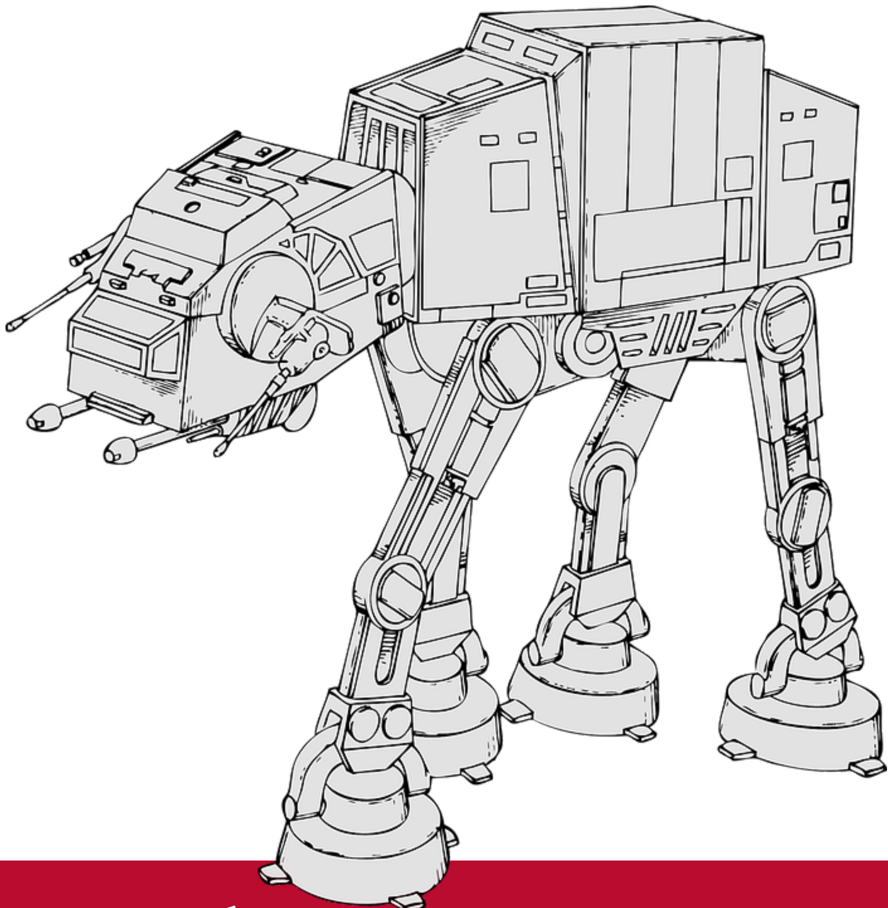
A Machine Learning Study Guide



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, CLAIRE GALEA 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

Caterini (2018) 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

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

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

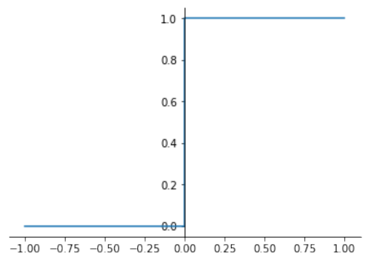


Figure 1: A graph of the step function.

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 Rosenblatt (1957) 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 (Snyman, 2005).

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 (Goodfellow et al., 2016).

Sigmoid Function

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

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

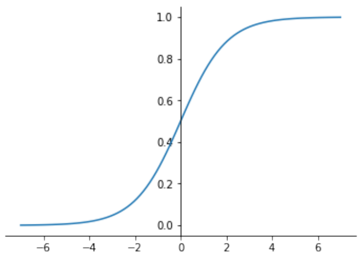


Figure 2: A graph of the sigmoid function.

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 Rumelhart et al. (1986), 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 (Bengio et al., 1994). 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 (Hochreiter and Schmidhuber, 1997).

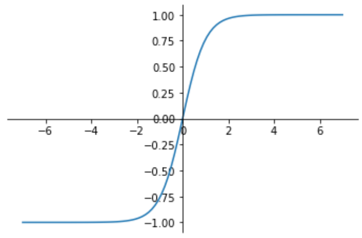


Figure 3: A graph of the hyperbolic tangent (tanh) function.

Hyperbolic Tangent

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

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

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.

LeCun et al. (2012) 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. Glorot and Bengio (2010) 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

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

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

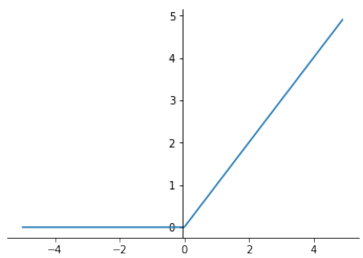


Figure 4: A graph of the ReLU function.

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. Nair and Hinton (2010) use ReLU to improve the performance of Restricted Boltzmann Machines in object recognition. Krizhevsky et al. (2012) 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 $0.01x$. Maas et al. (2013) 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.

Classification

Data mining can be described as "the procedure of extracting information from huge sets of data". Classification and *regression* are two mining techniques which are used to extract knowledge and perform predictions on the presented data (Sagar, 2017). The values predicted by these techniques may be either *quantitative* - involving discrete numerical values, or *qualitative* - with values within a particular class or label (James et al., 2006). Predicting marks that students will obtain in their final exams yields a discrete, continuous value. In this case, regression analysis is "used to predict missing or unavailable numerical data values" (Jiawei Han, Micheline Kamber, 2011). On the other hand, predicting the grade (A, B or C) that the students will obtain is considered as a classification problem, since the outcome value is a categorical class label (Jiawei Han, Micheline Kamber, 2011).

Classification is a data analysis technique in which models, also referred to as *classifiers*, are used to "predict categorical (discrete, unordered) class labels". Several scenarios in the real world may require classification models to predict several outcomes. For example, in the health sector, classification can be helpful in predicting whether a person is diagnosed with a particular medical condition or not, based on certain parameters related to experienced symptoms (Venkata Ramana et al., 2011; M. Alzahani et al., 2015). Financial sectors may find classifications helpful when it comes to predicting whether a company is bankrupt or in good financial health (Moradi et al., 2012). Moreover, they may apply classifiers to determine whether a bank should issue a loan to a customer (Thomas, 2000). In these scenarios, classifiers predict labels such as 'positive' or 'negative' for a medical condition, or 'bankrupt' or 'healthy' for the financial status of a company. Since values predicted by a classifier are discrete, any ordering amongst the values is meaningless.

Classification involves two main steps: training a model for classification, and then using the model to classify the data. When *supervised learning* is used for classification, the training stage involves analysing a training data consisting of samples which are labelled with a categorical class. These samples are used to train the model on how to perform classifications. Testing data, also made up of features and their corresponding class labels, is then used to predict the accuracy of the classifier, by comparing the predicted outcome to the actual class label in the testing samples (Neelamegam and Ramaraj, 2013). On the other hand, *unsupervised learning* builds classifiers for predictions using data which is unlabelled, i.e. the final class is not known. In this case, the model learns how to classify the data by observing the similarities or dissimilarities within the dataset (Jiawei Han, Micheline Kamber, 2011).

Binary Classification

When two classes are available as outputs of a classification problem, then this is called binary classification (Neelamegam and Ramaraj, 2013). Kolo (2011) defines *binary classifiers* as those which determine whether an input should classify as "either in the category or not in the category". This implies that an input which classifies as not part of a class should automatically be considered as a member of the other class.

For example, using a binary classifier to predict 'Pass' or 'Fail' in a subject, by providing it with the marks obtained in the exam, the classifier should determine whether the student 'passed' or 'did not pass' an exam. A 'Fail' label is assumed when the classifier determines that the student 'did not pass' the exam (Kolo, 2011). Decision Trees (Figure 5), Bayesian Networks and Logistic Regression are some of the methods used for binary classification.

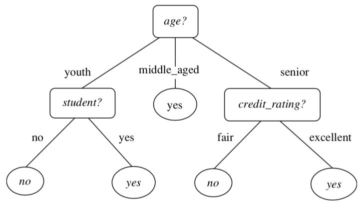


Figure 5: A binary Decision Tree example which determines whether a customer will purchase a computer or not. Reproduced from (Jiawei Han, Micheline Kamber, 2011)

Probabilistic Classification

Probabilistic classification can be considered as a subclass of classification, in which apart from assigning categorical classes to unseen data, probabilistic models "use statistical inference to find the best class for a given example" (Aggarwal, 2014). This implies that probabilistic models compute the probability that an unseen example belongs to the possible target classes, and then chooses the final target class by considering the highest probability obtained. One advantage of probabilistic models is that the prediction outcome is composed of both the target label, as well as the confidence with which this class label was obtained.

Initially, probabilistic classification involves the estimation of the posterior probability, $P(C_k \mid x)$, where C_k denotes the class and x refers to the input variables. This posterior probability estimate can be determined using a *generative model* or a *discriminative model*. Generative models work by obtaining the probability of each class, denoted by $P(C_k)$, as well as the probability of obtaining x when given a class, also known as the class-conditional probabilities, denoted by $P(x \mid C_k)$ (Aggarwal, 2014). Then, the posterior probability can be determined using Bayes Theorem, shown in equation 11, or by modelling the joint distribution over x and C_k as in equation 12.

$$P(C_k|x) = P(C_k) \cdot \frac{P(x|C_k)}{P(x)} \tag{11}$$

$$P(C_k|x) \propto P(C_k) \cdot P(x|C_k) \tag{12}$$

In these equations C_k denotes the class and x refers to the input variables. Hidden Markov Model and Naive Bayes classifier are two examples of generative models used in probabilistic classification. Discriminative models focus on determining a function that maps the input x directly onto the class C_k.

$$P(C_k|x) = \frac{1}{1 + e^{-\theta^T X}} \tag{13}$$

One popular discriminative model is the Logistic Regression Model, explained by equation 13, in which θ refers to the parameters to be estimated. Conditional Random Fields is another commonly used discriminative model.

Once the posterior probability is obtained, probabilistic classification then uses decision theory to deduce the class of an unseen example x (Aggarwal, 2014).

Multiclass Classification

Multiclass classification models are able to determine a class for an unseen example from k classes, where $k > 2$. A multiclass classification problem may be solved by extending the aforementioned binary classification algorithms, and others including *Neural Networks* and *Support Vector Machines* (Aly, 2005). For example, a Decision Tree algorithm can naturally handle multiclass classification problems. To predict an outcome using the tree model, the algorithm starts from the root node of the tree and repeatedly traverses branches based on the feature values, until a leaf node is reached (Rokach and Maimon, 2007). The value of the leaf node is considered as the predicted class label of the unseen example, and these values can refer to any of the k classes. Support Vector Machines are another example of classification algorithms which were initially built for binary classification, but eventually extended to be applied in multiclass problems. Extensions of this algorithm handle the separation of the various classes by amending the optimization problem (Aly, 2005).

Aly (2005) discusses approaching a multiclass classification problem by decomposing it into multiple binary classification problems, which can then be solved using binary classifiers.

One-versus-all (OVA)

This approach considers k binary classifiers, where k is the number of classes. Each classifier is trained on data composed of positive examples from the k^{th} class and negative examples from the remaining k-1 classes. Unseen examples are assigned the class of the classifier which obtains the highest output (Aly, 2005).

All-versus-all (AVA)

In this approach, $\frac{k(k-1)}{2}$ binary classifiers are built to distinguish between all the possible pairs of classes. When predicting, all classifiers are considered and the class obtaining the highest score is taken as the target class (Aly, 2005).

Error-Correcting Output-Coding (ECOC)

Using this approach for multiclass classification, n binary classifiers are trained to differentiate between K classes. A binary matrix M (Figure 6) is composed of K classes and the corresponding codewords with length N. Thus, binary classifiers are trained using the column values of M.

When predicting labels for unseen examples, a distance measure is used to compare codewords obtained from the N classifiers to the K codewords. The class label is determined by selecting the codeword having the smallest distance (Aly, 2005).

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
Class 1	0	0	0	0	0	0	0
Class 2	0	1	1	0	0	1	1
Class 3	0	1	1	1	1	0	0
Class 4	1	0	1	1	0	1	0
Class 5	1	1	0	1	0	0	1

Figure 6: ECOC Binary Matrix. Reproduced from (Aly, 2005)

Apart from these methods, *Generalized Coding* (L. Allwein et al., 2001) may also be applied. Finally, multiclass classification problems may be handled using *hierarchical classification*, in which classes are arranged in the form of a tree. Parent nodes of the trees are further split into child nodes, and a binary classifier is used to differentiate between the clusters pertaining to child classes. Eventually, the leaf node constitutes of a single class, thus the model can then be used to classify a new example (Aly, 2005). Binary Hierarchical Classifier (BHS) (Kumar et al., 2002) and Hierarchical SVM (HSVM) (Yangchi Chen et al., 2014) are two hierarchical classification approaches.

Confusion Matrix

A *confusion matrix* (CM), is a contingency table showing how well a model classifies categorical data. By convention (Sammut and Webb, 2017), 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	*¬spam*
True Class	*spam*	10	1
	¬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 (Martin and Jurafsky, 2018). 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 $\Xi_{c,S}(h)[d_1, d_2] = |S_{h=d_1,c=d_2}|$ (Cichosz, 2014). 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 (Saito and Rehmsmeier, 2015). 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 (Fielding and Bell, 1997).

	+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 (Lu et al., 2004). A number of statistics (Equations 14-20) derived from the CM have been proposed in the literature (Deng et al., 2016) to gain a better understanding of what are the strengths and weaknesses of different classifiers. Caution should be exercised when interpreting metrics (Jeni et al., 2013), since the CM could be misleading if the data is imbalanced and an important subrange of the domain is underrepresented (Raeder et al., 2012). 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 (Hassanien and Oliva, 2017). 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 14). 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 (Branco et al., 2016). The other metrics should be preferred since they are more class-sensitive and give better indicators when the dataset is imbalanced.

$$ACC = \frac{|TP \cup TN|}{|TP \cup FP \cup TN \cup FN|} \tag{14}$$

Negative predictive value (NPV) is the ratio of the correct negative predictions from the total negative predictions (Equation 15).

$$NPV = \frac{|TN|}{|TN \cup FN|} \tag{15}$$

True negative rate (TNR), or *specificity*, is the ratio of the correct negative predictions from the total true negatives (Equation 16).

$$TNR = \frac{|TN|}{|TN \cup FP|} \tag{16}$$

True positive rate (TPR), also called *sensitivity* or *recall*, is the ratio of the correct positive predictions from the total true positives (Equation 17).

$$TPR = \frac{|TP|}{|TP \cup FN|} \tag{17}$$

Sensitivity and specificity can be combined into a single metric (Equation 18). These metrics are often used in domains in which minority classes are important (Kuhn and Johnson, 2013). For example, the sensitivity of a medical classifier (El-Dahshan et al., 2010) measures how many patients with the condition tested positive, and specificity measures how many did not have the condition and tested negative.

$$Sensitivity \times Specificity = \frac{|TP| \times |TN|}{|TP \cup FN| \times |TN \cup FP|} \tag{18}$$

Positive predictive value (PPV), or *precision*, is the ratio of the correct positive predictions from the total positive predictions (Equation 19). The difference between accuracy and precision is depicted in Figure 7.

$$PPV = \frac{|TP|}{|TP \cup FP|} \tag{19}$$

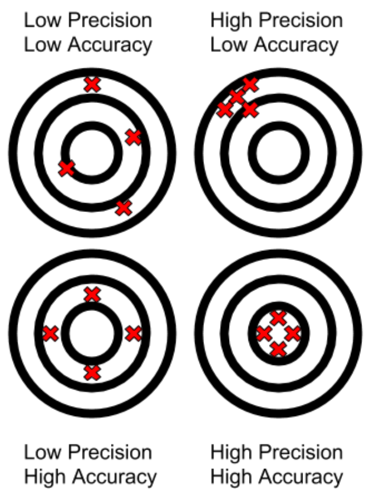


Figure 7: Accuracy vs Precision.

Precision and recall are borrowed from the discipline of *information extraction* (Sokolova and Lapalme, 2009). A composite metric called *F-score*, *F1-score*, or *F-measure* (Equation 20) can be derived by finding their harmonic mean (Kelleher et al., 2015).

$$F\text{-}score = 2 \times \frac{PPV \times TPR}{PPV + TPR} \tag{20}$$

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 (Stager et al., 2006). 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.

+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 (Ma and Cukic, 2007). 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 (Gu et al., 2009) 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 (Cichosz, 2014); for examples in Python refer to (Müller et al., 2016).

The following example is motivated by the samples in the *Scikit-Learn* documentation and the work of (Géron, 2017). The models in Figure 8 were trained on the *wines* dataset included with Scikit-Learn.

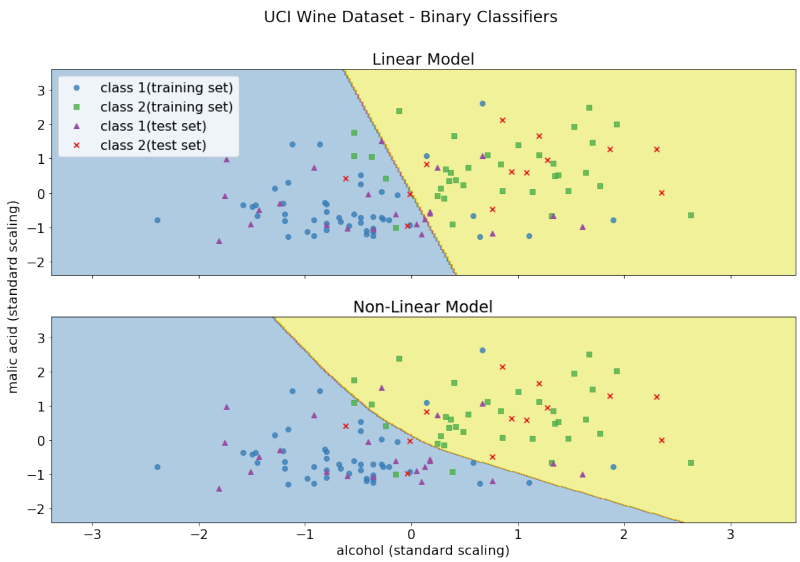


Figure 8: Decision boundary learned by a linear and non-linear binary classifier.

	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 9.

As it can be deduced from Figure 8, the decision boundary of the non-linear model is a better fit than the linear model. The CMs in Figure 9 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.

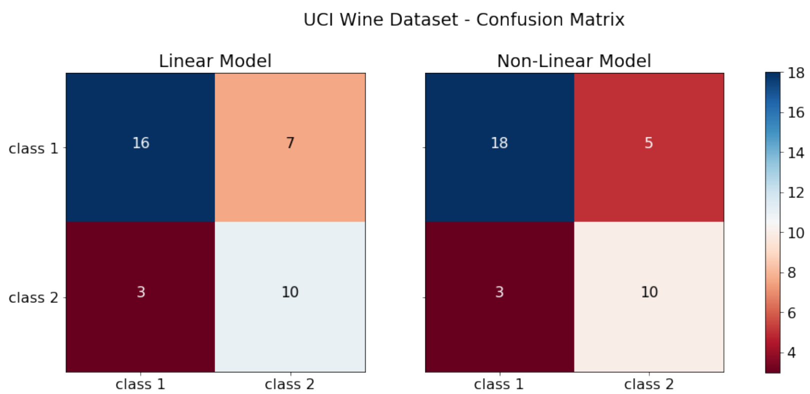


Figure 9: The linear classifier has 16 TP, 10 TN, 7 FN and 3 FP, whilst the non-linear classifier has 18 TP, 10 TN, 5 FN and 3 FP.

Cross-Validation

Cross-validation (CV) is an estimation method used on supervised learning algorithms to assess their ability to predict the output of unseen data (Varma and Simon, 2006; Kohavi, 1995). Supervised learning algorithms are computational tasks like classification or regression, that learn an input-output function based on a set of samples. Such samples are also known as the labeled training data where each example consists of an input vector and its correct output value. After the training phase, a supervised learning algorithm should be able to use the inferred function in order to map new input unseen instances, known as testing data, to their correct output values (Caruana and Niculescu-Mizil, 2006). When the algorithm incorporates supervised feature selection, cross-validation should always be done external to the selection (feature-selection performed within every CV iteration) so as to ensure the test data remains unseen, reducing bias (Ambroise and McLachlan, 2002; Hastie et al., 2001). Therefore, cross-validation, also known as out-of-sample testing, tests the function's ability to generalize to unseen situations (Varma and Simon, 2006; Kohavi, 1995).

Cross-validation has two types of approaches, being i) the exhaustive cross validation approach which divides all the original samples in every possible way, forming training and test sets to train and test the model, and ii) the non-exhaustive cross validation approach which does not consider all the possible ways of splitting the original samples (Arlot et al., 2010).

The above mentioned approaches are further divided into different cross-validation methods, as explained below.

Exhaustive cross-validation

Leave-p-out (LpO)

This method takes p samples from the data set as the test set and keeps the remaining as the training set, as shown in Fig. 11a. This is repeated for every combination of test and training set formed from the original data set and the average error is obtained. Therefore, this method trains and tests the algorithm $\binom{n}{p}$ times when the number of samples in the original data set is n, becoming inapplicable when $p > 1$ (Arlot et al., 2010).

Leave-one-out (LOO)

This method is a specific case of the LpO method having $p = 1$. It requires less computation efforts than LpO since the process is only repeated $nchoose1 = n$ times, however might still be inapplicable for large values of n (Arlot et al., 2010).

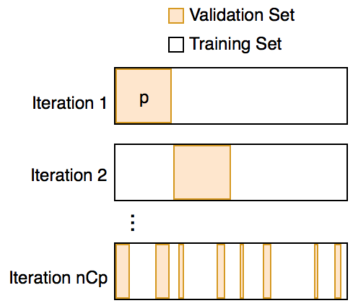


Figure 10: Leave-p-Out Exhaustive Cross Validation

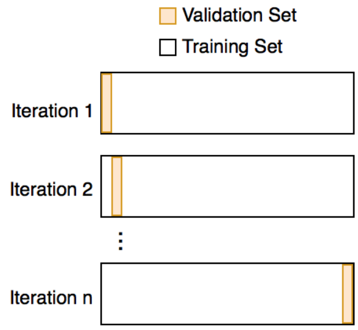


Figure 11: Leave-One-Out Exhaustive Cross Validation

Non-exhaustive cross-validation

Holdout method

This method randomly splits the original data set into two sets being the training set and the test set. Usually, the test set is smaller than the training set so that the algorithm has more data to train on. This method involves a single run and so must be used carefully to avoid misleading results. It is therefore sometimes not considered a CV method (Kohavi, 1995).

k-fold

This method randomly splits the original data set into k equally sized subsets, as shown in Fig. 12. The function is then trained and validated k times, each time taking a different subset as the test data and the remaining $(k-1)$ subsets as the training data, using each of the k subsets as the test set once. The k results are averaged to produce a single estimation. Stratified k-fold cross validation is a refinement of the k-fold method, which splits the original samples into equally sized and distributed subsets, having the same proportions of the different target labels (Kohavi, 1995).

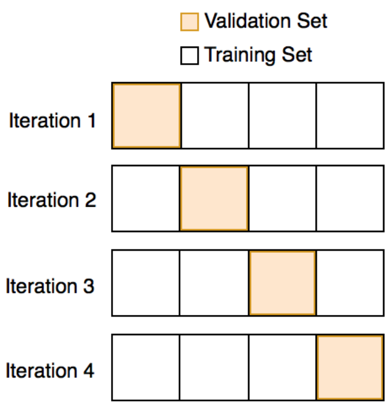


Figure 12: k-Fold Cross Validation where k=4

Repeated random sub-sampling

This method is also known as the Monte Carlo CV. It splits the data set randomly with replacement into training and test subsets using some predefined split percentage, for every run. Therefore, this generates new training and test data for each run but the test data of the different runs might contain repeated samples, unlike that of k-fold (Xu and Liang, 2001).

All of the above cross-validation methods are used to check whether the model has been overfitted or underfitted and hence estimating the model's ability of fitting to independent data . Such ability is measured using quantitative metrics appropriate for the model and data (Kohavi, 1995; Arlot et al., 2010). In the case of classification problems, the misclassification error rate is usually used whilst for regression problems, the mean squared error (MSE) is usually used. MSE is represented by Eq. 21, where n is the total number of test samples, Y_i is the true value of the i^{th} instance and $\hat{Y}_i$ is the predicted value of the i^{th} instance.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{21}$$

Underfitting is when the model has a low degree (e.g. $y = x$, where the degree is 1) and so is not flexible enough to fit the data making the model have a low variance and high bias (Baumann, 2003), as seen in Fig. 14a. Variance is the model's dependence on the training data and bias is model's assumption about the shape of the data (Arlot et al., 2010). On the other hand, as seen in Fig. 14b, overfitting is when the model has a too high degree (e.g. $y = x^{30}$, where the degree is 30) causing it to exactly fit the data as well as the noise and so lacks the ability to generalize (Baumann, 2003), making the model have a high variance. Cross-validation helps reduce this bias and variance since it uses most of the data for both fitting and testing and so helps the model learn the actual relationship within the data. This makes cross-validation a good technique for models to acquire a good bias-variance tradeoff (Arlot et al., 2010).

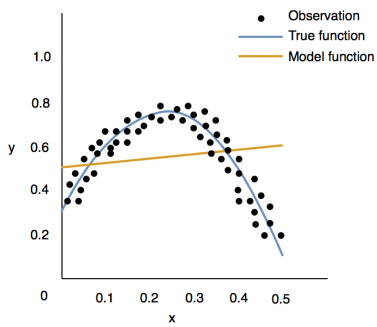


Figure 13: Model Underfitting

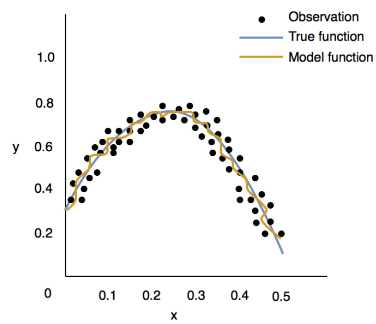


Figure 14: Model Overfitting

As stated in (Kohavi, 1995), the LOO method gives a 0% accuracy on the test set when the number of target labels are equal to the number of instances in the dataset. It is shown that the k-fold CV method gives much better results, due to its lower variance, especially when $k = 10, 20$. Furthermore, R. Kohavi et al. state that the best accuracy is achieved when using the stratified cross-validation method, since this has the least bias.

Therefore, lets take an example using the stratified k-fold cross-validation method with $k = 10$. Let's say that we are trying to solve age group classification, using eight non-overlapping age groups being 0-5, 6-10, 11-20, 21-30, 31-40, 41-50, 51-60, and 61+. We are using the FG-NET labelled data set, which contains around 1000 images of individuals aged between 0 and 69. Before we can start training our model (e.g. CNN), we must divide our data set into training and test subsets and this is where cross validation comes in. Therefore, we start by taking the 1000 images of our data set and splitting them according to their target class. Let us assume we have an equal amount of 125 $(1000/8)$ images per class[2]. As depicted in Fig. 15, we can now start forming our 10 folds by taking 10% of each age-group bucket, randomly without replacement. Hence, we will

[2] Down-sampling or up-sampling are common techniques used when there is an unequal amount of samples for the different classes.

end up with 10 subsets of 100 images that are equally distributed along all age-groups. With these subsets, we can estimate our model's accuracy with a lower bias-variance tradeoff. Since we are using 10-fold CV, we will train and test our model 10 times. For the first iteration, we shall use subset 1 as the validation set and subsets 2 to 10 as the training set, for the second iteration we use subset 2 as the test set and subsets 1 plus 3 to 10 as our training set, and so on (as shown in Fig. 12). For each iteration we use the misclassification error rate to obtain an accuracy value and we finally average the 10 accuracy rates to obtain the global accuracy of our model when solving age group classification, given the FG-NET data set. Hence, we have now estimated the prediction error of the model and have an idea of how well our model performs in solving such a problem. It is important to note that cross-validation is *just* an estimation method and when using our model in real-life applications we do not apply CV but rather train our model with all the data we have.

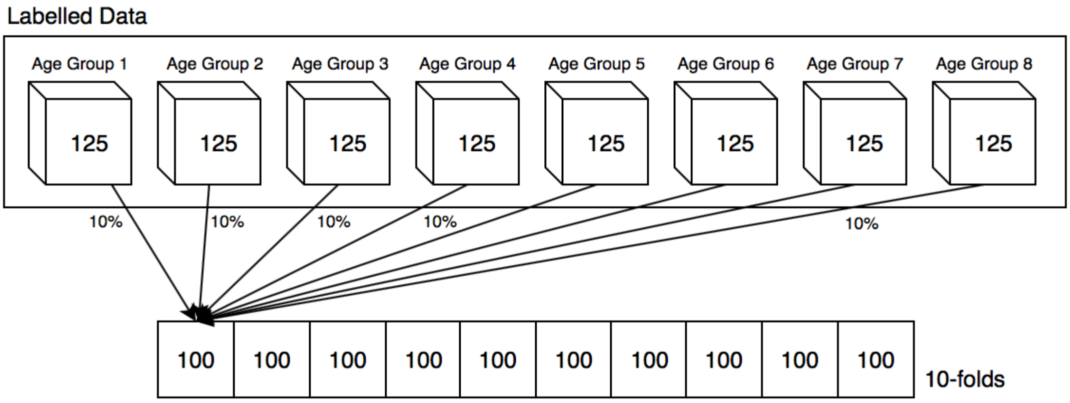


Figure 15: Stratified 10-fold cross-validation on 1000 labelled images of 8 different classes

As concluded by Varma and Simon (2006), cross-validation is well implemented when everything is taken place within every CV iteration (including preprocessing, feature-selection, learning new algorithm parameter values, etc.), and the least bias can be achieved when using nested CV methods.

Bibliography

C.C. Aggarwal. *Data Classification: Algorithms and Applications*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press, 2014. ISBN 9781466586758.

M Aly. Survey on multiclass classification methods. *Survey on Multiclass Classification Methods*, 01 2005.

Christophe Ambroise and Geoffrey J McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the national academy of sciences*, 99(10):6562–6566, 2002.

Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

Knut Baumann. Cross-validation as the objective function for variable-selection techniques. *TrAC Trends in Analytical Chemistry*, 22(6): 395–406, 2003.

Yoshua Bengio, Patrice. Simard, and Paolo. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994. ISSN 1045-9227.

Paula Branco, Luís Torgo, and Rita P Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, 49(2):31, 2016.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

Anthony L. Caterini. *Deep Neural Networks in a Mathematical Framework (SpringerBriefs in Computer Science)*. Springer, 2018. ISBN 9783319753034.

Pawel Cichosz. *Data mining algorithms: explained using R*. Wiley, 2014.

Xinyang Deng, Qi Liu, Yong Deng, and Sankaran Mahadevan. An improved method to construct basic probability assignment based on the confusion matrix for classification problem. *Information Sciences*, 340:250–261, 2016.

El-Sayed Ahmed El-Dahshan, Tamer Hosny, and Abdel-Badeeh M Salem. Hybrid intelligent techniques for mri brain images classification. *Digital Signal Processing*, 20(2):433–441, 2010.

Alan H Fielding and John F Bell. A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environmental conservation*, 24(1):38–49, 1997.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2016. ISBN 0262035618.

Qiong Gu, Li Zhu, and Zhihua Cai. Evaluation measures of the classification performance of imbalanced data sets. In *International Symposium on Intelligence Computation and Applications*, pages 461–471. Springer, 2009.

Aboul Ella Hassanien and Diego Alberto Oliva. *Advances in Soft Computing and Machine Learning in Image Processing*, volume 730. Springer, 2017.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *Springer Texts in Statistics*, volume 102. 2006. ISBN 9780387781884.

László A Jeni, Jeffrey F Cohn, and Fernando De La Torre. Facing imbalanced data–recommendations for the use of performance metrics. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 245–251. IEEE, 2013.

Jain Pei Jiawei Han, Micheline Kamber. *Data Mining âĂŞ Concepts & Techniques*. 2011. ISBN 9780123814791.

John D Kelleher, Brian Mac Namee, and Aoife D'Arcy. *Fundamentals of machine learning for predictive data analytics*. MIT Press, 2015.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8.

B. Kolo. *Binary and Multiclass Classification*. Weatherford Press, 2011. ISBN 9781615800162.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

Shailesh Kumar, Joydeep Ghosh, and Melba M Crawford. Hierarchical Fusion of Multiple Classifiers for Hyperspectral Data Analysis 1. Technical report, 2002.

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1, 2001.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Zhiyong Lu, Duane Szafron, Russell Greiner, Paul Lu, David S Wishart, Brett Poulin, John Anvik, Cam Macdonell, and Roman Eisner. Predicting subcellular localization of proteins using machine-learned classifiers. *Bioinformatics*, 20(4):547–556, 2004.

Salha M. Alzahani, Afnan Althopity, Ashwag Alghamdi, Boushra Alshehri, and Suheer Aljuaid. An overview of data mining techniques applied for heart disease diagnosis and prediction. *Lecture Notes on Information Theory*, 2, 2015.

Yan Ma and Bojan Cukic. Adequate and precise evaluation of quality models in software engineering studies. In *Proceedings of the third International workshop on predictor models in software engineering*, page 1. IEEE Computer Society, 2007.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

James H Martin and Daniel Jurafsky. *Speech and language processing*. Pearson, 2018.

Mohsen Moradi, Shafiee Sardasht, and Maliheh Ebrahimpour. An Application of Support Vector Machines in Bankruptcy Prediction; Evidence from Iran. *World Applied Sciences Journal*, 17(6):710–717, 2012.

Andreas C Müller, Sarah Guido, et al. *Introduction to machine learning with Python: a guide for data scientists*. O'Reilly, 2016.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7.

S. Neelamegam and E. Ramaraj. Classification algorithm in Data mining: An Overview. *International Journal of P2P Network Trends and Technology*, 3:369, 2013.

Troy Raeder, George Forman, and Nitesh V Chawla. Learning from imbalanced data: evaluation matters. In *Data mining: Foundations and intelligent paradigms*, pages 315–331. Springer, 2012.

Lior Rokach and Oded Maimon. *Data mining with decision trees: theory and applications*. 2007. ISBN 9789812771711.

Frank Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323 (6088), 1986. ISSN 0028-0836.

Pinki Sagar. Analysis of Prediction Techniques based on Classification and Regression General Terms. Technical Report 7, 2017.

Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning and data mining*. Springer, 2017.

Jan Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-based Algorithms (Applied Optimization Book 97)*. Springer US, 2005. ISBN 978-0-387-24348-1.

Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

Mathias Stager, Paul Lukowicz, and Gerhard Troster. Dealing with class skew in context recognition. In *26th International Conference on Distributed Computing Systems*, pages 58–58. IEEE, 2006.

Lyn Thomas. A survey of credit and behavioural scoring: Forecasting financial risk of lending to consumers. *International Journal of Forecasting*, 16:149–172, 2000.

Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7(1): 91, 2006.

Bendi Venkata Ramana, MSurendra Prasad Babu, N B Venkateswarlu, and Associate Professor. A Critical Study of Selected Classification Algorithms for Liver Disease Diagnosis. *International Journal of Database Management Systems (IJDMS)*, 3, 2011.

Qing-Song Xu and Yi-Zeng Liang. Monte carlo cross validation. *Chemometrics and Intelligent Laboratory Systems*, 56(1):1–11, 2001.

Yangchi Chen, M.M. Crawford, and J. Ghosh. Integrating support vector machines in a hierarchical output space decomposition framework. *IEEE International Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004*, 2(October 2004):949–952, 2014.

Index