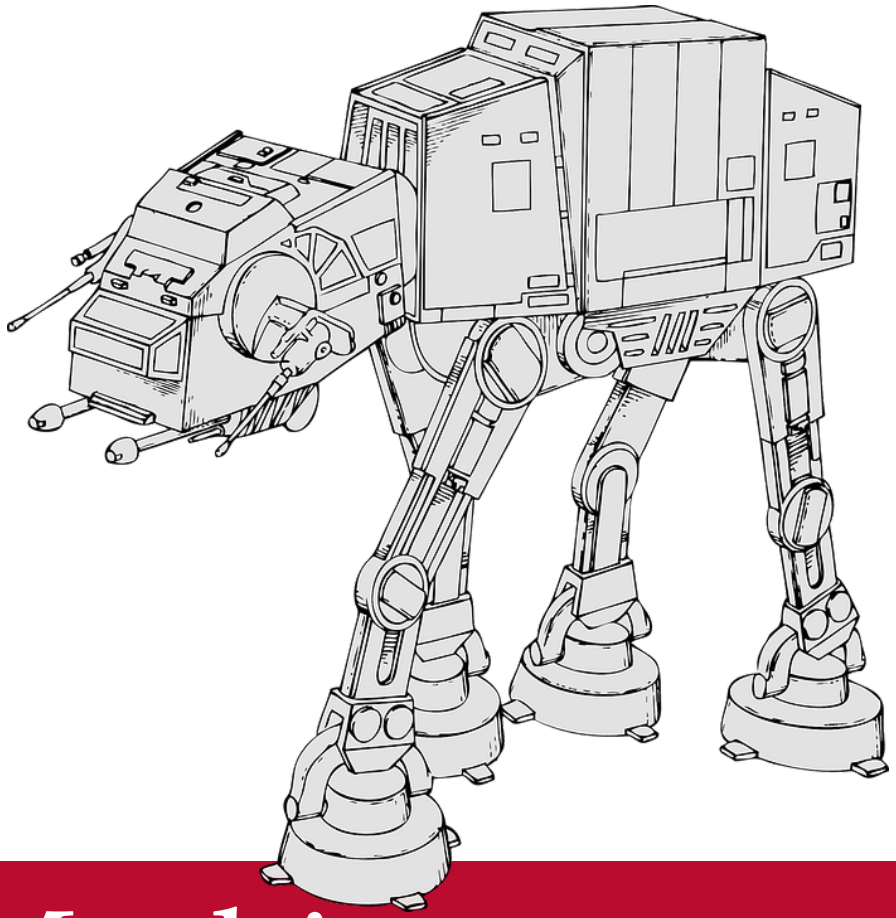
A Machine Learning Study Guide



Machine Learning Handbook

The Definitive Guide

ICS5110, class of 2018/9

L-Università ta' Malta

JEAN-PAUL EBEJER, DYLAN SEYCHELL, LARA MARIE DEMAJO, DANIEL FARRUGIA, KEITH MINTOFF, FRANCO CASSAR MANGHI, DAVID FARRUGIA, IVAN SALOMONE, ANDREW CACHIA, JAKE J. DALLI, JOSEPH AZZOPARDI, NATALIA MALLIA, MARK MUSCAT, STEFAN CASSAR, PATRICK BEZZINA, DYLAN VASSALLO, BRIAN PACE, GEORGE EDUARDO BUCKUP SULZBECK, KATRIN JANSEN, MATTHEW FARRUGIA ADD YOUR NAME TO THIS LIST

First printing, January 2019

Contents

Introduction

This book explains popular Machine Learning terms. We focus to explain each term comprehensively, through the use of examples and diagrams. The description of each term is written by a student sitting in for ICS5110 APPLIED MACHINE LEARNING[1] at the University of Malta (class 2018/2019). This study-unit is part of the MSc. in AI offered by the Department of Artificial Intelligence, Faculty of ICT.

[1] https://www.um.edu.mt/courses/studyunit/ICS5110

Activation Functions

? defined artificial neural networks as "a model that would imitate the function of the human brain—a set of neurons joined together by a set of connections. Neurons, in this context, are composed of a weighted sum of their inputs followed by a nonlinear function, which is also known as an activation function."

Activation functions are used in artificial neural networks to determine whether the output of the neuron should be considered further or ignored. If the activation function chooses to continue considering the output of a neuron, we say that the neuron has been activated. The output of the activation function is what is passed on to the subsequent layer in a multilayer neural network. To determine whether a neuron should be activated, the activation function takes the output of a neuron and transforms it into a value commonly bound to a specific range, typically from 0 to 1 or -1 to 1 depending on the which activation function is applied.

Step Function

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{1}$$

$$\frac{d}{d(x)} f(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases} \tag{2}$$

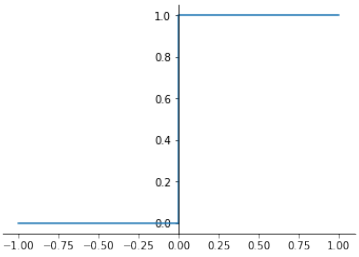


Figure 1: A graph of the step function.

The Heavside step function, visualised in figure 1 and defined by equation 1, is one of the simplest activation functions that can be used in a neural network. This function returns 0 if the input of a node is less than a predetermined threshold (typically 0), or otherwise it returns 1 if the output of the node is greater than or equal to the threshold. This activation function was first used in a machine learning context by **?** in his seminal work describing the perceptron, the precursor to the modern day neural network.

Nowadays, the step function is seldom used in practice as it cannot be used to classify more than one class. Furthermore, since the derivative of this function is 0 , as defined by equation 2, gradient descent algorithms are not be able to progressively update the weights of a network that makes use of this function (**?**).

Linear Functions

$$f(x) = ax + b \tag{3}$$

$$\frac{d}{d(x)} f(x) = a \tag{4}$$

A linear activation function, is any function in the format of equation 3, where $a, b \in \mathbb{R}$. This function seeks to solve some of the shortcomings of the step function. The output produced by a linear activation function is proportional to the input. This property means that linear activation functions can be used for multi-class problems. However, linear functions can only be utilised on problems that are linearly separable and can also run into problems with gradient descent algorithms, as the derivative of a linear function is a constant, as seen in equation 4. Additionally, since the output of the linear function is not bound to any range, it could be susceptible to a common problem when training deep neural networks called the exploding gradient problem, which can make learning unstable (**?**).

Sigmoid Function

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{5}$$

$$\frac{d}{d(x)} f(x) = f(x)(1 - f(x)) \tag{6}$$

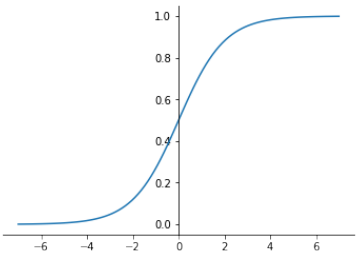


Figure 2: A graph of the sigmoid function.

The sigmoid function or logistic function, visualised in figure 2 and represented by equation 5, is one of the most commonly used activation functions in neural networks, because of its simplicity and desirable properties. The use of this function in neural networks was first introduced by **?**, in one of the most important papers in the field of machine learning, which described the back-propagation algorithm and the introduction of hidden layers, giving rise to modern day neural networks. The values produced by the sigmoid function are bound between 0 and 1, both not inclusive, which help manage the exploding gradient problem. The derivative of this function, represented by equation 6, produces a very steep gradient for a relatively small range of values, typically in the range of -2 to 2. This means that for most inputs that the function receives it will return values that are very close to either 0 or 1.

On the other hand, this last property makes the sigmoid function very susceptible to the vanishing gradient problem (**?**). When observing the shape of the sigmoid function we see that towards the ends of the curve, the function becomes very unresponsive to changes in the input. In other words, the gradient of the function for large inputs becomes very close to 0. This can become very problematic for neural networks that are very deep in design, such as recurrent neural networks (RNNs).To address this problems in RNNs Long Short-Term Memory (LSTM) units where introduced as a variant of the traditional RNN architecture (**?**).

Hyperbolic Tangent

$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{7}$$

$$\frac{d}{d(x)} f(x) = 1 - f(x)^2. \tag{8}$$

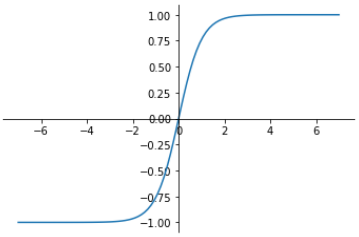


Figure 3: A graph of the hyperbolic tangent (tanh) function.

The hyperbolic tangent (tanh) function, visualised in figure 3 and represented by equation 7, is another common activation function that is sometimes used instead of sigmoid. The tanh function has the same characteristics of the sigmoid function mentioned above. In fact, when comparing figure 2 to figure 3 one can observe that the tanh function is simply a scaled and translated version of the sigmoid function. As a result of this scaling and translation, the tanh function has a steeper gradient towards the origin, and it returns values between -1 and 1. The derivative of the hyperbolic tangent function is represented by equation 8.

? analysed various factors that affect the performance of backpropagation, and suggested that tanh may be better suited than sigmoid as an activation function due to its symmetry about the origin, which is more likely to produce outputs that are on average close to zero, resulting in sparser activations. This means that not all nodes in the network need to be computed, leading to better performance. **?** studied in detail the effects of the sigmoid and tanh activation functions and noted how the sigmoid function in particular is not well suited for deep networks with random initialisation and go on to propose an alternative normalised initialisation scheme which produced better performance in their experiments.

Rectified Linear Unit

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{9}$$

$$\frac{d}{d(x)} f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{10}$$

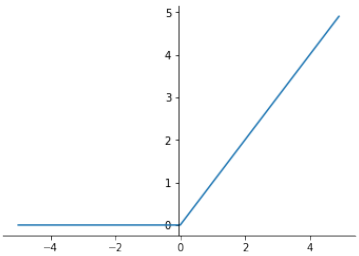


Figure 4: A graph of the ReLU function.

The Rectified Linear Unit (ReLU) function, visualised in figure 4 and represented by equation 9, returns 0 if the input of the function is negative, otherwise it outputs the value of the input itself. This function is non-linear in nature even though at first glance it may seem similar to an identity function. The ReLU function is becoming one of the more commonly used activation functions due to its simplicity, performance, and suitability to networks with many layers. Another benefit of the ReLU function is that it produces sparse activations unlike many other commonly used functions such as the sigmoid.

The ReLU function has been used in many neural network models to improve their performance. **?** use ReLU to improve the performance of Restricted Boltzmann Machines in object recognition. **?** introduced a breakthrough Convolutional Neural Network (CNN)

architecture called AlexNet, which pioneered the use of the ReLU activation function together with dropout layers to minimise over fitting in CNNs.

Unfortunately, because the gradient of the function for inputs that are negative is 0, as seen in equation 10, the ReLU function can still be susceptible to the vanishing gradient problem. To manage this problem a variant of the ReLU function, called Leaky ReLU is sometimes used. Rather than simply returning 0 for negative inputs, the leaky ReLU returns a very small value such as $0.01x$. **?** compared the performance of Sigmoid, ReLU and Leaky ReLU functions and found that while the the performance of both the ReLU and Leaky ReLU functions was better than the performance achieved with the sigmoid function, the performance of the two ReLU functions was nearly identical.

Bayesian Models in Machine Learning

Many machine learning techniques employ probability theory to make the best decisions given some data. Probabilistic approaches are often also summarised under the term “Bayesian approaches" (**?**). The following chapter will provide an overview of the underlying theorem as well as the most popular classifier based on said theorem.

Bayes' Theorem

Bayes' Theorem describes how to calculate the so-called posterior probability of an event A given our observed evidence B, that is $P(A|B)$. It is derived from the definition of conditional probabilities (11). The probability of an event B to occur given that the event A already has occurred is defined as the probability of the intersection of the two events A and B divided by the probability of the event A (**?**):

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \tag{11}$$

The probability of the intersection of the two events A and B can be rewritten as (12). However, since $P(A \cap B)$ is equal to $P(B \cap A)$, it can be rewritten as (13), as well (**?**).

$$P(A \cap B) = P(A)P(B|A) \tag{12}$$

$$P(B \cap A) = P(B)P(A|B) \tag{13}$$

Consequently, Bayes' Theorem can be derived as follows:

$$\begin{aligned} P(B|A) &= \frac{P(A \cap B)}{P(A)} = \frac{P(B \cap A)}{P(A)} = \frac{P(B)P(A|B)}{P(A)} \\ \Leftrightarrow P(A|B) &= \frac{P(B|A)P(A)}{P(B)} \end{aligned} \tag{14}$$

$P(A|B)$ is the aforementioned posterior probability, $P(B|A)$ the likelihood of event B given event A, $P(A)$ the prior probability of event A, and $P(B)$ the so-called marginal, that is the likelihood of an independent occurrence of event B (**??**).

Naïve Bayes Classification Algorithm

Intuitively, Bayes' Theorem can be used to calculate the probability of an hypothesis being true given some observed evidence (**???**) - it is a structured way of integrating all available information into an estimation of the truth of that hypothesis. However, it can also be used as a classification method (**????**): The instance to be classified (such as an email, a web page, or a newspaper article) is taken to be the "observed evidence" for that instance to belong to a certain class. For example, the content of an email is taken to be the information available to classify that email as spam or ham. Bayes' Theorem is applied to the conditional probability $P(c|d)$ of a given document d belonging to a certain class c, resulting in (15). The denominator $P(d)$ can be dropped, since the prior probability of a document to occur is the same for all possible classes and does not have any effect on the final classification decision (**?**). Ultimately, the result is (16).

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{15}$$

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \tag{16}$$

(16) can be read as the probability of a given document d belonging to a certain class c being proportional to the prior probability of a document occurring in class c multiplied by the probability of that class c having generated a document consisting of all the terms t_k in d. Naïve Bayes classifiers are also called generative classifiers (**?**): Given a specific observation d, they return the class c that is most likely to have generated said observation - also called the maximum a posteriori (MAP) class.

The two parameters needed to calculate $P(c|d)$, $P(c)$ and $P(t_k|c)$, are obtained via approximations from the training set, based on maximum likelihood estimations (MLE) (**??**). For the prior probability $P(c)$, the estimate is as follows:

$$\hat{P}(c) = \frac{N_c}{N}, \tag{17}$$

which is the number of documents in the training set belonging to class c divided by the total amount of documents in the training set (**??**). Similarly, the probability of a term t_k occurring in a document of class c is obtained like this:

$$\hat{P}(t_k|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}, \tag{18}$$

which is the number of times a term T appears in documents of class c divided by the number of times that term appears in the vocabulary V, i.e. in all documents of the training set (**??**).

These approximations pose two problems. First of all, the multiplication of all the conditional probabilities $P(t_k|c)$ may result in a

so-called floating-point underflow (**?**). In order to avoid this, Naïve Bayes classifiers usually do not calculate the product over $P(t_k|c)$, but the sum over the log values of $P(t_k|c)$ (see (19)). Using the log values instead of the original ones does not interfere with the final classification decision in any way: The class with the highest log probability is still the one most likely to have generated the given document.

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} \log P(t_k|c) \tag{19}$$

The second problem encountered by the estimations above is data sparseness, resulting in terms not occurring in documents of particular classes in the training set. The preferred solution is to eliminate the resulting zeros by using add-one or Laplace-smoothing (see (20), with $B' = |V|$, the length of the vocabulary), assuming that each and every relevant term occurs at least once in a corpus (**?**).

$$\hat{P}(t_k|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B'} \tag{20}$$

Naïve Assumptions

While the calculations above appear to be straightforward, they pose a problem in terms of their feasibility, especially regarding big sets of data. Relating individual probabilities to all terms of a document – even to those occurring multiple times in different positions – increases the number of parameters dramatically and requires huge sets of training data, which are almost never available. Naïve Bayes offers solutions to this problem via two rather naïve assumptions giving the classifier its name (**??**).

Hence, Naïve Bayes classifiers assume the bag of words model for all terms in a document, that is they assume the probability of a term to be the same, regardless of its position in the document. This is also called the positional independence assumption (**??**).

The second assumption is called the conditional independence assumption. It simplifies the calculation further by proposing that all terms in a document are independent of one another, i.e. the occurrence of a specific term does not make it any more likely for another, possibly related term to appear in the same document. Obviously, the exact opposite is true - in reality, the terms in a document are indeed dependent on one another. However, Naïve Bayes classifiers still offer accurate classification decisions, regardless of the inaccuracies in their probability estimations (**??**).

Conclusion

The classification method described just now is just one of the versions of Bayesian classifiers, called the multinomial Naïve Bayes classifier.

One alternative is the Bernoulli model (**?**). In order to arrive at a classification decision, the Bernoulli model uses only binary information regarding the occurrence (1) or absence (0) of a term in a document. This often leads to errors regarding the classification of long documents, since the actual number of occurrences of a term are not taken into account, which is why the multinomial model is preferred.

Concept Drift

Nowadays, huge amounts of data are being generated every day. Such data is providing insights on patterns and its importance is increasing over time, thus making it impossible to collect and process data manually. Moreover, simple data analysis is not sufficient to make educated decisions as the most important knowledge is hidden within the captured data.

Classification techniques are within the most utilised algorithms in order to extract hidden patterns. One major limitation of such classification models is the assumption that the underlying data concepts will not change over time. This assumption poses a major classification limitation as in the real-world observations and classes do change over time. A real-world example is a spam filter where spammers continuously find new ways of how to send spam to increase their success rate. Thus, a classification technique will decrease it accuracy over time as the observations, statistics and their classifications will not hold forever. This problem is referred to as concept drift. This increases machine learning model's complexity, mostly in those cases where new data is treated as an important contributor to the final concept classification (**??**).

Moreover, concept drift refers to the changes within the learned structure which occur over time. These changes could result into misclassifications and also includes long term and short-term fluctuations may disrupt the underlying classification probability.

Types of concept drift

In literature one finds five basic types of concept drift which are:

1. Sudden drift: This kind of concept drift, happens in those cases where the concept drift happens abruptly. An example of such drift is season changes in sales (**?**).

2. Incremental: The drift happens whenever variables change their value over time. An example of this type of drift is the effect of prices due to inflation where prices will keep rising overtime (**?**).

3. Gradual drift: Although its similar to incremental, a gradual drift refers to those cases where over time the concept changes definition. Example in a fraud detection, a fraudulent may transition from one category to another and back (**?**).

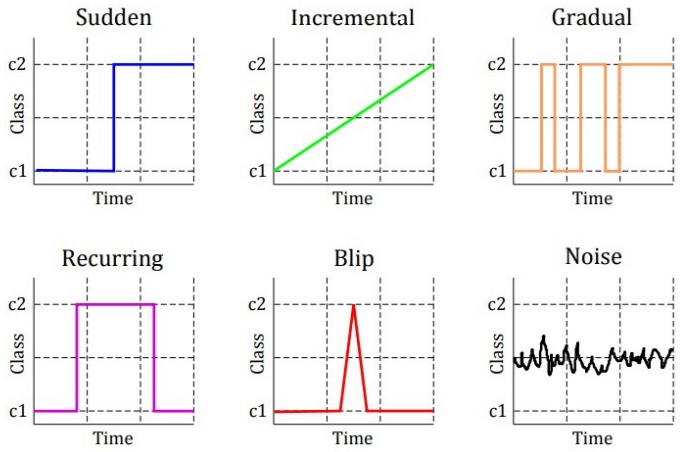


Figure 5: Different types of concept drift excluding noise. (**?**)

4. Recurring drift: This refers to those instances where the classification changes over time and alternates between the classes. A real-world example is the usage of a particular mobile application where a user may use the application differently between work and home (**?**).

5. Blip drift: Is an abrupt drift, which only lasts for a very short duration. There has been number of debates over blip if it qualifies as noise or is ignore or the model is adjusted to correctly classify the blip drift. Nonetheless, adjusting the model may harm future classifications as following the blip the old model must be restored. An example of blip drift is Cyber Monday where on-line shops might experience an increase in their sales or page hits which is an abnormal pattern (**?**).

6. Noise: This refers to insignificant fluctuations which are not connected or related to the target concept. If noise is present, it must be filtered out as it's not a concept drift and might conflict with the classification model (**?**).

Detecting Concept drift

The aim of drift detectors is to raise alarms when concept drift occurs in order to update or rebuild the model. This is a crucial step in models as it increases the robustness and model accuracy. Two simple drift detectors are the Cumulated Sum (CUSUM) (**?**) and the Geometric Moving Average (GMA) (**?**). The main difference between these detectors is that CUSUM raises an alarm when the sum of the inputted data is not zero whilst GMA considers the weighted average in a window and raised the alarm if the average is higher than a given threshold.

Dealing with concept drift

There exist a number of approaches to handle concept drift which mainly are categorized in three approached. Moreover, the aim of these approached is to allow the model to be dynamically updated and adapted to drift without the need of rebuilding which is a costly operation.

1. Instance selection: Is the most common technique for handling concept drift and its aim is to identify those instances that are truly related to the current concept. This is achieved by creating a form of a moving window which considers recent information whilst utilising learnt prediction concepts in the immediate future. Moreover, due to the fact that the classification will be utilising recent data, the model will be automatically excluding old data (**?**).

2. Instance weighting: In this technique instances are weighted depending on their age and relevance to the concept. This allows gradual drift to start being properly handled by the model as new data would have more weight whilst past data would start losing its classification power (**?**).

3. Ensemble learning: allows one to obtain better classification results by processing the data through multiple algorithms. The resultant of each algorithm is combined and results would be considered in the classification decision. Moreover, one could also introduce weights, which weighting is assigned to the different algorithm based on their classification power and accuracy. One downside of ensemble learning is that the data needs to be processed by every algorithm selected for the model classification (**?**).

Conclusions

In this chapter, we introduced and discussed concept drift. Machine learning is increasing its popularity and more real-world application are being released like self-driving cars and fraud detection, which applications do suffer from concept drift. This is due to the fact that data in the real-world is not static, thus concept drift will be residing in such applications and must be dealt with. Moreover, one major limitation in concept drift, is the lack of real-data during the experimentation and testing. This is due to the fact that drift data has to be generated artificially, thus, limiting the experiment efficiency. Example, even though Tesla did rigorous testing and were given the go-ahead of releasing self-driving cars, a number of deadly crashed emerged. Thus, concept drift is an important aspect in machine learning as one needs to enable the model to adapt to new concepts or raise alarms if drastic shifts are encountered (**?**).

Confusion Matrix

A *confusion matrix* (CM), is a contingency table showing how well a model classifies categorical data. By convention (**?**), the CM of an N-class model is an N×N matrix indexed by the true class in the row dimension and the predicted class in the column dimension (Table 1).

		Predicted Class	
		spam	$\neg$*spam*
True Class	*spam*	10	1
	$\neg$*spam*	2	100

Table 1: CM of a hypothetical binary classifier which predicts whether out-of-sample text objects are spam or not. In this example, 10 spam and 100 non-spam objects are classified correctly, whilst 1 spam and 2 non-spam objects are misclassified.

Even though CMs are commonly used to evaluate binary classifiers, they are not restricted to 2-class models (**?**). A CM of a multi-class model would show the number of times the classes were predicted correctly and which classes were confused with each other (Table 2).

	M&M's	*Skittles*	*Smarties*
M&M's	34	3	8
Skittles	1	28	5
Smarties	2	4	22

Table 2: CM of a hypothetical sweets classifier. The main diagonal of the CM shows the number of correct predictions, whilst the remaining elements indicate how many sweets were misclassified.

The CM of the model $h : X \mapsto C$ over the concept $c : X \mapsto C$ using dataset $S \subset X$ is formally defined as a matrix Ξ such that $\Xi_{c,S}(h)[d_1, d_2] = |S_{h=d_1, c=d_2}|$ (**?**). The CM is constructed by incrementing the element corresponding to the true class *vis-a-vis* the predicted class for each object in the dataset (Algorithm 1).

$\Xi \leftarrow 0$
for $x \in S$ **do**
  $d_1 \leftarrow c(x)$
  $d_2 \leftarrow h(x)$
  $\Xi_{d_1,d_2} \leftarrow \Xi_{d_1,d_2} + 1$

Algorithm 1: The CM is initialised to the zero matrix, and populated by iterating over all the objects x with corresponding true class d_1 and predicted class d_2 and incrementing the element (d_1, d_2) by 1 for each matching outcome.

In binary classification, the CM consists of 2 specially designated classes called the *positive* class and the *negative* class (**?**). As indicated in Table 3, positive outcomes from the true class which are classified correctly are called *true positives* (TP), whilst misclassifications are called *false negatives* (FN). On the other hand, negative true class outcomes which are classified correctly are called *true negatives* (TN),

and misclassifications are called *false positives* (FP). In natural sciences, FP are called *Type I* errors and FN are known as *Type II* errors (**?**).

	+ve	*-ve*
+ve	TP	FN
-ve	FP	TN

Table 3: CMs of binary classifiers have positive (+ve) and negative (-ve) classes, and elements called *true positives* (TP), *false positives* (FP), *true negatives* (TN) and *false negatives* (FN).

The information presented in the CM can be used to evaluate the performance of different binary classifiers (**?**). A number of statistics (Equations 21-27) derived from the CM have been proposed in the literature (**?**) to gain a better understanding of what are the strengths and weaknesses of different classifiers. Caution should be exercised when interpreting metrics (**?**), since the CM could be misleading if the data is imbalanced and an important subrange of the domain is underrepresented (**?**). For instance, an albino zebra classifier which always returns negative will achieve high accuracy since albinism is a rare disorder.

These metrics are important in situations in which a particular type of misclassification, i.e. FP or FN, could have worse consequences than the other (**?**). For example, FP are more tolerable than FN in classifiers which predict whether a patient has a disease. Both outcomes are undesirable, but in medical applications it is better to err on the side of caution since FN could be fatal.

Accuracy (ACC) is the proportion of correct predictions (Equation 21). It is a class-insensitive metric because it can give a high rating to a model which classifies majority class objects correctly but misclassifies interesting minority class objects (**?**). The other metrics should be preferred since they are more class-sensitive and give better indicators when the dataset is imbalanced.

$$ACC = \frac{|TP \cup TN|}{|TP \cup FP \cup TN \cup FN|} \tag{21}$$

Negative predictive value (NPV) is the ratio of the correct negative predictions from the total negative predictions (Equation 22).

$$NPV = \frac{|TN|}{|TN \cup FN|} \tag{22}$$

True negative rate (TNR), or *specificity*, is the ratio of the correct negative predictions from the total true negatives (Equation 23).

$$TNR = \frac{|TN|}{|TN \cup FP|} \tag{23}$$

True positive rate (TPR), also called *sensitivity* or *recall*, is the ratio of the correct positive predictions from the total true positives (Equation 24).

$$TPR = \frac{|TP|}{|TP \cup FN|} \tag{24}$$

Sensitivity and specificity can be combined into a single metric (Equation 25). These metrics are often used in domains in which

minority classes are important (**?**). For example, the sensitivity of a medical classifier (**?**) measures how many patients with the condition tested positive, and specificity measures how many did not have the condition and tested negative.

$$Sensitivity \times Specificity = \frac{|TP| \times |TN|}{|TP \cup FN| \times |TN \cup FP|} \tag{25}$$

Positive predictive value (PPV), or *precision*, is the ratio of the correct positive predictions from the total positive predictions (Equation 26). The difference between accuracy and precision is depicted in Figure 6.

$$PPV = \frac{|TP|}{|TP \cup FP|} \tag{26}$$

Precision and recall are borrowed from the discipline of *information extraction* (**?**). A composite metric called *F-score*, *F1-score*, or *F-measure* (Equation 27) can be derived by finding their harmonic mean (**?**).

$$F\text{-}score = 2 \times \frac{PPV \times TPR}{PPV + TPR} \tag{27}$$

The complements of ACC, NPV, TNR, TPR and PPV are called, respectively, *error rate*, *false omission rate*, *false positive rate*, *false negative rate* and *false discovery rate*.

The metrics can be adapted for evaluating multi-class models by decomposing an N-class CM into 2-class CMs, and evaluating them individually (**?**). The literature describes two methods for decomposing this kind of CM. In the *1-vs-1* approach, 2-class CMs are constructed for each pairwise class as shown in Table 4.

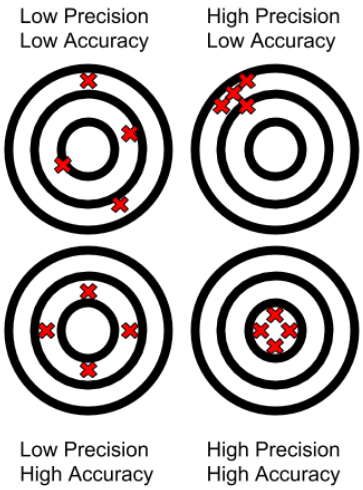


Figure 6: Accuracy vs Precision.

+ve	*-ve*
M&M's	{Skittles, Smarties}
Skittles	{M&M's, Smarties}
Smarties	{M&M's, Skittles}

Table 4: 2-class CMs derived from the classes in Table 2. The +ve classes are paired separately with each -ve class.

In the *1-vs-rest* approach, 2-class CMs are constructed for each class and the remaining classes combined together as shown in Table 5.

+ve	-ve
M&M's	Skittles ∪ Smarties
Skittles	M&M's ∪ Smarties
Smarties	Skittles ∪ M&M's

Table 5: 2-class CMs derived through decomposition of the 3-class CM from Table 2 using the 1-vs-rest approach.

Using all metrics could be counterproductive due to information redundancy, but none of the metrics is enough on its own (**?**). For instance, recall is class-sensitive but it would give a perfect score to an inept model which simply returns the positive class. Thus, the best approach is to evaluate with complementary pairs (**?**) such as sensitivity *vs* specificity, or precision *vs* recall; or a combined measure such as the F-score.

Taking into account the above, CMs are suitable for visualising, evaluating, and comparing the performance of binary or multi-class

classifiers. They should be used in conjunction with metrics such as the F-measure to avoid bias, especially if the dataset is unbalanced. For further details on the theoretical aspects of CMs and for practical examples in R refer to (**?**); for examples in Python refer to (**?**).

The following example is motivated by the samples in the *Scikit-Learn* documentation and the work of (**?**). The models in Figure 7 were trained on the *wines* dataset included with Scikit-Learn.

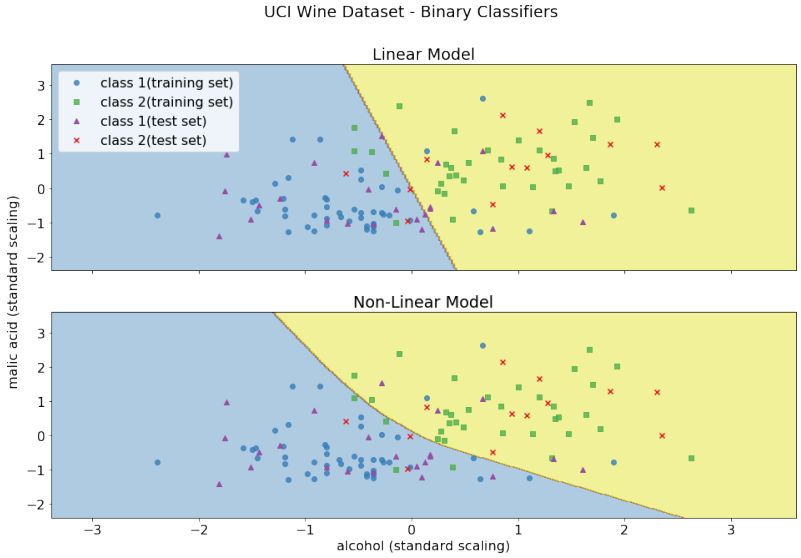


Figure 7: Decision boundary learned by a linear and non-linear binary classifier.

As it can be deduced from Figure 7, the decision boundary of the non-linear model is a better fit than the linear model. The CMs in Figure 8 also show that non-linear model performs better with a higher TP, and consequently lower TN. The biggest advantage of the non-linear model is the higher sensitivity resulting in a better F-score.

	Linear	Non-Linear
Accuracy	0.72	0.78
Specificity	0.77	0.77
Sensitivity	0.70	0.78
Precision	0.84	0.86
F-score	0.76	0.82

Table 6: Statistics derived from the CMs in Figure 8.

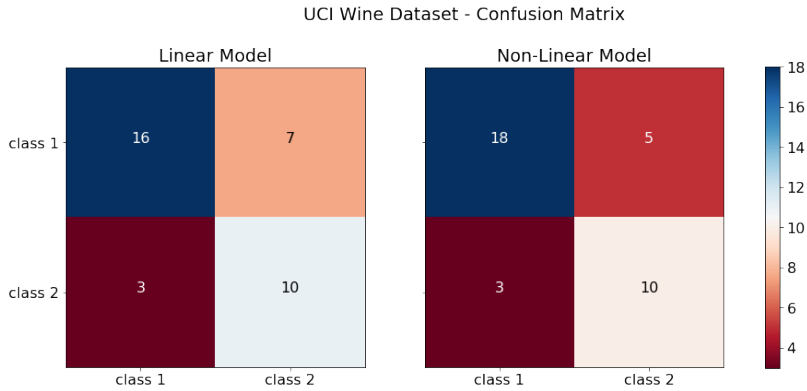


Figure 8: The linear classifier has 16 TP, 10 TN, 7 FN and 3 FP, whilst the non-linear classifier has 18 TP, 10 TN, 5 FN and 3 FP.

Convolution

Convolution in Mathematics

Definition

Convolution is a mathematical operation on two functions to produce a result that reflects how one of the input functions is modified by the other input function.
The convolution of functions $f(x)$ and $g(x)$ (denoted $f * g(x)$) is defined (**?**), for continuous functions f and g, as[2]

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{28}$$

[2] In order for the continuous convolution operator $f * g$ to be defined, both f and g must be integrable functions.

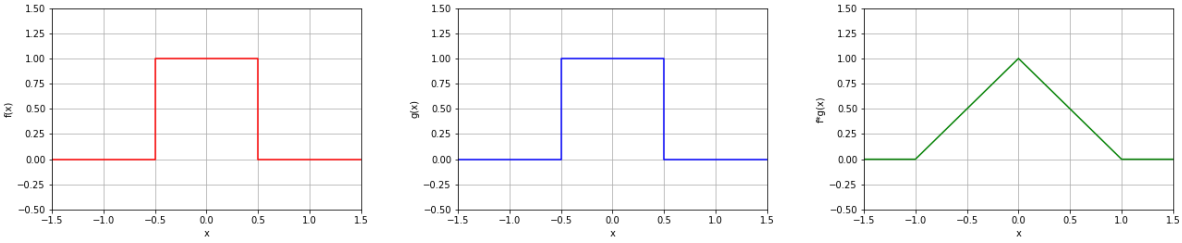


Figure 9: These plots show two continuous functions ($f(x)$ and $g(x)$) and their convolution $f * g(x)$

and, for discrete functions $f(n)$ and $g(n)$, as

$$(f * g)(n) \equiv \sum_{m=-\infty}^{\infty} f(m)g(n-m) \tag{29}$$

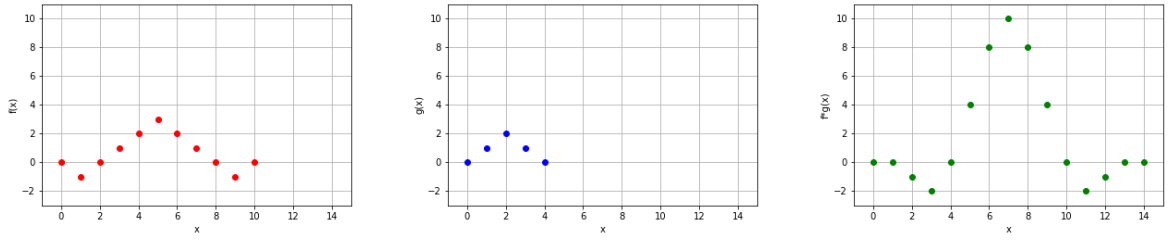


Figure 10: These plots show two discrete functions ($f(n)$ and $g(n)$) and their convolution $f * g(n)$

Properties

Since convolution is defined as a product of integrable functions on the linear space, the following algebraic properties are satisfied (**?**):

Commutativity

$$(f * g) = (g * f) \tag{30}$$

Associativity

$$(f * (g * h)) = ((f * g) * h) \tag{31}$$

Distributivity

$$(f * (g + h)) = (f * g) + (f * h) \tag{32}$$

Multiplication by a scalar value

$$a(f * g) = (af) * g \tag{33}$$

Multiplicative identity, where δ denotes the delta distribution

$$f * \delta = f \tag{34}$$

Differentiation

$$\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx} \tag{35}$$

Integration

$$\int_{R^d}(f * g)(x)dx = \left(\int_{R^d} f(x)dx\right)\left(\int_{R^d} g(x)dx\right) \tag{36}$$

Applications

Listed below are some of the main applications of the convolution operator in various fields of knowledge (**?**):

- **Image Processing** - Different operations are performed on images, in which the original image (larger matrix) and the filter to be applied (smaller matrix, also known as 2D kernel) are treated as 2-dimensional arrays. The kernel size and the values of its elements determine the effect on the original image
- **Signal Filtering** - Provided the filter function is the same as the impulse response function used in in signal filtering, the two operations are equivalent
- As a handy tool for **Polynomial Multiplication** - If we consider two polynomials being multiplied, we can use a convolution process to obtain the coefficients of the resulting polynomial
- **Audio Processing** - Reverberation is a desired effect in auditoriums, music halls, cinemas, and similar constructions. Convolution is used to digitally simulate reverberation in such structures, providing architects with information about the acoustic quality of a building prior to its construction
- **Artificial Intelligence** - Convolutional Neural Networks use convolution in one or more of its internal layers in order to process input data and enhance specific features.
- **Probability Theory** - The Probability Density Function (PDF) of the sum of two independent random variables can be obtained by the convolution of the PDFs of the two variables

Convolution in Neural Networks

Definition

Convolutional Neural Networks (CNNs) are a subclass of multi-layer neural networks in which some of its hidden layers are convolutional layers.
Like most neural networks, CNNs can be trained using back propagation algorithms and are mostly used to recognize visual patters with minimal or no preprocessing (**?**).

Origins and Evolution

CNNs were initially developed as an attempt to replicate the processing of sight in living organisms.
A seminal paper published in 1968 (**?**) noted that two types of neurons took part in the process of identifying images: simple cells (responsible for the detection of straight edges and their orientation), and complex cells (with larger receptive fields, but not affected by the exact position of the edges in the input image).
During the 1980s, CNNs evolved, followed by the introduction of Time Delay Neural Networks (TDNNs) in 1987 (**?**).
In the early 1990s, CNNs were modified and used for medical image processing and for automatic detection of breast cancer in mammograms (**?**).
In 1998 LeCun lead a team that created LeNet-5, a CNN used by banks to identify handwritten digits in checks.
With the introduction of Graphic Processing Units (GPUs), training of CNNs was greatly improved, allowing for the implementation of efficient Deep Learning Neural Networks with impressive results in image processing and other applications (**?**).

Layers

A typical CNN has sets of layers with specific functions, such as:

- **Convolutional layer** - This is the main feature of a CNN.
  In this layer, a set of filters (also known as kernels) is convoluted over the entire input data. The result of this operation (the activation map of the filter), extracts or enhances specific features in the input data that are passed to the following layers in the CNN.
  Since the result of each convolution operation is affected by only a small part of the input data (due to the reduced size of the filter if compared to the size of the input data), this can be interpreted as each point of the activation map being affect by only a small subset of input points.

- **Pooling layer** - In this layer, simple operations are applied in order to reduce the size of the input data, such as maximum pooling or average pooling.

In 2x2 average pooling, for example, each result point the the average of 4 input points, reducing the size of the input field by a factor of 4.

- **Rectified Linear Unit (ReLU) layer** - This layer applies the $f(x) = max(0, x)$ activation function to its input, removing negative values from the input field.
- **Fully Connected layer** - In this layer, usually applied after several convolutional and pooling layers, neurons are connected to all activations in the previous layer, as in regular neural networks.
- **Loss layer** - This is the last layer in a typical CNN. In this layer, training is based on the divergence between the desired and predicted labels.

Training

One important feature of CNNs is that, with the exception of the Loss layer, all other layers (Convolutional, Pooling, ReLU, and Fully Connected layers) can be trained using unsupervised training (**?**).
Also, usually each layer or set of layers is trained individually, which also improves the speed and the overall result of the training.
With the introduction of Graphic Processing Units (GPUs), speed of training and processing in the Convolutional layer has been greatly improved (**?**)

Applications

CNNs are currently applied to a wide range of areas, such as (**?**):

- **Image recognition and Video analysis** - Image Recognition Systems using CNNs are very efficient, specially if applied to facial recognition. Very good results have been obtained in challenges, and in the ImageNetNet tests performed almost as good as humans. If compared to image recognition, video analysis adds a level of complexity, since, in addition to the analysis of each frame, a time dimension needs to be added to the problem and convolution needs to be performed both on each image as well as on the time domain.
- **Game strategy** - A number of board games have been implements using CNNs, most notably Checkers, in Chess and in Go, the first time Artificial Intelligence beat an expert human player in this game.
- **Drug discovery** - In this area, CNNs are used to predict the behaviour of drug molecules and human proteins.
- **Natural language processing** - For NLP, CNNs are used extensively, including semantic parsing, search query retrieval, sentence modeling, classification, and prediction.