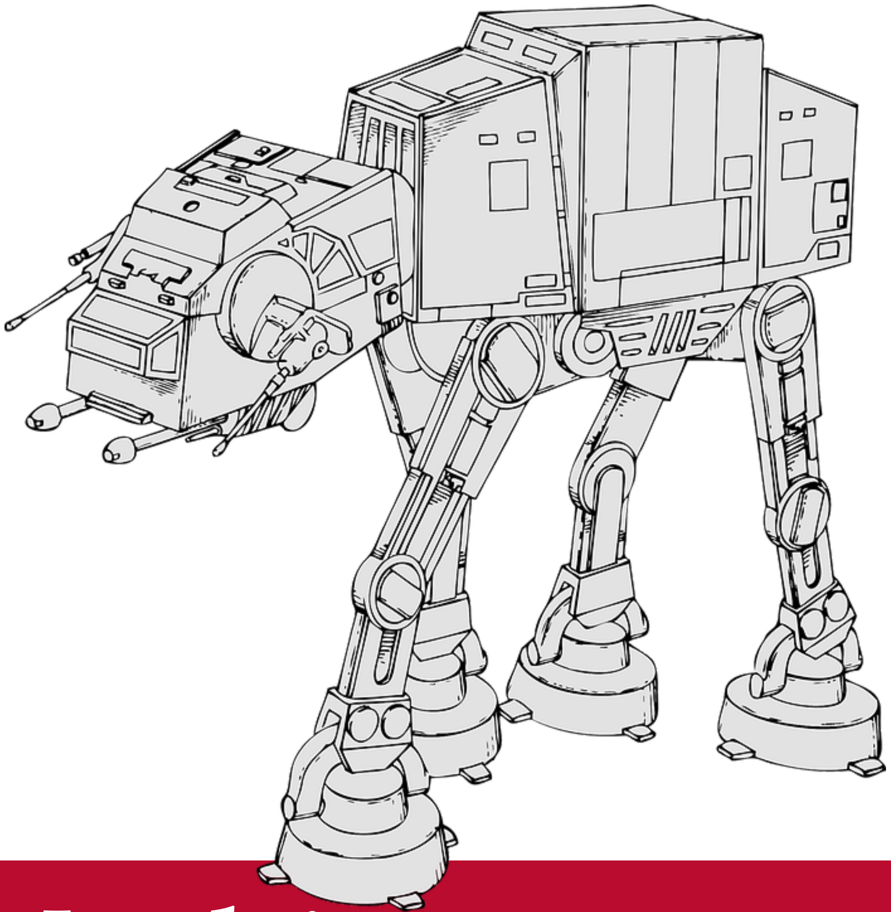
A Machine Learning Study Guide



Machine Learning Handbook

The Definitive Guide

ICS5110, class of 2018/9

L-Università
ta' Malta

Copyright © 2019 ICS5110 APPLIED MACHINE LEARNING class of 2018/9, University of Malta.

JEAN-PAUL EBEJER, DYLAN SEYCHELL, LARA MARIE DEMAJO, DANIEL FARRUGIA, KEITH MINTOFF, STEFAN CASSAR ADD YOUR NAME TO THIS LIST

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at `http://www.apache.org/licenses/LICENSE-2.0`. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, January 2019

Contents

Introduction 5

Activation Functions 7

Confusion Matrix 11

Cross-Validation 17

Dimensionality Reduction 21

Index 29

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

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

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

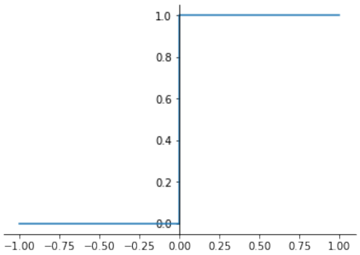


Figure 1: A graph of the step function.

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

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

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

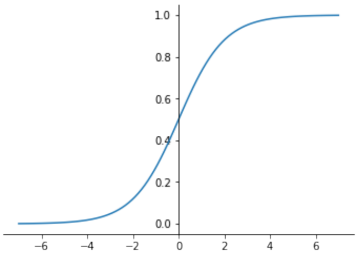


Figure 2: A graph of the sigmoid function.

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).

Hyperbolic Tangent

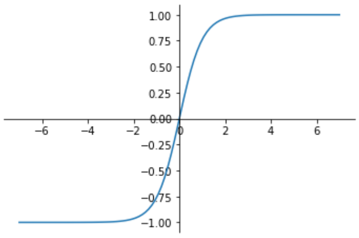


Figure 3: A graph of the hyperbolic tangent (tanh) function.

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

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

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.

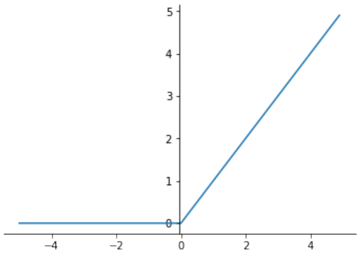


Figure 4: A graph of the ReLU function.

Rectified Linear Unit

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

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

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.

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 11-17) 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 11). 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.

$$ACC = \frac{|TP \cup TN|}{|TP \cup FP \cup TN \cup FN|} \tag{11}$$

Negative Predictive Value (NPV) is the ratio of the correct negative predictions from the total negative predictions (Equation 12).

$$NPV = \frac{|TN|}{|TN \cup FN|} \tag{12}$$

True Negative Rate (TNR), or *Specificity*, is the ratio of the correct negative predictions from the total true negatives (Equation 13).

$$TNR = \frac{|TN|}{|TN \cup FP|} \tag{13}$$

True Positive Rate (TPR), also called *Sensitivity* or *Recall*, is the ratio of the correct positive predictions from the total true positives (Equation 14).

$$TPR = \frac{|TP|}{|TP \cup FN|} \tag{14}$$

Sensitivity and Specificity can be combined into a single metric (Equation 15). 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.

$$\mathit{Sensitivity} \times \mathit{Specificity} = \frac{|TP| \times |TN|}{|TP \cup FN| \times |TN \cup FP|} \tag{15}$$

Positive Predictive Value (PPV), or *Precision*, is the ratio of the correct positive predictions from the total positive predictions (Equation 16). The difference between accuracy and precision is depicted in Figure 5.

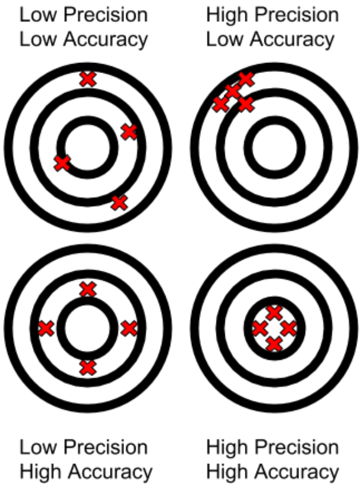


Figure 5: Accuracy vs Precision.

$$PPV = \frac{|TP|}{|TP \cup FP|} \tag{16}$$

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 17) can be derived by finding their harmonic mean (Kelleher et al., 2015).

$$\mathit{F\text{-}score} = 2 \times \frac{PPV \times TPR}{PPV + TPR} \tag{17}$$

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 6 were trained on the *wines* dataset included with Scikit-Learn.

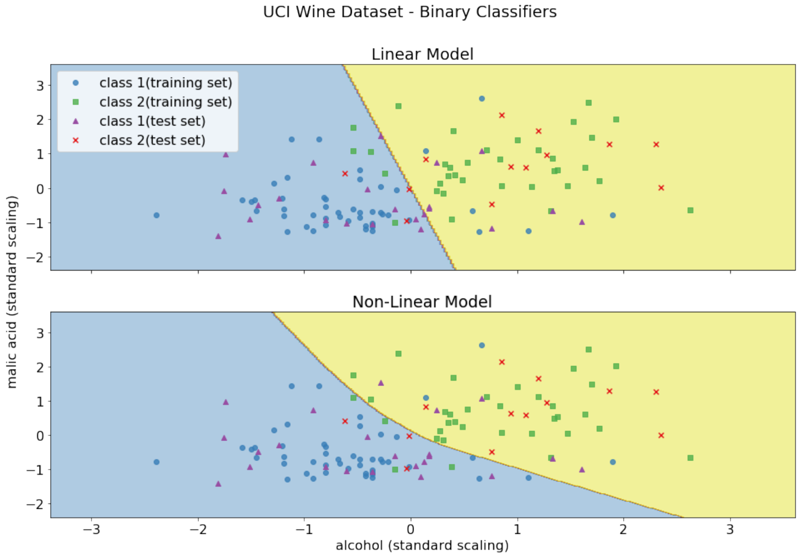


Figure 6: Decision boundary learned by a linear and non-linear binary classifier.

	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 7.

As it can be intuitively deduced from Figure 6, the decision boundary of the non-linear model is a better fit than the linear model. The CMs in Figure 7 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.

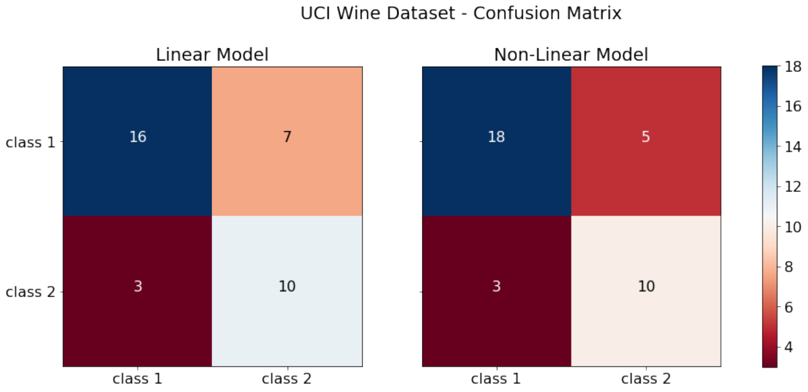


Figure 7: The linear classifier has 16 TP, 10 TN, 7 FN and 3 FP, whilst the non-linear classifier has 18 TP, 10 TN, 5 FN and 3 FP.

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. 9a. 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).

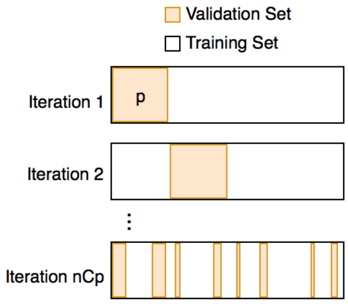

Validation Set
Training Set
Iteration 1
p
Iteration 2
Iteration nCp

Figure 8: Leave-p-Out Exhaustive Cross Validation

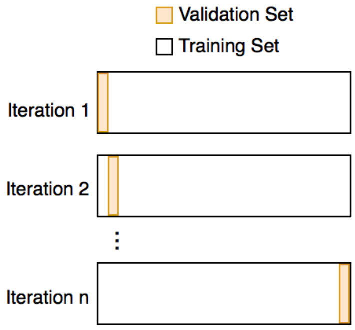

Validation Set
Training Set
Iteration 1
Iteration 2
Iteration n

Figure 9: 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. 10. 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).

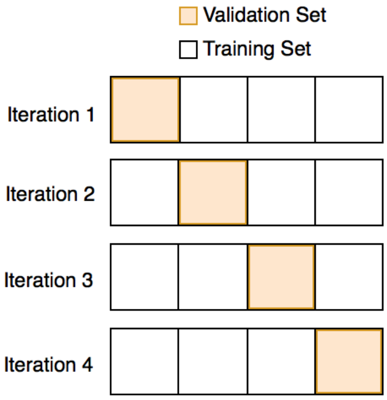

Validation Set
Training Set
Iteration 1
Iteration 2
Iteration 3
Iteration 4

Figure 10: 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. 18, 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.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{18}$$

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. 12a. 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. 12b, 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).

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. 13, we can now start forming our 10 folds by taking 10% of each age-group bucket, randomly without replacement. Hence, we will

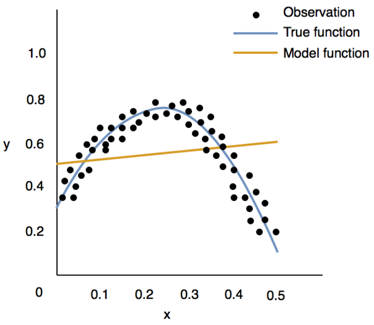


Figure 11: Model Underfitting

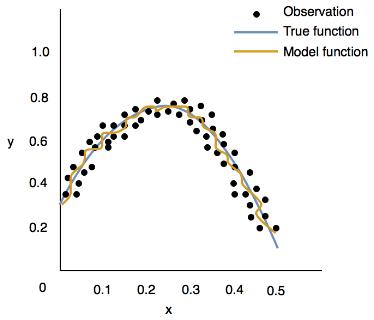


Figure 12: Model Overfitting

[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. 10). 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.

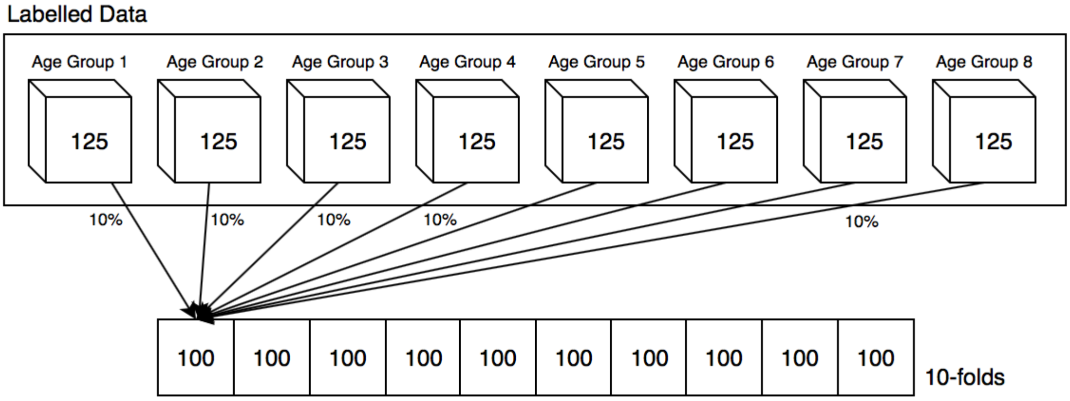

Labelled Data
Age Group 1
Age Group 2
Age Group 3
Age Group 4
Age Group 5
Age Group 6
Age Group 7
Age Group 8
125
125
125
125
125
125
125
125
10%
10%
10%
10%
10%
100
100
100
100
100
100
100
100
100
100
10-folds

Figure 13: 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.

Dimensionality Reduction

As the volume of data collected increased, several machine learning approaches have been formulated to identify patterns and perform predictions from this data. Apart from their large volume, these datasets constitute of a large number of variables. This high dimensionality within the data presented challenges such as excessive computational costs and increased memory requirements (Bolón-Canedo et al., 2015).

For these reasons, prior to applying any of these data mining techniques, pre-processing mechanisms are normally applied on the data. These methods include dimensionality reduction which is the process in which the large number of input variables are reduced to a smaller set of features (Sorzano et al., 2014).

Dua and Saini (2009) mention *feature selection* and *feature extraction* as two possible methods which reduce the dimensionality of the data. Feature selection involves selecting only the relevant features of the dataset, whilst ignoring the remaining features (Dua and Saini, 2009).

On the other hand, feature extraction techniques are applied to construct a feature vector with lower dimensionality. Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are two common techniques applied for feature extraction, in which a linear transformation matrix is used to project the original features to a new feature space (Wang and Paliwal, 2003).

Random Forests

Breiman (2001) introduced Random Forests (RF) as an algorithm for *classification* and *regression* problems. However, this method is also useful in determining feature importance or irrelevance, thus widely used for variable selection (Genuer et al., 2010).

The concept behind RFs involves the iterative construction of a number of binary *decision trees*, which have a low degree of correlation (Genuer et al., 2010). In each iteration, a random sample is extracted from the training sample on which the decision tree will be trained. During training, each split involves randomly selecting k features from the original feature vector P, where $k < P$. Data is split using the best feature out of the selected k features. Once the separate decision trees, also referred to as random forests, are constructed, each tree can be used to predict outcomes for new samples. In classification problems, these outcomes are then averaged to obtain the final prediction (Kuhn and Johnson, 2013).

Díaz-Uriarte and Alvarez de Andrés (2006) perform feature selection using a RF approach to select a smaller set of genes that can still provide effective predictions in classification or regression problems. In their strategy, variable importances are initially calculated within each iteration. The variables with the smallest importances are eliminated and a new forest is then constructed using the remaining high importance variables. Finally, the prediction error of each forest, also referred to as the out-of-bag (OOB) error rate, is calculated. The genes belonging to forests with the smallest error in prediction, are selected as the final variables (Díaz-Uriarte and Alvarez de Andrés, 2006).

Principle Component Analysis

Jolliffe (2002) introduces PCA as the technique which aims "to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set." PCA aims to transform a set of correlated variables P into a set of uncorrelated k variables, where $k < P$ and the k variables maintain the majority of the information in P.

Prior to applying PCA, the dataset is adjusted such that the means of the original variables are subtracted from the observations, to produce a dataset with mean zero (Jackson, 2005). PCA works by initially constructing the *covariance matrix* of the data (Guan and Dy, 2009).

$$cov(x,y) = \frac{\sum_{i=1}^{n}(X_i - \hat{X})(Y_i - \hat{Y})}{n-1} \tag{19}$$

The covariance between two variables x and y may be computed by equation 19, where X_i is the x variable for the i^{th} sample, Y_i is the y variable for the i^{th} sample, $\hat{X}$ and $\hat{Y}$ are the means of the X and Y variables respectively, and n is the sample size. A high covariance value indicates that the variables are correlated, whereas a zero value indicates that no correlation exists between the variables. Moreover, the covariance between x and y is equal to the variance of x, when $x = y$.

$$\begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

Figure 14: Covariance Matrix.

The variance between features is indicated by the diagonal values of a covariance matrix (figure 14), whilst the off-diagonal values refer to the covariance between the features. High values in the off-diagonal terms indicate that high redundancy exists, and thus the covariance matrix must be diagonalized to minimize the correlations between the features.

The resultant diagonalized covariance matrix is used to calculate the *eigenvalues* and *eigenvectors*. The eigenvectors refer to the direction in which the data varies, and these are ordered such that the ones with the highest eigenvalues are chosen as the principal components of the data. Eventually, the selected vectors are kept to form a *feature vector*, which will be used to construct the new dataset with lower dimensionality, as in equation 20

$$NewDataset = FeatureVector^{T} x AdjustedData^{T} \tag{20}$$

This equation 20 multiplies the transpose of the feature vector by the transpose of the mean adjusted data, to obtain the original data with smaller dimensionality (Jackson, 2005).

Linear Discriminant Analysis

LDA is a supervised dimensionality reduction technique which produces a lower dimensionality feature space in which a feature vector belonging to a class is distinguishable from that of the other classes (Sharma and Paliwal, 2015). Figure 15 depicts the projection of a two-dimensional feature vector to a one-dimensional feature space. C_1, C_2 and C_3 denote the three different classes being used within the feature vectors. By projecting the feature vectors onto line $\hat{W}$, which represents one dimension, it is noted that a strong relationship exists between the feature vectors of the three classes. LDA seeks an orientation which strongly separates the feature vectors of one class from those belonging to other classes. This orientation can be obtained by rotating $\hat{W}$, resulting in projection W, where the feature vectors of the classes are highly separated. Algorithmically, Fisher's criterion is maximized to obtain the required one-dimensional feature vector W.

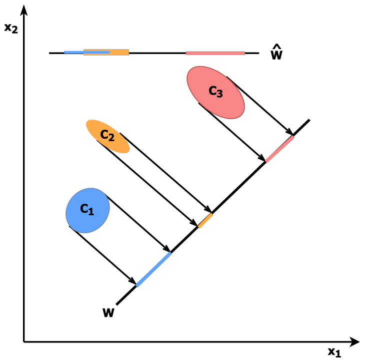


Figure 15: Projection of a two-dimensional feature vector to a one-dimensional feature space.

Feature extraction using LDA (Sharma and Paliwal, 2015; Huang et al., 2002) works by initially calculating the separability between the mean values of each class.

$$S_B = \sum_{i=1}^{c} n_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{21}$$

This is referred to as the *between-class matrix* and is represented by equation 21, where c is the number of classes, n_i is the number of samples for the i^{th} class, μ_i is the mean for the i^{th} class and μ is the total mean of all samples.

$$S_W = \sum_{j=1}^{c} \sum_{i=1}^{n_j} (x_{ij} - \mu_j)(x_{ij} - \mu_j)^T \tag{22}$$

Following this, the *within-class matrix* is calculated using equation 22, where c is the number of classes, n_j is the number of samples for the j^{th} class, x_{ij} is the i^{th} sample in the j^{th} class, and μ_j is the mean for the j^{th} class. This matrix indicates the distance between the mean and sample values for each class.

Finally, a *transformation matrix* W which maximizes the between-class variance and minimizes the within-class variance should be constructed.

$$\arg\max_{w} \frac{W^T S_B W}{W^T S_W W} \tag{23}$$

$$S_W W = \lambda S_B W \tag{24}$$

Fisher's criterion, shown in equation 23 is used to obtain the maximum ratio of the between-class matrix (denoted by S_B) and within class matrix (represented by S_W). By transforming the latter into equation 24, where λ denotes the eigenvalues, eigenvectors and the corresponding eigenvalues of W may be calculated.

A lower dimensional space is constructed by selecting the eigenvectors having the highest eigenvalues.

Bibliography

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.

Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. *Feature Selection for High-Dimensional Data*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer International Publishing, 2015. ISBN 978-3-319-21857-1.

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

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

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.

Ramón Díaz-Uriarte and Sara Alvarez de Andrés. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7, 2006.

Sumeet Dua and Sheetal Saini. Data shrinking based feature ranking for protein classification. In Sushil K. Prasad, Susmi Routray, Reema Khurana, and Sartaj Sahni, editors, *Information Systems, Technology and Management*, pages 54–63. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00405-6.

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.

Robin Genuer, Jean Michel Poggi, and Christine Tuleau-Malot. Variable selection using random forests. *Pattern Recognition Letters*, 31 (14):2225–2236, 2010.

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.

Yue Guan and Jg Dy. Sparse probabilistic principal component analysis. In *AISTATS*, pages 185–192, 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.

Rui Huang, Qingshan Liu, Hanqing Lu, and Songde Ma. Solving the small sample size problem of LDA. *International Conference on Pattern Recognition*, 3:29–32, 2002.

J.E. Jackson. *A User's Guide to Principal Components*. Wiley Series in Probability and Statistics. Wiley, 2005. ISBN 9780471725329.

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.

I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002. ISBN 9780387954424.

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.

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.

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.

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.

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.

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.

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.

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.

Alok Sharma and Kuldip K Paliwal. Linear discriminant analysis for the small sample size problem: an overview. *International Journal of Machine Learning and Cybernetics*, 6(3):443–454, 2015.

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.

C. O. S. Sorzano, J. Vargas, and A. Pascual Montano. A survey of dimensionality reduction techniques. *arXiv*, pages 1–35, 2014.

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.

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

Xuechuan Wang and Kuldip K Paliwal. Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition. *Pattern Recognition*, 36(10):2429–2439, 2003.

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

Index

activation functions, 7–10
  hyperbolic tangent, 9
  linear activation function, 8
  rectified linear unit (relu), 9
  sigmoid function, 8
  step function, 7

confusion matrix, 11–14
  1-vs-1, 13
  1-vs-rest, 13
  accuracy, 12
  error rate, 13
  f-score, 13
    f-measure, 13
    f1-score, 13
  false discovery rate, 13
  false negative rate, 13
  false omission rate, 13
  false positive rate, 13
  negative predictive value, 12
  positive predictive value, 13
    precision, 13
  true negative rate, 12
    specificity, 12
  true positive rate, 12
    recall, 12
    sensitivity, 12
cross-validation, 17
  holdout, 18
  k-fold, 18
  leave-one-out, 18
  leave-p-out, 17

dimensionality reduction, 21–24
  feature extraction, 21
  feature selection, 21
  linear discriminant analysis, 23
    between-class matrix, 23
    transformation matrix, 23
    within-class matrix, 23
  principle component analysis, 22
    covariance matrix, 22
    eigenvalues, 22
    eigenvectors, 22
    feature vector, 22
  random forests, 21
    classification, 21
    decision trees, 21
    regression, 21

false negatives, 11
false positives, 12

license, 1

overfitting, 19

repeated random sub-sampling, 18

true negatives, 12
true positives, 11

underfitting, 19