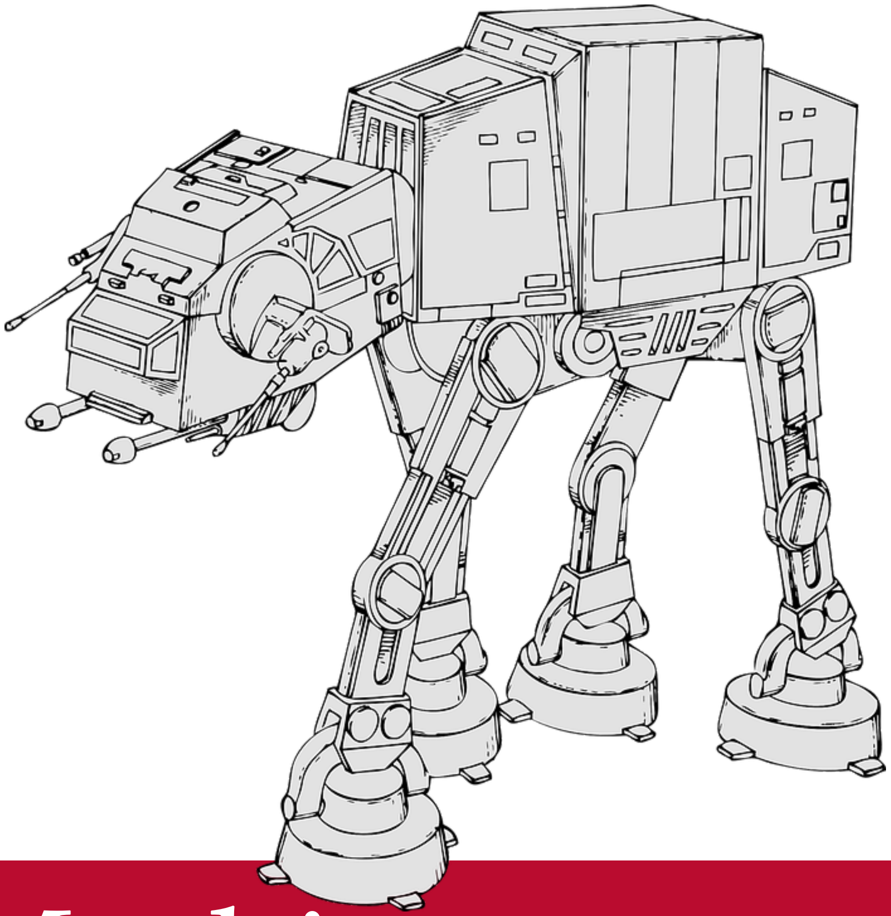
A Machine Learning Study Guide



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, DAVID FARRUGIA, IVAN SALOMONE, ANDREW CACHIA, JAKE J. DALLI, JOSEPH AZZOPARDI, CLAIRE GALEA 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

Classification 11

Confusion Matrix 15

Cross-Validation 19

Noise in datasets 23

Online Learning Algorithms 27

Regularisation of Models 31

Sample Selection Bias 35

Transfer Learning 41

Index 55

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

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

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

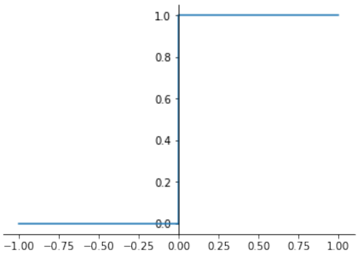


Figure 1: A graph of the step function.

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

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

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

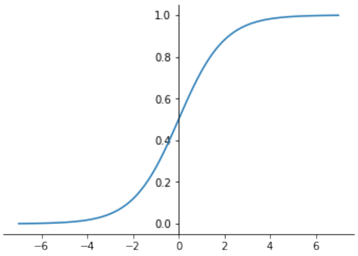


Figure 2: A graph of the sigmoid function.

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

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

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

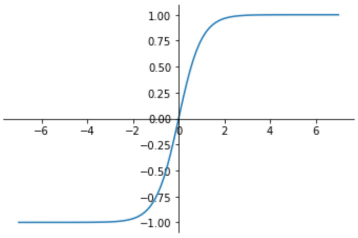


Figure 3: A graph of the hyperbolic tangent (tanh) function.

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

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

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

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

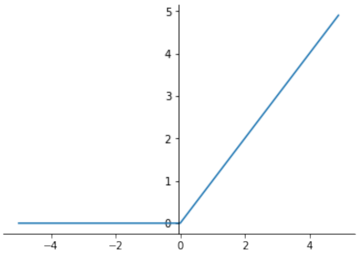


Figure 4: A graph of the ReLU function.

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.

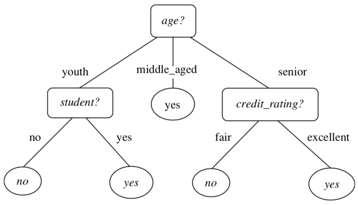


Figure 5: A binary Decision Tree example which determines whether a customer will purchase a computer or not. Reproduced from (Jiawei Han, Micheline Kamber, 2011)

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.

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

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.

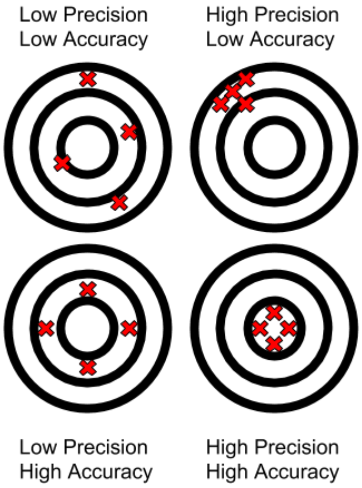


Figure 7: Accuracy vs Precision.

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

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).

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

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.

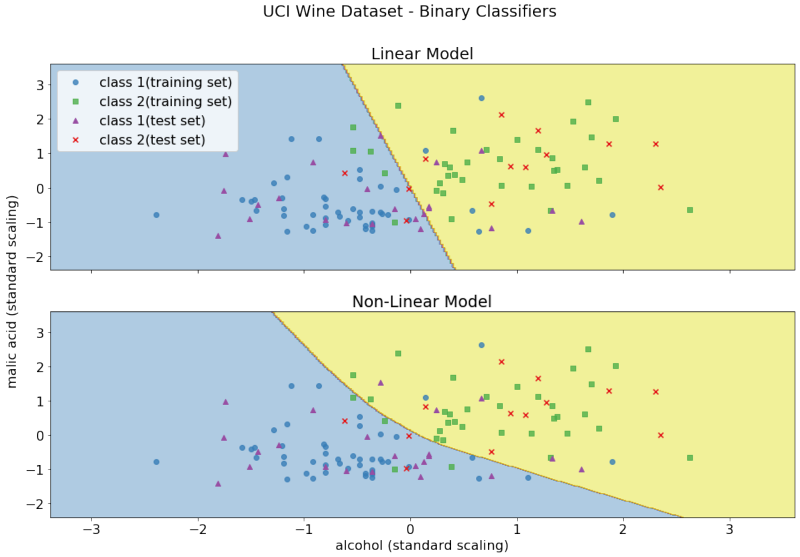


Figure 8: Decision boundary learned by a linear and non-linear binary classifier.

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.

	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.

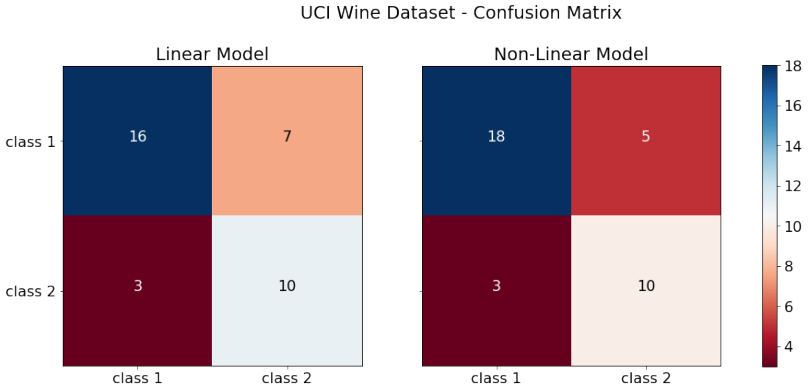


Figure 9: The linear classifier has 16 TP, 10 TN, 7 FN and 3 FP, whilst the non-linear classifier has 18 TP, 10 TN, 5 FN and 3 FP.

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).

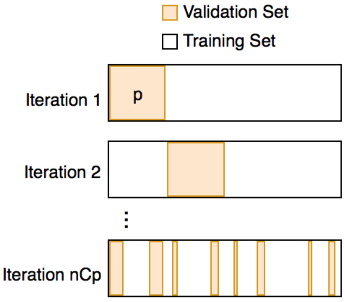

Validation Set
Training Set
Iteration 1
p
Iteration 2
Iteration nCp

Figure 10: Leave-p-Out Exhaustive Cross Validation

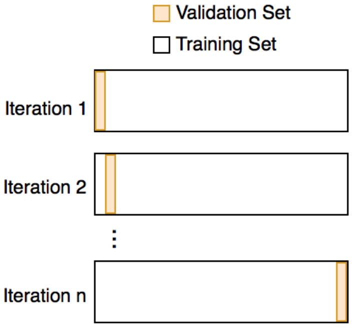

Validation Set
Training Set
Iteration 1
Iteration 2
Iteration n

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).

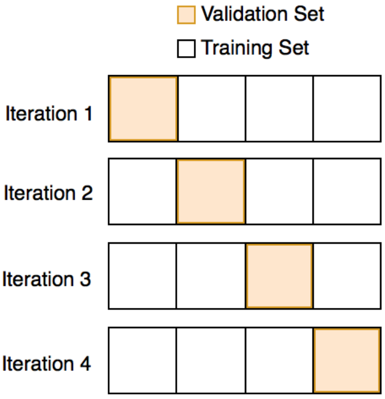

Validation Set
Training Set
Iteration 1
Iteration 2
Iteration 3
Iteration 4

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).

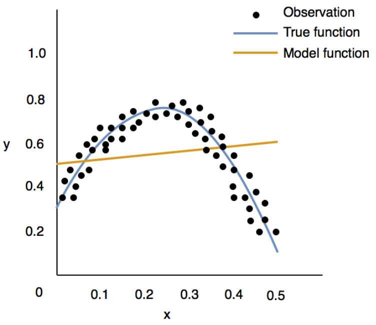


Figure 13: Model Underfitting

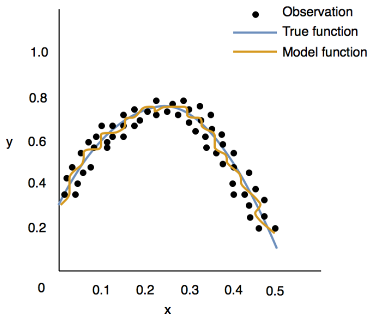


Figure 14: Model Overfitting

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.

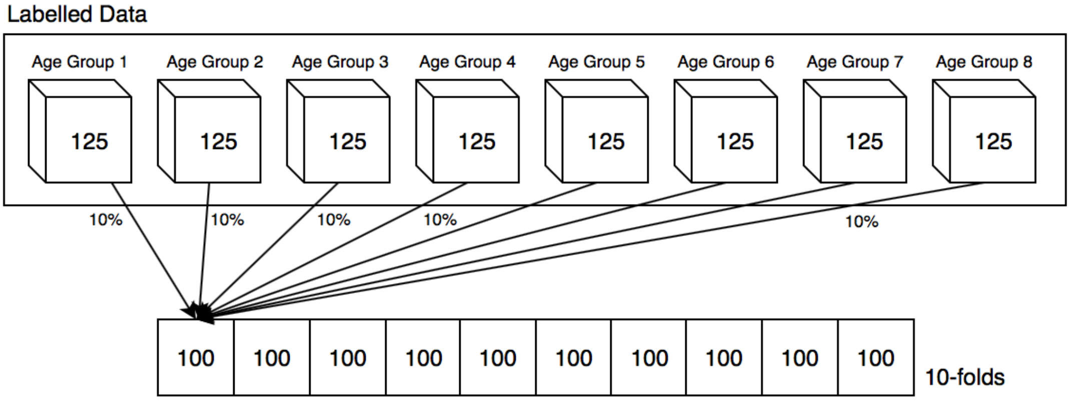

Labelled Data
Age Group 1
Age Group 2
Age Group 3
Age Group 4
Age Group 5
Age Group 6
Age Group 7
Age Group 8
125
125
125
125
125
125
125
125
10%
10%
10%
10%
10%
100
100
100
100
100
100
100
100
100
100
10-folds

Figure 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.

Noise in datasets

As defined by Hickey (1996), noise is anything that distorts the relationship between the describing attributes and the class. Broadly speaking there are two types of noise: attribute (or feature) noise and class (or label) noise (Garćia et al., 2013; Frénay and Verleysen, 2014). In attribute noise, errors in one or more of the attributes that describe the class distort the true representation of the data. Class noise on the other hand, is the mislabelling of instances in a dataset. Missing observations can exist in both attributes and class, and are also considered as noise (Zhu and Wu, 2004).

The example dataset in Table 7 shows the various forms of noise. Instances *5* and *6* exhibit attribute noise, with instance *5* having a missing value for attribute x_1, and instance *6* having attribute x_2 erroneously marked as *c*. Instances *1*, *2*, *4* and *7* exhibit class noise. Instance *1* was mislabelled as *0*, which should read *1*. Instances *2* and *4* are contradicting instances, implying that either one was mislabelled or the attribute readings were interpreted differently. Instance *7* has a missing label.

#	x_1	x_2	Y
1	a	d	0
2	a	c	1
3	b	c	1
4	a	c	0
5	-	d	0
6	b	c	1
7	b	d	-

Table 7: A sample from a noisy dataset. *Note: Noise is marked in red for ease of demonstration.*

Attribute noise is normally introduced through errors in the data collection or data processing stages, but also by corruption whilst the data are stored or transported (Garćia et al., 2013). Class noise on the other hand can be introduced through (Frénay and Verleysen, 2014):

- insufficient information when labelling the instances;
- expert labelling errors;
- subjectivity of the classes (for example in the case of medical diagnosis where experts can classify differently);
- encoding and communication problems; and
- using a cheap labelling method (such as non-expert or automated approaches).

Unfortunately, noise is very common since reliable, noise free data are expensive and time consuming to obtain (Frénay and Verleysen, 2014). The typical error rate in a dataset is 5%. (Zhu and Wu, 2004).

Effects of noise

When learning a concept, algorithms assume and expect a correct and perfectly labelled dataset (Frénay et al., 2014). Primarily, the noise in the training set reduces the predictive ability of the inferred model (Garćia et al., 2013; Frénay et al., 2014; Frénay and Verleysen, 2014).

This can be seen in Figure 16 that shows the effect of noise on the classification accuracy of various classifiers.

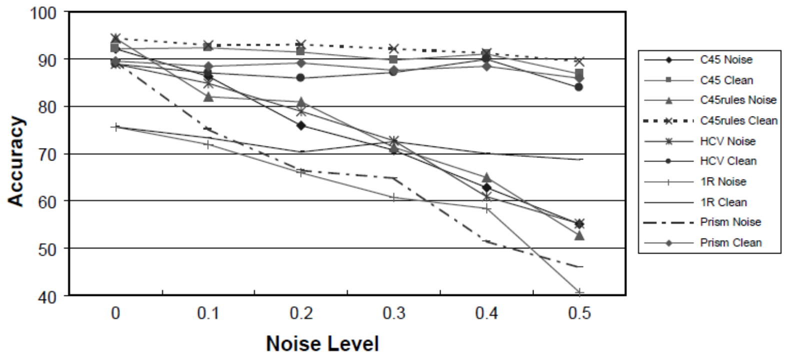


Figure 16: A graph showing the effect of noise on the classification accuracy of various classifiers. Each model was trained on a noisy dataset and on a manually cleaned dataset. These are represented as *XXX Noise* and *XXX Clean* respectively, where *XXX* denotes the classifier in question (Zhu and Wu, 2004).

The extent of the damage caused by noise depends on various factors. Class noise has been found to be more harmful than attribute noise (Zhu and Wu, 2004). The reason for this is that whilst an attribute shares the predictive capability with other attributes, there is only one label and errors confuse the learning algorithm (Zhu and Wu, 2004).

Zhu and Wu (2004) also found that not all attributes are equal in predicting the class and noise in the more important features are more damaging than others. The authors found that the higher the correlation between attribute and class the greater the impact of noise in the attribute.

Learning from a noisy dataset requires a larger training set (Frénay et al., 2014; Frénay and Verleysen, 2014) and is usually lengthier (Garćia et al., 2013). Another cost of noise is the higher complexity of the inferred model (Garćia et al., 2013; Frénay et al., 2014; Frénay and Verleysen, 2014).

Dealing with noise

To improve the performance of the inferred model, the effect of noise must be minimised (Zhu and Wu, 2004). There are various ways addressed in literature by which one can minimise such effects, which can be broadly classified into two categories: (i) improving the quality of the dataset (i.e. cleansing noise); and (ii) building models that are robust to noise.

Cleansing noise

Noise is usually identified by a domain expert since automatic noise identification is difficult (Garćia et al., 2013). However, a domain expert is not always available and manual identification of noise is a lengthy process, therefore automated noise identification techniques are needed that either correct the noisy data or eliminate it.

One cannot identify noise without making assumptions (Frénay and Verleysen, 2014). Techniques that filter out noise remove instances that appear mislabelled or that disproportionately increase the model complexity (Frénay et al., 2014). In some cases a classifier (noise filter) is trained to identify noisy instances. Other techniques aim at correcting errors or imputing missing values (Zhu and Wu, 2004).

For class noise, filtering out instances that appear noisy was found to improve results, but in the case of attribute noise, removing an instance for a noisy (including missing) attribute would not make sense, especially since other attributes may contain information that is useful for learning (Zhu and Wu, 2004). Instead, correcting or imputing the values was found to achieve better results. Techniques that deal with class noise include *ensemble filters*, *cross-validated committees filters* and *iterative-partitioning filters* (García et al., 2015).

Ensemble filters aim to identify and remove mislabelled instances in the pre-processing phase (García et al., 2015; Brodley and Friedl, 1999). The filter consists of an ensemble of m different classifiers (for example a decision tree, a 1-NN and an SVM) that are trained on the training data to act as noise filters. The training data is split into n parts and for each of the m classifiers, n different algorithms are trained. Each algorithm classifies one of the n subsets after being trained on the remaining n-1 subsets. If the predicted class does not match the true class, the element is marked as potentially noisy. The results of each of the m classifiers are then compared and a consensus whether an element is noisy is obtained through a voting scheme. Elements deemed noisy are removed from the training data.

Cross-validated committees filters use an approach very similar to that of ensemble filters except that the ensemble is made up only of decision trees (García et al., 2015; Verbaeten and Van Assche, 2003). It uses k-fold cross-validation to split the training data and train the base classifiers. Once again the noisy elements are identified through a voting scheme and eliminated.

Iterative-partitioning filters are used for cleaning large datasets (García et al., 2015). The training data is partitioned into manageable parts and cleansed in iterations until a stopping criterion is met. The stopping criterion is normally the percentage of noise that is tolerated.

Figure 17 depicts the approach taken by Zhu and Wu (2004) to correct attribute noise, where for each attribute a strong correlation with other attributes is sought upon which noisy instances can be predicted (through a learning model) and corrected. If a correlation is not found, corrections are based on other methods such as *clustering* or *k-nearest neighbour*.

The choice of noise filtering method should be based on the task at hand (Frénay and Verleysen, 2014). The effectiveness of the noise filter should be evaluated on a dataset that is degraded with artificial noise. The filtering precision can then be calculated through the number of correct instances that are filtered (*Type I errors* - Equation 22) and the number of incorrect instances that are not filtered (*Type II errors* -

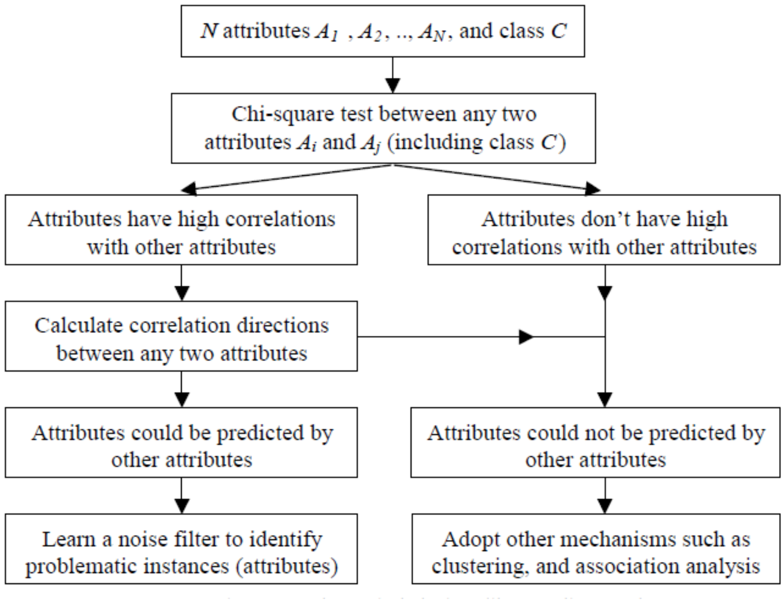

N attributes A_1, A_2, .., A_N, and class C
Chi-square test between any two attributes A_i and A_j (including class C)
Attributes have high correlations with other attributes
Attributes don't have high correlations with other attributes
Calculate correlation directions between any two attributes
Attributes could be predicted by other attributes
Attributes could not be predicted by other attributes
Learn a noise filter to identify problematic instances (attributes)
Adopt other mechanisms such as clustering, and association analysis

Figure 17: The approach adopted by Zhu and Wu (2004) to filter and correct attribute noise.

Equation 23). These are calculated as follows:

$$ER_1 = \frac{\text{\# of correctly labelled instances which are removed}}{\text{\# of correctly labelled instances}} \tag{22}$$

$$ER_2 = \frac{\text{\# of mislabelled instances which are not removed}}{\text{\# of mislabelled instances}} \tag{23}$$

Consequently, the *Noise Elimination Precision* (NER) can be calculated through Equation 24 (Frénay and Verleysen, 2014).

$$NER = \frac{\text{\# of mislabelled instances which are removed}}{\text{\# of removed instances}} \tag{24}$$

Noise robust models

No learning algorithm is immune to noise but some algorithms perform better than others in the presence of noise (Frénay et al., 2014). Kalapanidas et al. (2003) studied the noise sensitivity of ten different machine learning algorithms against various levels of artificially induced noise. The results show that classifiers are much more noise tolerant than regressors. The *linear regression* algorithm proved to be the regressor least sensitive to noise, whilst the *decision table classifier* proved to be the best performer overall. A similar study (Nettleton et al., 2010) found the *naïve bayes* algorithm to be the most robust to noise.

Online Learning Algorithms

Introduction

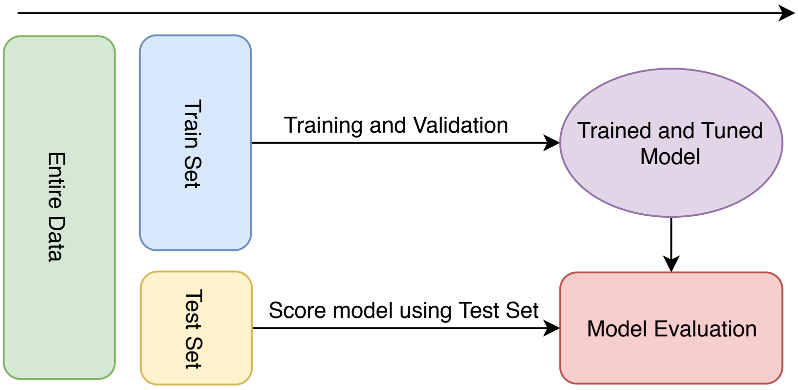


Figure 18: The traditional batch train-test machine learning approach workflow.

In the traditional machine learning approach depicted in Figure 18, we usually have some historical data to train an algorithm on for predicting some future events (Oza, 2005). However, since most data environments are dynamic and will change, the trained model eventually becomes outdated. To tackle this, we usually automate model re-training based on a timeframe (i.e. weekly or daily basis). Although this helps with keeping the model up-to-date, this is still not enough. Even if we consider model re-training on a daily basis, the model would still be at least one day late.

Furthermore, as Pagels (2018) argued, no matter how good a specific model is, it would always be an imperfect representation of the problem. Moreover, to have the best prediction for today, we cannot rely on a model with knowledge about yesterday only. Enter online learning algorithms, a family of techniques that are modelled to consume as much data available (one sample at a time), as fast as possible, while continuously learning and adapting different learning parameters. In the following sections, we will dive deeper into the subject of online learning, and its particular usage.

Why use Online Learning?

While being highly adaptable to dynamic underlying data structures since they make no statistical assumptions on the distribution of the

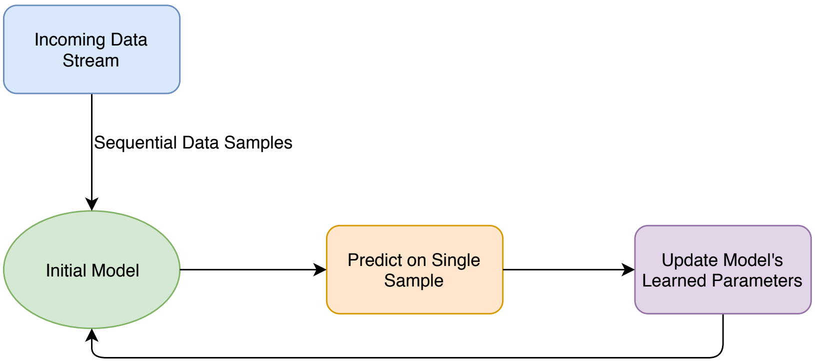


Figure 19: Basic workflow architecture of online learning.

data (Hoi et al., 2014), online learning techniques are also highly data efficient. Since online learning algorithms are only updated using the most recent data samples in the stream (as illustrated in figure 19), such data samples are no longer stored or needed once the algorithm has passed over them, maintaining a much smaller data storage (Oza, 2005). Such algorithms are also very fast since only a single pass on a smaller data set is made, in contrast to the standard approach where the optimisation function needs multiple iterations over the entire dataset. Thus, as argued by Hoi et al. (2018), online learning algorithms scale much better than the traditional approach.

As aforementioned, in offline machine learning, we load an entire dataset in memory, process it, then train a specific model, and then deploy the model into production. However, as more and more data is being generated, especially with the bright spotlight on Big Data, this methodology is proving to be more and more tedious. Some data sets are too large to fit into memory, even with distributed computing measures in place. Thus, online learning can drastically help in this scenario due to its small data storage property, especially when considered as online distributed computing and out-of-core computation (Zhang et al., 2017), which is a huge plus.

To further extract the important usability of online learning, let us, as an analogy, consider the case of an online news portal where news articles are custom and shown to the users based on which categories that respective user usually tends to click. Pretending that a terrible disaster is happening or has happened on one specific day, and the government issues a 24-hour emergency evacuation; therefore, the majority of the user-base would start clicking on this news more and more. With the traditional batch approach, even with a re-training time of 24 hours, the system would fail to push this article to users who typically do not click on domestic affairs articles (i.e. users only interested in sports or entertainment). As a result, the same data content structure will be assumed by the algorithm even though there was a drastic change of events.

In addition to this, given the same batch algorithm, after re-training in the following day, it would now start to suggest this article to a high percentage of the user-base, which by this time, such news might no longer be relevant or applicable.

Another small application resides in the online advertising domain. With different events and occasions happening every day, especially unscheduled or unforeseeable events which go viral, ads must stay relevant all the time to ensure the highest click-rate probability, and thus, must always synchronise to the affairs of the physical world. Ads must be intelligent enough to be aware of the hidden data distribution to adapt to data morphism.

In both examples, a traditional static model will fail due to being too slow to react to the dynamic underlying relationships present in the data. This problem is more formally known as concept drift (Schlimmer and Granger, 1986). The following section will further explain the notion of Concept Drift.

Concept Drift

Concept Drift occurs when the hidden context of the data changes. For instance, weather predictions are highly dependant on the season (the context), and as the seasons change so does the weather (Widmer and Kubat, 1996). As highlighted by Krawczyk and Cano (2018); Gama et al. (2014), based on the distribution drift speed and severity, concept drift can be of four types as depicted in Figure 20:

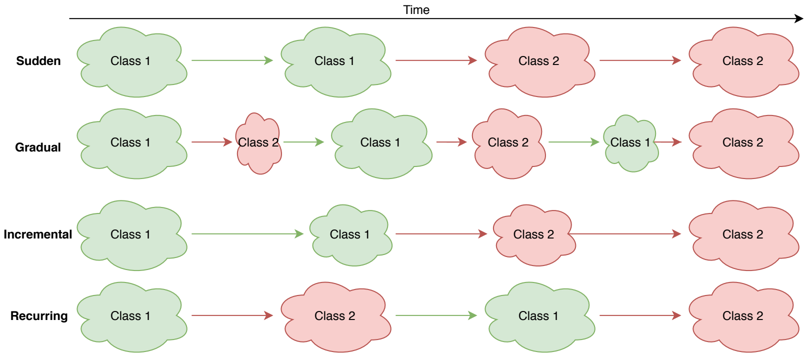


Figure 20: The four types of Concept Drift.

1. **Sudden:** *The data distribution is immediately changed to a different class.*

2. **Gradual:** *The data distribution gradually transitions by having varying proportions of the different classes mixing together over time, until it completely changes to the new class.*

3. **Incremental:** *The data distribution slowly morphs from one class to another.*

4. **Recurring:** *The data distribution periodically transitions between previous classes.*

Dealing with Concept Drift

Thus, to combat this, as the concept drifts, so must the model's transition function that maps the inputs to the outputs. Due to the constant model updates performed through online learning (sample by sample), the transition function would be dynamic and adapts to the changing distribution (Gama et al., 2014; Hoi et al., 2018; Lane and Brodley, 1998). In addition to this, another approach is to have a sliding window that shifts with the data stream. The purpose of this window, as discussed by Wozniak (2011), is to keep a set of instances that offer the best representation of the present data distribution. As newer data samples arrive in the stream, the window slides towards more recent instances, resulting in the exclusion of the oldest samples from the window. Online learning achieves this window technique through the 'forgetting rate' which sets how fast older data is discarded to make room for newer instances.

Forgetting Rate

Even though the design for most online learning algorithms is for fast execution speeds and thus adapted from less complex algorithms, implementation challenges are also present. As argued by Gepperth and Hammer (2016), this leads us to one of the most significant problems in online learning, Catastrophic Interference. The latter happens when the model abruptly forgets knowledge learnt for previous data. Most online learning algorithms have a forgetting rate parameter. This parameter allows the user to decide the speed at which the learning algorithm forgets old data; thus, how much data to retain. Moreover, the correct calibration of this rate is essential and challenging to perfect since a high value would result in catastrophic interference, while a lower value would result in the algorithm not adapting to the incoming samples in the stream. In addition to this, good initialisations are critical in this approach to steer away from slow convergence.

Conclusion

In this chapter, we introduced and discussed the sub-field of online learning algorithms concerning machine learning. Online learning is a highly useful tool that allows us to take machine learning to a whole other level by solving problems that otherwise would seem to be out of our technical ability. With the exponential importance for Big Data analytics, online learning arms us with the capabilities to process high-velocity data while also being fast to adapt to frequent changes in the data due to the ever-increasing data velocity.

Regularisation of Models

Introduction

Regularisation of Models deals with the widespread problem of overfitting in machine learning (ML) models. When a ML model is overfitting it implies that the model has been trained in such a way to perform well on the particular training data but performs badly when using test or unseen data. The overfitting model learns the pattern as well as the noise in the training data. This can be caused when the model has a high variance and when the model is highly complex with respect to the underlying data. The other extremity is underfitting where the model has a high bias. These are illustrated in Figure 21 where the underfitting model is represented by a low degree polynomial ($y = x$) with respect to the underlying data, and the overfitting model is represented by a high degree polynomial ($y = x^n$) (Taschka, 2015). High variance refers to the variability of the model output prediction for test inputs that were not used during training and represent the overfitting case, whereas high bias refers to the errors due to the 'assumptions' of the model that differ from the actual, since the model does not reflect the underlying relationship of the data. The challenge is to find a good bias-variance trade-off.

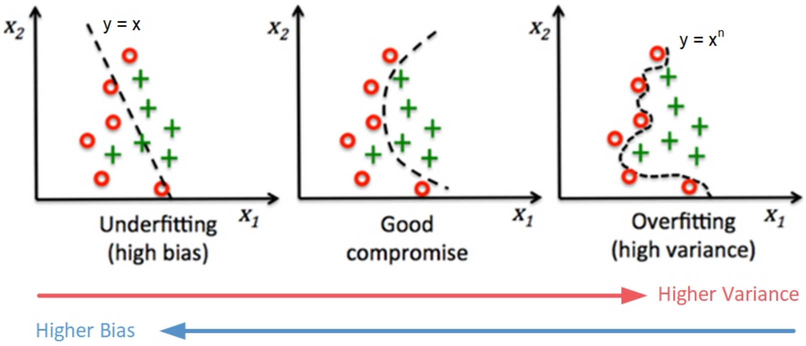


Figure 21: Overfitting and Underfitting: Challenge for a good bias-variance trade-off (Adapted fromTaschka (2015)).

The *compromise* or *generalisation* challenge can be tackled using regularisation techniques. A number of regularisation techniques will be discussed in this chapter including i) L1 regularization, ii) L2 Regularization, iii) Dropout, and iv) Early Stopping.

L2 and L1 regularization

Regularization is a technique that alters the learning algorithm to add noise during the learning process in such a way that the model can generalise better. This generalisation is achieved by introducing a bias so that an overfitting model achieves bias-variance compromise as illustrated in Figure 21. The learning algorithms such as regression and deep learning typically learn using a cost or error function which is minimized such that the model with minimum error is determined. Regularisation uses a *regularisation* parameter λ to add the bias or penalty to the error function as the model complexity increases. Function 25 defines a simplified loss/cost function with regularisation.

$$minimize(costfunction, c(x) + \lambda(penaltyfunction)) \tag{25}$$

Function 25 includes two terms; i) the cost function, and ii) the penalty (regularization) function, where the penalty function is constrained to be less than or equal to a constant, t (Hastie et al., 2001).

To train the model, minimisation is done on both the cost and penalty functions. The cost function, c(x) depends on the training data whereas the regularisation function is independent of the data variable (x_n). Parameter λ is determined empirically or through cross validation, and it is used to control how to balance out the two terms in function 25 by balancing how much the model should learn the training set, and how much bias to add. There are two methods for regularisation that apply a bias which are termed L2 and L1, known as Ridge and Lasso Regression respectively. L2 and L1 penalize weights differently.

L2 Regularisation

L2 regularisation penalizes the $weight^2$. Therefore, considering the cost function for linear regression as an example, function 25 can be re-written as:

$$minimize(\sum_{i=1}^{n}(y_i - w_0 - \sum_{j=1}^{p}(x_ij.w_j))^2 + \lambda\sum_{j=1}^{p}(w_j^2)) \tag{26}$$

Where y_i is the predicted value from which the actual value is subtracted. The weight, (w_0) (intercept in linear regression) is left of out the penalty function. In L2 regularisation λ is a complexity parameter that controls the amount of shrinkage (Hastie et al., 2001). The idea of penalizing by the $weight^2$ is also used in neural networks where it is known as weight decay (Krogh and Hertz, 1992; Moody, 1992). Krogh and Hertz (1992), claim that the "generalisation ability of a Neural Networks depend on a balance between the information in the training example and the complexity of the network", where the complexity is related to the number of weights in the model. L2 regularisation tries to minimize the number of weights, and therefore make the model less complex (decrease the polynomial degree), whilst still minimizing the error.

L1 Regularisation

On the other hand, L1 penalizes on the |weight| (Hastie et al., 2001). Therefore the function is rewritten as:

$$minimize(\sum_{i=1}^{n}(y_i - w_0 - \sum_{j=1}^{p}(x_i j.w_j))^2 + \lambda \sum_{j=1}^{p}(w_j)) \tag{27}$$

The derivative of L2 and L1 regularisation term would result in $2w$ and k (a constant) respectively (considering penalty function only), when computing partial derivatives with respect to the weights, (w_n). Therefore, L2 regularisation removes a percentage from the weight, whereas L1 subtracts a constant from the weight. This creates a significant difference from the Ridge function as it will cause some of the coefficients to be exactly zero for an appropriate value of t (Hastie et al., 2001). L2 regularization pushes less important weights towards zero however it does not force them to be exactly zero.

Ng (2004) considered supervised learning problems where the feature space is made up of many irrelevant features (noisy), and studied L1 and L2 regularization applied on logistic regression methods for preventing overfitting. L1 regularisation cause the weights of some features to go to zero, making it highly suitable for models where many of the features should be ignored. He has found L1 regularisation to be effective in these scenarios and concluded that L1 regularized logistic regression can be effective even if there are exponentially many irrelevant features as there are training examples.

Since L1 regularisation encourages the weight for meaningless features to drop to exactly zero, consequently being removed from the model, it makes this technique suitable for sparse datasets where there could be potentially considerable meaningless features or dimensions. On the other hand L1 regularisation can have the negative affect that it zeros the weight for weakly informative dimensions.

Dropout

Dropout is another regularisation technique that is targeted for neural networks (NN) models and was proposed by Srivastava et al. (2014). The dropout techniques adds bias or noise to the NN model in order to prevent overfitting similar to the L1 and L2 regularisation techniques described earlier. In order to add this bias, the dropout technique removes nodes together with their input and output connections randomly during training. The Dropout technique is illustrated in Figure 22,23.

The random dropout action is performed during the training phase only, and it creates a different thinned NN (Figure 23) for each epoch. Therefore the minimisation, through back propagation, of the NN loss function is only applied to the thinned network, and the inactive

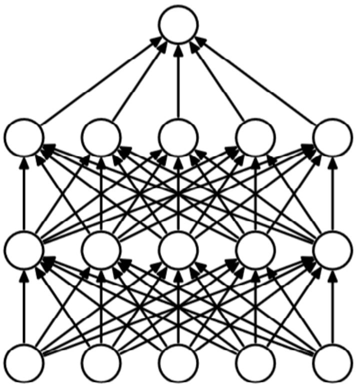


Figure 22: Standard Neural Net (Reproduced from Srivastava et al. (2014)).

neurons do not participate in the training of that epoch. The intensity of the dropout is regulated by hyperparameter p, describing the probability of retaining a unit, where $p = 1$ implies no dropout. Dropout can be applied both for the input layer, and also on each of the hidden layers. Srivastava et al. (2014) have determined that the typical values of p for hidden layers is between $0.5 - 0.8$, whilst for real-valued inputs 0.8 is used. Choosing an incorrect value of p can induce too much bias and may lead to underfitting

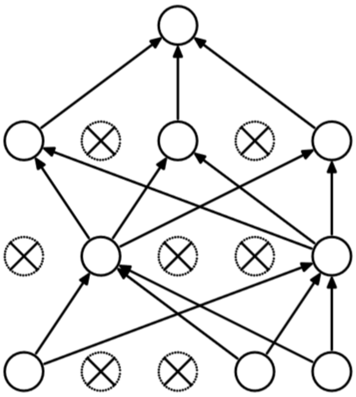


Figure 23: Thinned Neural Net after applying dropouteproduced from Srivastava et al. (2014)).

During each training epoch a *masking* vector is created using the probability p applied on a *Bernoulli Distribution*, where its output is either 1 or 0 to determine which neurons are activated. Therefore considering a layer of n neurons, $(1 - p) \times n$ neurons would be masked at each epoch.

At the end of the training each node would have been trained a different number of times, however each epoch contributes to the same sets of weights. On the other hand, during the testing phases dropout is not utilised and all the neurons are again active. However, before testing, the weights obtained from the different thinned networks are further penalized by multiplying each outgoing weight with the probability p that was used during testing.

Srivastava et al. (2014) et al have determined that dropout was successful in various domains including speech recognition and document classification to mention a few. The authors have concluded that the dropout method is a general technique that can be applied across different domains, however it has the drawback of extending the training time by typically $2 - 3$ times.

Early Stopping

Early Stopping is a method applied to NN where the training model is stopped before the training error is minimized (Sarle, 1995).Figure 24 illustrates a typical plot of the accuracy of model versus the epochs for the 'Training Data' and 'Test Data'. At each iteration the test data is used to evaluate out the model being trained. As can be noticed although the training accuracy continues to increase as cost function is minimized, the 'Test Data' accuracy start dropping after a certain epoch.

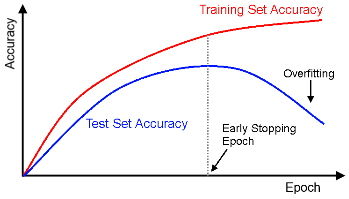


Figure 24: Training Data Accuary Vs Test Data Accuracy

Early Stopping determines the number of epochs a model is allowed to run by evaluating the *training* model after each epoch (or n epochs). The model is stopped if subsequent training results in lower performance. This marks the 'Early Stopping Epoch' as shown in Figure 24. Zur et al. (2009) showed that early stopping reduces the effect of overfitting but is it is not as effective as weight decay using L1 and L2 regularisation. Early Stopping is considered a form of regularisation method since it helps the model from overfitting.

Sample Selection Bias

In traditional statistics, the algorithms assume that the the data samples are being drawn in line with the same distribution, and different classes and values of data should appear with roughly the same frequency that they actually occur in the real world. However, this is rarely the case and results in the data becoming biased, meaning that the method of sample collection favours a particular type of data, skewing the distribution of the values (Cuddeback et al., 2004).

There could be a number of reasons as to why this bias occurs (Tommasi et al., 2017), such as:

1. It might be costly or unpractical to collect certain data.
    - For example, measuring spending habits of large groups of people would be cumbersome, and these people may not necessarily give accurate information.
2. The entity collecting the data only has access to data belonging to a particular class.
    - For example, if a doctor only has access to patients that are sick, a large portion of the sample would represent sick patients, whereas in relation to the total population the number of sick people would be relatively much smaller.
3. Confirmation bias, whereby people tend to recall only examples that confirm their existing beliefs.
4. Incorrect sampling techniques (Marshall, 1996), such as sampling from the top of a list instead of randomly
5. Sampling using results generated from another process.
    - For example, if a system is being trained on detecting fraudulent transactions, and some of these transactions were classified incorrectly by another process, then the model will be trained on false data.

An interesting example that tends to happen within Maltese populations is that related with politics. When a political party passes an online poll regarding a particular issue, the results always tend to be in its favour. This stems from the fact that even though the poll is open to the general public, the outreach of the poll would be much more inclined to reach people within the party - they would have subscribed to social media and news pages, and would regularly follow or check up on their articles and news.

False Information created by Selection Bias

Having selection bias within a dataset can create false information which does not exist in the actual population, and can lead to inaccurate estimates of the relationships between variables (Cuddeback et al., 2004).

For example, assume a college accepts students that either have high math skills or high science skills. Therefore if a student in this college does not have high math skills, then he must instead have high science skills. This means that a negative correlation between maths and science skills has been created, which does not represent the actual population.

How To Prevent Selection Bias

When collecting a sample of data, one should be attentive in order to prevent selection bias occurring in the dataset. This step of preventing bias is important as not only could it skew the results, but might end up rendering the analysis and conclusions gained from the dataset as useless.

Stratified Sampling

This process involves splitting the population into sub-populations before selecting, in order to ensure that an adequate number from each group is selected. For example, splitting a population into age categories and selecting a number from each category (Krautenbacher et al., 2017)

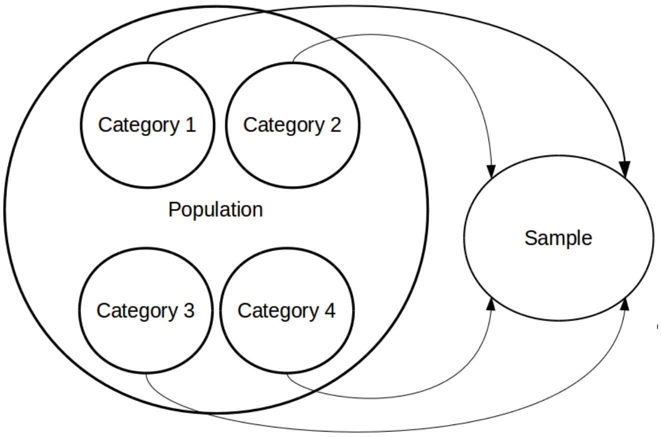


Figure 25: Stratified sampling: splitting the population into categories and selecting accordingly

Not Self Selecting

Draw from a sample that is not self selecting. If the sample is self-selected, people who are willing to volunteer may share similar characteristics (such as be more outgoing and extroverted), which will create a bias.

Analyze Dropouts

Determine factors affecting drop outs and establish that no differences exist between participants and non-participants. If those refusing to participate all belong to a certain category, this will create a bias and results need to be corrected to account for these differences (Alonso et al., 2006).

Sufficient Sample Size

Ensure a large enough sample size. If the sample size is too small, it will make it more difficult to create an accurate representation.

Detecting Selection Bias within a Dataset

Once data has been collected, or when working on a dataset gained from a different source, it is important to determine whether any sample selection bias exists within the dataset before performing analysis or drawing any conclusions. The processes mentioned below are some methods which can be useful in the detecting such bias.

Comparing Distributions

If it is possible to capture information regarding distribution about the whole population, one may then compare this with the sample population. If the distribution if drastically different, it will indicate that a sample selection bias is present.

As a simple example, if the total population contains 50% males and females, whilst your sample contains 75% males and 25% females, it will indicate a selection bias which will skew the results.

Techniques exist to allow the comparison of distributions and measuring the the difference, such as Bayesian Analysis, Kolmogorov–Smirnov test or Chi-Squared test. (Griffin et al., 2013)

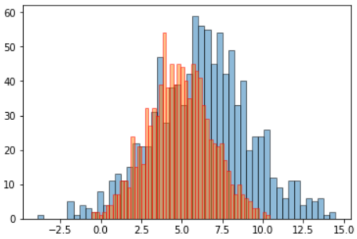


Figure 26: Comparing Distributions

Two Step Estimator

This method, as defined by Heckman (1979), comprises the use of two multiple regression models:

- One model is used to examine the interest of the study.
- The other regression model is used to detect selection bias and to statistically correct the substantive model. The independent variable could be set to represent participation. The dependent variable could represent different related statistics.

Two Step Estimator Example

Imagine a survey aiming to collect information about European citizens quality of life. This would include factors such as job status, salary, level of eduction, country GDP, amongst others.

The first model would simply use these factors to create a model determining each citizens overall quality of life.

The second model would then measure relationships between participation and other variables. If for example a positive correlation exists between level of education and participation, then a bias in the study has occurred whereby those with a lower level of eduction are being underrepresented.

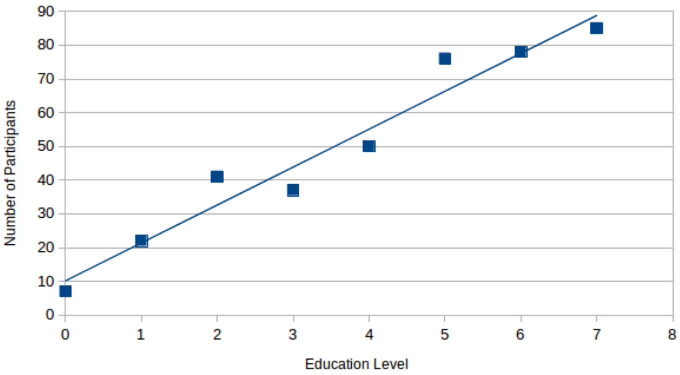


Figure 27: Two Step Estimator: Using regression models to determine correlations between participants and non-participants

How to Deal With Selection Bias

If presented with a dataset which contains selection bias but which cannot be changed or modified due to a number of reasons, then adequate measures should be taken to either mitigate the bias or account for it in the results.

Post-stratification

In an attempt to make the results more representative of the total population, higher weightings are given to the lower class. If say a sample contains 25% women, but the general population has 50%, then you could adjust the estimates by assigning women in the sample higher weights (Holt and Smith, 1979).

As explained by Cortes et al. (2008), the generalization error R on the newly weighted sample is defined as follows:

$$R(h) = \sum_{i=1}^{m} w_i c(h, z_i) \tag{28}$$

Where m is the number of samples, h is the error value, w is the weight, c is the cost function and z is the sample value.

Synthetic Minority Oversampling TEchnique (SMOTE)

Another method of dealing with imbalanced classes is that of generating synthetic data that closely resembles the underrepresented class. In cases where the majority class greatly outweigh the minority class, having a larger dataset that more closely resembles the minority class will balance the results generated.

The way this process works is that new samples are created by finding the midpoint between the line segments joining the existing samples within the minority class.

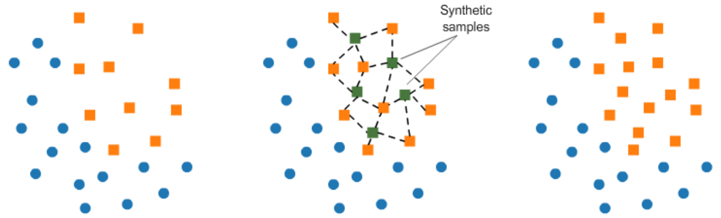


Figure 28: SMOTE: generating synthetic samples of the minority class

SMOTE may also create certain drawbacks. Since the synthetic results are generated on a small set of existing features, it may create an overfitting scenario which might not be able to correctly classify the true outlier nature of the minority class. SMOTE might also skew the distribution of the dataset which could affect certain analysis. The large addition of additional samples could also slow down learning speed.(Weiss et al., 2007)

Studies seem to have shown that using SMOTE techniques to over-represent the minorty class, as opposed to under-representing the majority class by selecting fewer, tends to perform better (Chawla et al., 2002).

Applicability Disclosure

Giving full disclosure: If it is not possible to eliminate the selection bias from the dataset, then it is important to fully disclose who exactly the results and knowledge gained from the data are applicable to.

Transfer Learning

Machine learning algorithms typically operate within isolated problem domains - for example, a model trained to recognize motorcycles would not aim to recognize bicycles. However, this approach is not an accurate representation of human intelligence; in fact, we know that humans can relate knowledge of motorcycles to bicycles (Aytar and Zisserman, 2011).

Unlike classical machine learning approaches, human beings instinctively learn from different sources, drawing from varied past experiences and transferring knowledge across different contexts.

Transfer Learning is the study of extending classical machine learning approaches to apply knowledge acquired from a number of source tasks, to a different but related target task, similar to the way humans learn (Thrun and Pratt, 1998).

Source tasks and target tasks can be related in different ways. For example, if a source task is a *'Dog or Cat'* classifier; we can transfer knowledge to a similar domain but a different task, such as an *'Elephant or Tiger'* classifier; or we can transfer knowledge to the same task in a different domain, such as a *'Cartoon Dog or Cat'* classifier (Torrey and Shavlik, 2009).

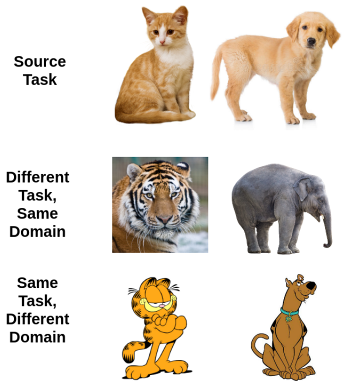


Figure 29: Transfer learning can occur across domains or across tasks. Images reproduced from commons.wikimedia.org (Tiger, Cartoon Cat), imdb.com (Cartoon Dog), pixabay.com (Cat), pnghunter.com (Elephant) and shutterstock.com (Dog).

Transfer Learning Approaches

When labeled data is in short supply or learning is computationally expensive and time consuming, one may select a surrogate task to train for, and transfer knowledge to the intended target task. This is especially beneficial in cases where we do not have enough data, or where training and test data do not possess the same characteristics; containing a different feature space, different distributions, different data sources or even different target labels.

In fact, Transfer Learning approaches can be generalised into three categories, depending on the availability of labeled data for the source and target tasks (Pan and Yang, 2010):

1. **Inductive Transfer Learning;** applicable when data is available for the target domain. Cases where there is no data for the source domain are referred to as self-taught learning; whereas cases where data is also available in the source domain is referred to as multi-task learning.

2. **Transductive Transfer Learning;** applicable when where labeled

data is only available in the source domain. Transductive learning where we assume the same task but different domains is referred to as domain adaptation, while learning a similar domain but a different task is referred to as Sample Bias Selection.

3. **Unsupervised Transfer Learning;** applicable when labelled data is not available neither for the source nor the target task.

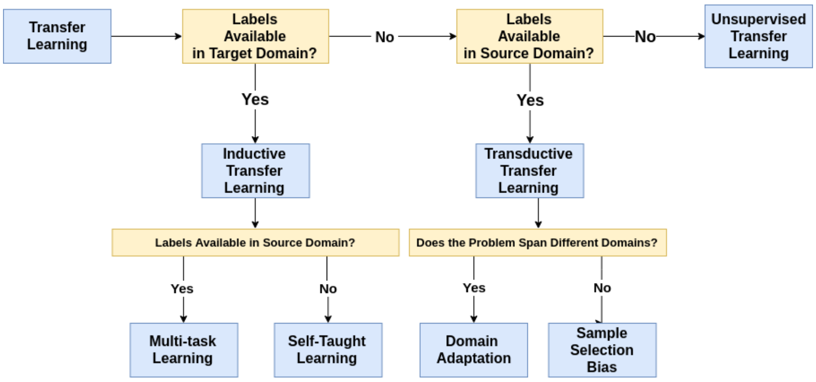

Transfer Learning
Labels Available in Target Domain?
No
Labels Available in Source Domain?
No
Unsupervised Transfer Learning
Yes
Yes
Inductive Transfer Learning
Transductive Transfer Learning
Labels Available in Source Domain?
Does the Problem Span Different Domains?
Yes
No
Yes
No
Multi-task Learning
Self-Taught Learning
Domain Adaptation
Sample Selection Bias

Figure 30: Transfer learning approaches vary depending on the availability of labeled data for the source and target tasks.

The above approaches vary on what data is available for the learning task - conversely, this also affects what knowledge can be transferred across tasks.

For example, it is sometimes impossible to apply a source task's data directly to a target task. Instead we can attribute selected instances or features from the source task, which are in turn re-weighted and re-distributed to fit the target task. These methods are referred to as Instance Transfer and Feature-representation Transfer approaches.

In cases where labels are available for either source tasks or target tasks (both inductive and transductive transfer learning), some forms of instance-transfer and feature-representation transfer are possible. Moreover, unsupervised transfer learning techniques are limited to feature-representation transfer (Pan and Yang, 2010).

Other examples of transfer learning include broader knowledge transfers such as parameter transfer (or model transfer), where full or parts of parameterized models are transferred from the source task to the target task; or relational-knowledge transfer, where relationships between data points within the source task are transferred to re-distribute data within the target task (Cook et al., 2013). Such transfer is only possible where the target task's labels are available.

	Inductive Transfer Learning	Transductive Transfer Learning	Unsupervised Transfer Learning
Instance Transfer	X	X	
Feature-representation Transfer	X	X	X
Parameter Transfer	X		
Relational-knowledge Transfer	X		

Table 8: Transfer types by Approach

Deep Transfer Learning Techniques

The above approaches have manifested themselves in two major techniques of transfer learning within deep learning. Multi-layer neural networks are built in such a way that lower-level layers address generic features and higher-level layers address more complex and specific features. The initial layers for a group of related tasks are similar if not identical; this characteristic can be exploited to enable the transfer of knowledge between models (Yosinski et al., 2014).

Off-the shelf models, or pre-trained models, involve training one or more multi-layer networks and adapting parts of the model to another target task. The source models act as a form of pre-training, to learn shallow parts of the problem, while the target learning task is solely limited to the final layer of the model. In fact, one application of pre-trained models can be seen as a feature-selection process (Zhu et al., 2018).

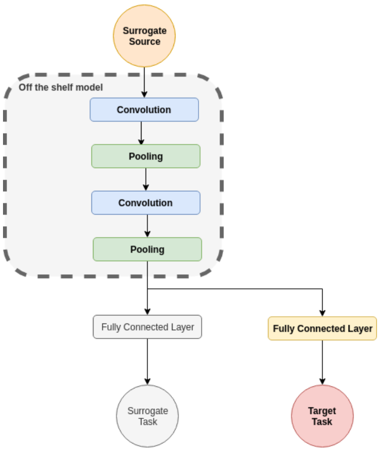

Surrogate Source
Off the shelf model
Convolution
Pooling
Convolution
Pooling
Fully Connected Layer
Fully Connected Layer
Surrogate Task
Target Task

Figure 31: In off-the-shelf pre-trained models, we replace the final layer of the neural network.

Another approach is to use transfer learning to replace selective parts of the model, rather than just the final layer, this is referred to as model Fine-tuning.

With Fine-tuning techniques, we pre-train a model using a number of different source tasks, and then re-train the model for the target task; however the target training is done selectively, selecting which layers are to be frozen (and thus inherited from the source tasks) or fine-tuned (to be updated during the backpropagation process). We can define a metric to decide the learning rate for each layer, creating a variable degree of freezing and fine-tuning (Chu et al., 2016).

The success of transferability is highly dependent on source task's level of specialization, successful transfer learning projects work on generalized source tasks. As a matter of fact, transfer learning across tasks which are too dissimilar as a result of being too specific would result in a negative learning (Rosenstein et al., 2005).

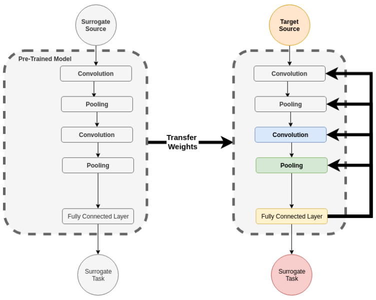

Surrogate Source
Pre-Trained Model
Convolution
Pooling
Convolution
Pooling
Fully Connected Layer
Surrogate Task
Transfer Weights
Target Source
Convolution
Pooling
Convolution
Pooling
Fully Connected Layer
Surrogate Task

Figure 32: In Pre-trained model fine tuning, we use the source model to initialize the neural network, and fine-tune it using backpropogation.

Applications and Improvements

Notwithstanding savings on time and computational resources, the above techniques for transfer learning in multi-layer neural networks frequently render better results when compared to training the neural networks from scratch (Yosinski et al., 2014). This is especially the case in fields where data collection and annotation is notoriously difficult, such as computer vision and natural language processing.

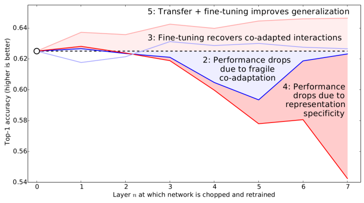


Figure 33: The performance gained form fine-tuning a neural network is only relative to the layer generalization and specificity of each distinct layer (Yosinski et al., 2014).

The use of pre-trained models machine learning has achieved new state-of-the-art results in several tasks; within the field of computer vision, in object detection (He et al., 2017), semantic segmentation (Zhao et al., 2016), human pose estimation (Papandreou et al., 2017) and action recognition (Carreira and Zisserman, 2017) ; and within the field of natural language processing in text classification, question answering, language inference and conference resolution amongst others (Howard and Ruder, 2018) (Joshi et al., 2018).

Bibliography

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

A. Alonso, M. Seguí-Gømez, and J. de Irala. Predictors of follow-up and assessment of selection bias from dropouts using inverse probability weighting in a cohort of university graduates. *European Journal of Epidemiology*, 21(5):351–358, 2006.

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.

Y. Aytar and A. Zisserman. Tabula rasa: Model transfer for object category detection. In *IEEE International Conference on Computer Vision*, 2011.

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.

Carla E Brodley and Mark A Friedl. Identifying mislabeled training data. *Journal of artificial intelligence research*, 11:131–167, 1999.

João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750, 2017.

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.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over–sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. Best practices for fine-tuning visual classifiers to new domains. In *ECCV Workshops*, 2016.

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

Diane Cook, Kyle D. Feuz, and Narayanan C. Krishnan. Transfer learning for activity recognition: a survey. *Knowledge and Information Systems*, 36(3):537–556, Sep 2013.

Corinna Cortes, Mehryar Mohri, Michael Riley, and Afshin Rostamizadeh. Sample selection bias correction theory. In *Proceedings of the 19th international conference on Algorithmic Learning Theory*, ALT '08, pages 38–53. Springer-Verlag Berlin, Heidelberg, 2008.

Gary Cuddeback, Elizabeth Wilson, John G. Orme, and Terri Combs-Orme. Detecting and statistically correcting sample selection bias. *Journal of Social Service Research*, 30(3):19–33, 2004.

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.

Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2014.

Benoît Frénay, Ata Kabán, et al. A comprehensive introduction to label noise. In *ESANN*, 2014.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, March 2014. ISSN 0360-0300.

Luís Paulo F. Garćia, André C. P. L. F. de Carvalho, and Ana C. Lorena. Noisy data set identification. In Jeng-Shyang Pan, Marios M. Polycarpou, Michał Woźniak, André C. P. L. F. de Carvalho, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent*

Systems, pages 629–638, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40846-5.

Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016. URL `https://hal.archives-ouvertes.fr/hal-01418129`.

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.

J. E. Griffin, M. Kolossiatis, and M. F. J. Steel. Comparing distributions by using dependent normalized random-measure mixtures. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3): 499–529, 2013.

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.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

J. J. Heckman. Sample selection bias as a specification error. *Econometrica*, 47(1):153–161, 1979.

Ray J Hickey. Noise modelling and evaluating learning from examples. *Artificial Intelligence*, 82(1-2):157–179, 1996.

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

Steven C. H. Hoi, Jialei Wang, and Peilin Zhao. LIBOL: A Library for Online Learning Algorithms. *J. Mach. Learn. Res.*, 15(1):495–499, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2627450.

Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online Learning: A Comprehensive Survey. *arXiv:1802.02871 [cs]*, February 2018. URL http://arxiv.org/abs/1802.02871. arXiv: 1802.02871.

D. Holt and T. M. F. Smith. Post stratification. *Journal of the Royal Statistical Society*, 142(1):33–46, 1979.

Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

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.

Vidur Joshi, Matthew Peters, and Mark Hopkins. Extending a parser to distant domains using a few dozen partially annotated examples. *CoRR*, abs/1805.06556, 2018.

Elias Kalapanidas, Nikolaos Avouris, Marian Craciun, and Daniel Neagu. Machine learning algorithms: a study on noise sensitivity. In *Proc. 1st Balcan Conference in Informatics*, pages 356–365, 2003.

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.

Norbert Krautenbacher, Fabian J. Theis, and Christiane Fuchs. Correcting classifiers for sample selection bias in two–phase case–control studies. *Computational and Mathematical Methods in Medicine*, 2017, 2017.

Bartosz Krawczyk and Alberto Cano. Online ensemble learning with abstaining classifiers for drifting and noisy data streams. *Applied Soft Computing*, 68:677–692, July 2018. ISSN 1568-4946. URL http://www.sciencedirect.com/science/article/pii/S1568494617307238.

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.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

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.

Terran Lane and Carla E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 259–263. AAAI Press, 1998. URL `http://dl.acm.org/citation.cfm?id=3000292.3000339`.

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.

Martin N Marshall. Sampling for qualitative research. *Family Practice*, 13(6):522–526, 1996.

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

John E Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In *Advances in neural information processing systems*, pages 847–854, 1992.

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.

David F Nettleton, Albert Orriols-Puig, and Albert Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial intelligence review*, 33(4):275–306, 2010.

Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

N. C. Oza. Online bagging and boosting. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345 Vol. 3, Oct 2005.

Max Pagels. What is online machine learning?, April 2018. URL `https://medium.com/value-stream-design/online-machine-learning-515556ff72c5`.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, and Kevin P. Murphy. Towards accurate multi-person pose estimation in the wild. *CoRR*, abs/1701.01779, 2017.

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.

Michael T. Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G. Dietterich. To transfer or not to transfer. In *In NIPSâĂŹ05 Workshop, Inductive Transfer: 10 Years Later*, 2005.

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.

Warren S. Sarle. Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, pages 352–360, 1995.

Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, September 1986. ISSN 1573-0565. URL `https://doi.org/10.1007/BF00116895`.

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.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 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.

Sebastian Taschka. *Python Machine Learning*. Packt Publishing, Birmingham, 2015.

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

Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. ISBN 0-7923-8047-9.

Tatiana Tommasi, Novi Patricia, Barbara Caputo, and Tinne Tuytelaars. A deeper look at dataset bias. In *Domain Adaptation in Computer Vision Applications*, pages 37–55. Springer, Cham, 2017.

Lisa Torrey and Jude Shavlik. Transfer learning. In *Transfer Learning*, 2009.

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.

Sofie Verbaeten and Anneleen Van Assche. Ensemble methods for noise elimination in classification problems. In *International Workshop on Multiple Classifier Systems*, pages 317–325. Springer, 2003.

Gary M. Weiss, Kate McCarthy, and Bibi Zabar. Cost–sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? In *DMIN*, 2007.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, April 1996. ISSN 1573-0565. URL `https://doi.org/10.1007/BF00116900`.

Michal Wozniak. A hybrid decision tree training method using data streams. *Knowledge and Information Systems*, 29(2):335–347, November 2011. ISSN 0219-3116. URL `https://doi.org/10.1007/s10115-010-0345-5`.

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.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

Wenpeng Zhang, Peilin Zhao, Wenwu Zhu, Steven C. H. Hoi, and Tong Zhang. Projection-free Distributed Online Learning in Networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4054–4062, International Convention Centre, Sydney, Australia, August 2017. PMLR. URL `http://proceedings.mlr.press/v70/zhang17g.html`.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial intelligence review*, 22(3):177–210, 2004.

Yi Zhu, Jia Xue, and Shawn D. Newsam. Gated transfer network for transfer learning. *CoRR*, abs/1810.12521, 2018.

Richard M Zur, Yulei Jiang, Lorenzo L Pesce, and Karen Drukker. Noise injection for training artificial neural networks: A comparison with weight decay and early stopping. *Medical physics*, 36(10):4810–4818, 2009.

Index

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

classification, 11–14
  binary classification, 12
    binary classifiers, 12
  classifiers, 11
  multiclass classification, 13
    all-vs-all, 14
    error-correcting output-coding, 14
    generalized coding, 14
    hierarchical classification, 14
    neural networks, 13
    one-versus-all, 13
    support vector machines, 13
  probabilistic classification, 12
    discriminative model, 12
    generative model, 12
  qualitative predictions, 11
  quantitative predictions, 11
  regression, 11
  supervised learning, 11
  unsupervised learning, 12
confusion matrix, 15–19
  1-vs-1, 17
  1-vs-rest, 17
  accuracy, 16
  error rate, 17
  f-score, 17
    f-measure, 17
    f1-score, 17
  false discovery rate, 17
  false negative rate, 17
  false omission rate, 17
  false positive rate, 17
  negative predictive value, 16
  positive predictive value, 17
    precision, 17
  true negative rate, 16
    specificity, 16
  true positive rate, 16
    recall, 16
    sensitivity, 16
cross-validation, 19–22
  holdout, 20
  k-fold, 20
  leave-one-out, 20
  leave-p-out, 19

Dropout, 33

Early Stopping, 34

false negatives, 15
false positives, 16

L2 and L1 regularization, 32
  L1 Regularisation, 33
  L2 Regularisation, 32
Lasso Regression, 32
license, 1

noise in datasets, 23–26
  artificial, 25
  attribute noise, 23–25
  class noise, 23–25
  cleansing, 24
  cross-validated committees filters, 25
  effects, 23
  ensemble filters, 25
  iterative-partitioning filters, 25
  Noise Elimination Precision, 26
  noise robust models, 26
  Type I errors, 25
  Type II errors, 26

online-learning, 27
  concept drift, 29
  forgetting rate, 30
  gradual drift, 29
  incremental drift, 29
  recurring drift, 29
  sudden drift, 29
overfitting, 21

Regularisation of Models, 34
regularisation-of-models, 31
repeated random sub-sampling, 20
Ridge Regression, 32

sample selection bias, 35
  confirmation bias, 35
  distribution, 37
  smote, 39
  stratification, 36, 38
  two-step estimator, 37

Thinned Neural Network, 33
transfer learning, 41–43
  applications and improvements, 43
  deep transfer learning techniques, 43
  transfer learning approaches, 41
true negatives, 16
true positives, 15

underfitting, 21