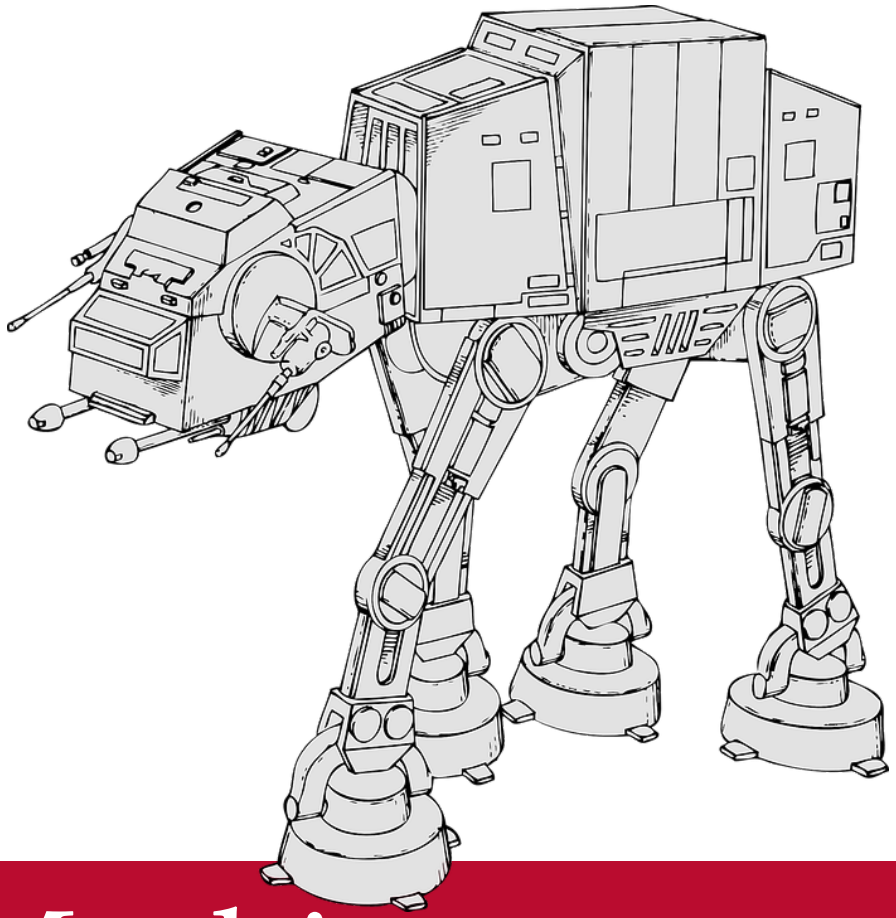
*A Machine Learning Study Guide*



# Machine Learning Handbook

*The Definitive Guide*

*ICS5110, class of 2018/9*

L-Università ta' Malta




JEAN-PAUL EBEJER, DYLAN SEYCHELL, LARA MARIE DEMAJO, DANIEL FARRUGIA, KEITH MINTOFF, FRANCO CASSAR MANGHI, DAVID FARRUGIA, IVAN SALOMONE, ANDREW CACHIA, JAKE J. DALLI, JOSEPH AZZOPARDI, NATALIA MALLIA, MARK MUSCAT, STEFAN CASSAR, PATRICK BEZZINA, DYLAN VASSALLO, BRIAN PACE, GEORGE EDUARDO BUCKUP SULZBECK, KATRIN JANSEN, ALIIA EROFEEVA, MICHAEL FILLETTI, NEIL FARRUGIA, CLAIRE GALEA, ALBERT BEZZINA, PHILIP MIFSUD, NEIL MICALLEF, MATTHEW FARRUGIA, SAMUEL MIZZI, AARON GRECH, PETER INCORVAJA



*First printing, January 2019*

# *Contents*

# *Introduction*

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

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

# *Activation Functions*

Caterini (2018) defined artificial neural networks as "a model that would imitate the function of the human brain—a set of neurons joined together by a set of connections. Neurons, in this context, are composed of a weighted sum of their inputs followed by a nonlinear function, which is also known as an activation function."

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

## *Step Function*

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

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

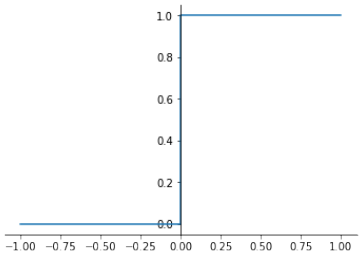



Figure 1: A graph of the step function.

The Heavside step function, visualised in figure 1 and defined by equation 1, is one of the simplest activation functions that can be used in a neural network. This function returns 0 if the input of a node is less than a predetermined threshold (typically 0), or otherwise it returns 1 if the output of the node is greater than or equal to the threshold. This activation function was first used in a machine learning context by Rosenblatt (1957) in his seminal work describing the perceptron, the precursor to the modern day neural network.

Nowadays, the step function is seldom used in practice as it cannot be used to classify more than one class. Furthermore, since the derivative of this function is 0 , as defined by equation 2, gradient descent algorithms are not be able to progressively update the weights of a network that makes use of this function (Snyman, 2005).

### *Linear Functions*

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

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

A linear activation function, is any function in the format of equation 3, where $a, b \in \mathbb{R}$. This function seeks to solve some of the shortcomings of the step function. The output produced by a linear activation function is proportional to the input. This property means that linear activation functions can be used for multi-class problems. However, linear functions can only be utilised on problems that are linearly separable and can also run into problems with gradient descent algorithms, as the derivative of a linear function is a constant, as seen in equation 4. Additionally, since the output of the linear function is not bound to any range, it could be susceptible to a common problem when training deep neural networks called the exploding gradient problem, which can make learning unstable (Goodfellow et al., 2016).

### *Sigmoid Function*

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

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

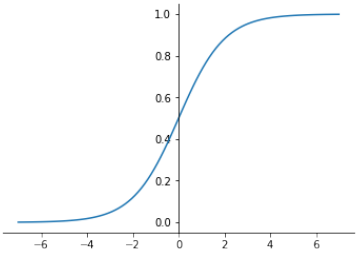



Figure 2: A graph of the sigmoid function.

The sigmoid function or logistic function, visualised in figure 2 and represented by equation 5, is one of the most commonly used activation functions in neural networks, because of its simplicity and desirable properties. The use of this function in neural networks was first introduced by Rumelhart et al. (1986), in one of the most important papers in the field of machine learning, which described the back-propagation algorithm and the introduction of hidden layers, giving rise to modern day neural networks. The values produced by the sigmoid function are bound between 0 and 1, both not inclusive, which help manage the exploding gradient problem. The derivative of this function, represented by equation 6, produces a very steep gradient for a relatively small range of values, typically in the range of $-2$ to 2. This means that for most inputs that the function receives it will return values that are very close to either 0 or 1.

On the other hand, this last property makes the sigmoid function very susceptible to the vanishing gradient problem (Bengio et al., 1994). When observing the shape of the sigmoid function we see that towards the ends of the curve, the function becomes very unresponsive to changes in the input. In other words, the gradient of the function for large inputs becomes very close to 0. This can become very problematic for neural networks that are very deep in design, such as recurrent neural networks (RNNs).To address this

problems in RNNs Long Short-Term Memory (LSTM) units where introduced as a variant of the traditional RNN architecture (Hochreiter and Schmidhuber, 1997).

### *Hyperbolic Tangent*

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

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

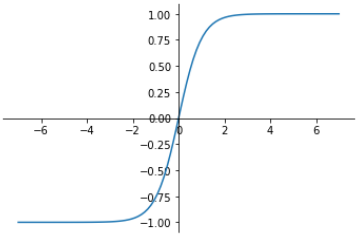



Figure 3: A graph of the hyperbolic tangent (tanh) function.

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*

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

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

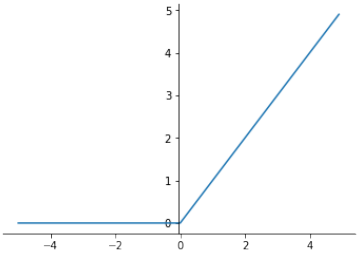



Figure 4: A graph of the ReLU function.

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. (2012a) 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.

# *Bayesian Models in Machine Learning*

Many machine learning techniques employ probability theory to make the best decisions given some data. Probabilistic approaches are often also summarised under the term “Bayesian approaches" (Murphy, 2012). The following chapter will provide an overview of the underlying theorem as well as the most popular classifier based on said theorem.

## *Bayes' Theorem*

Bayes' Theorem describes how to calculate the so-called posterior probability of an event $A$ given our observed evidence $B$, that is $P(A|B)$. It is derived from the definition of conditional probabilities (11). The probability of an event $B$ to occur given that the event $A$ already has occurred is defined as the probability of the intersection of the two events $A$ and $B$ divided by the probability of the event $A$ (Murphy, 2012):

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

The probability of the intersection of the two events $A$ and $B$ can be rewritten as (12). However, since $P(A \cap B)$ is equal to $P(B \cap A)$, it can be rewritten as (13), as well (Manning et al., 2009).

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

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

Consequently, Bayes' Theorem can be derived as follows:

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

$P(A|B)$ is the aforementioned posterior probability, $P(B|A)$ the likelihood of event $B$ given event $A$, $P(A)$ the prior probability of event $A$, and $P(B)$ the so-called marginal, that is the likelihood of an independent occurrence of event $B$ (Manning et al., 2009; van Rijsbergen, 1979).

### *Naïve Bayes Classification Algorithm*

Intuitively, Bayes' Theorem can be used to calculate the probability of an hypothesis being true given some observed evidence (Manning et al., 2009; Jurafsky and Martin, 2018; van Rijsbergen, 1979) - it is a structured way of integrating all available information into an estimation of the truth of that hypothesis. However, it can also be used as a classification method (Manning et al., 2009; Jurafsky and Martin, 2018; Murphy, 2012; van Rijsbergen, 1979): The instance to be classified (such as an email, a web page, or a newspaper article) is taken to be the "observed evidence" for that instance to belong to a certain class. For example, the content of an email is taken to be the information available to classify that email as spam or ham. Bayes' Theorem is applied to the conditional probability $P(c|d)$ of a given document $d$ belonging to a certain class $c$, resulting in (15). The denominator $P(d)$ can be dropped, since the prior probability of a document to occur is the same for all possible classes and does not have any effect on the final classification decision (Jurafsky and Martin, 2018). Ultimately, the result is (16).

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

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

(16) can be read as the probability of a given document $d$ belonging to a certain class $c$ being proportional to the prior probability of a document occurring in class $c$ multiplied by the probability of that class $c$ having generated a document consisting of all the terms $t_k$ in $d$. Naïve Bayes classifiers are also called generative classifiers (Murphy, 2012): Given a specific observation $d$, they return the class $c$ that is most likely to have generated said observation - also called the maximum a posteriori (MAP) class.

The two parameters needed to calculate $P(c|d)$, $P(c)$ and $P(t_k|c)$, are obtained via approximations from the training set, based on maximum likelihood estimations (MLE) (Manning et al., 2009; Jurafsky and Martin, 2018). For the prior probability $P(c)$, the estimate is as follows:

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

which is the number of documents in the training set belonging to class $c$ divided by the total amount of documents in the training set (Manning et al., 2009; Jurafsky and Martin, 2018). Similarly, the probability of a term $t_k$ occurring in a document of class $c$ is obtained like this:

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

which is the number of times a term $T$ appears in documents of class $c$ divided by the number of times that term appears in the vocabulary $V$, i.e. in all documents of the training set (Jurafsky and Martin, 2018; Manning et al., 2009).

These approximations pose two problems. First of all, the multiplication of all the conditional probabilities $P(t_k|c)$ may result in a so-called floating-point underflow (Manning et al., 2009). In order to avoid this, Naïve Bayes classifiers usually do not calculate the product over $P(t_k|c)$, but the sum over the log values of $P(t_k|c)$ (see (19)). Using the log values instead of the original ones does not interfere with the final classification decision in any way: The class with the highest log probability is still the one most likely to have generated the given document.

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

The second problem encountered by the estimations above is data sparseness, resulting in terms not occurring in documents of particular classes in the training set. The preferred solution is to eliminate the resulting zeros by using add-one or Laplace-smoothing (see (20), with $B' = |V|$, the length of the vocabulary), assuming that each and every relevant term occurs at least once in a corpus (Manning et al., 2009).

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

### *Naïve Assumptions*

While the calculations above appear to be straightforward, they pose a problem in terms of their feasibility, especially regarding big sets of data. Relating individual probabilities to all terms of a document âĂŞ even to those occurring multiple times in different positions âĂŞ increases the number of parameters dramatically and requires huge sets of training data, which are almost never available. NaÃŕve Bayes offers solutions to this problem via two rather naïve assumptions giving the classifier its name (Manning et al., 2009; Jurafsky and Martin, 2018).

Hence, Naïve Bayes classifiers assume the bag of words model for all terms in a document, that is they assume the probability of a term to be the same, regardless of its position in the document. This is also called the positional independence assumption (Manning et al., 2009; Jurafsky and Martin, 2018).

The second assumption is called the conditional independence assumption. It simplifies the calculation further by proposing that all terms in a document are independent of one another, i.e. the occurrence of a specific term does not make it any more likely for another, possibly related term to appear in the same document. Obviously, the exact opposite is true - in reality, the terms in a document are indeed

dependent on one another. However, Naïve Bayes classifiers still offer accurate classification decisions, regardless of the inaccuracies in their probability estimations (Manning et al., 2009; Jurafsky and Martin, 2018).

### *Conclusion*

The classification method described just now is just one of the versions of Bayesian classifiers, called the multinomial Naïve Bayes classifier. One alternative is the Bernoulli model (Manning et al., 2009). In order to arrive at a classification decision, the Bernoulli model uses only binary information regarding the occurrence (1) or absence (0) of a term in a document. This often leads to errors regarding the classification of long documents, since the actual number of occurrences of a term are not taken into account, which is why the multinomial model is preferred.

# *Classification*

Classification and regression are two techniques which are nowadays being used to extract knowledge and perform predictions on the huge amounts of available data (Sagar, 2017). These techniques predict values which may be either *quantitative* - involving discrete numerical values, or *qualitative* - with values within a particular class or label (James et al., 2013). 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.

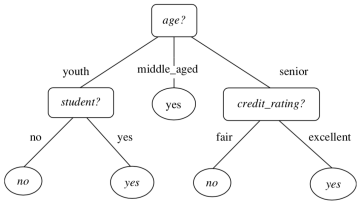



Figure 5: A binary Decision Tree example which determines whether a customer will purchase a computer or not. Reproduced from Jiawei Han, Micheline Kamber (2011).

### *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 21, or by modelling the joint distribution over $x$ and $C_k$ as in Equation 22.

$$P(C_k|x) = P(C_k) \cdot \frac{P(x|C_k)}{P(x)} \tag{21}$$

$$P(C_k|x) \propto P(C_k) \cdot P(x|C_k) \tag{22}$$

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 $C_k$.

$$P(C_k|x) = \frac{1}{1 + e^{-\theta^T X}} \tag{23}$$

One popular discriminative model is the Logistic Regression Model, explained by Equation 23, in which $\theta$ 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, 2014). 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.

# *Concept Drift*

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

Classification techniques are within the most utilised algorithms in order to extract hidden patterns. One major limitation of such classification models is the assumption that the underlying data concepts will not change over time. This assumption poses a major classification limitation as in the real-world observations and classes do change over time. A real-world example is a spam filter where spammers continuously find new ways of how to send spam to increase their success rate. Thus, a classification technique will decrease it accuracy over time as the observations, statistics and their classifications will not hold forever. This problem is referred to as concept drift. This increases machine learning model's complexity, mostly in those cases where new data is treated as an important contributor to the final concept classification (Bishop, 2006; Tsymbal, 2004).

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

## *Types of concept drift*

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

1. Sudden drift: This kind of concept drift, happens in those cases where the concept drift happens abruptly. An example of such drift is season changes in sales (Tsymbal, 2004).

2. Incremental: The drift happens whenever variables change their value over time. An example of this type of drift is the effect of prices due to inflation where prices will keep rising overtime (Tsymbal, 2004).

3. Gradual drift: Although its similar to incremental, a gradual drift refers to those cases where over time the concept changes definition. Example in a fraud detection, a fraudulent may transition from one category to another and back (Tsymbal, 2004).

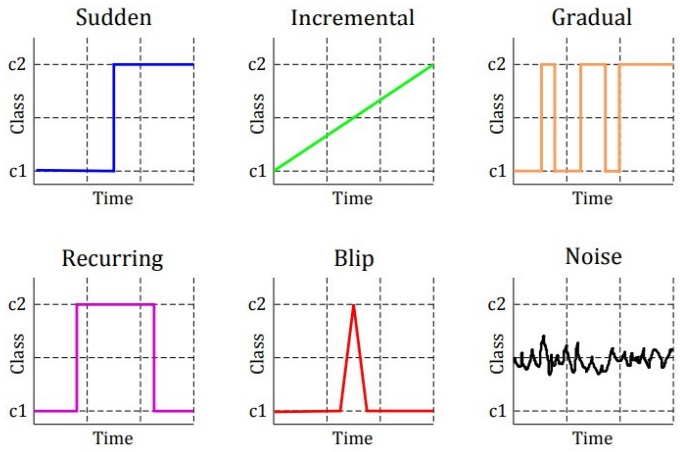



Figure 7: Different types of concept drift excluding noise. (Gómez Losada, 2017)

4. Recurring drift: This refers to those instances where the classification changes over time and alternates between the classes. A real-world example is the usage of a particular mobile application where a user may use the application differently between work and home (Tsymbal, 2004).

5. Blip drift: Is an abrupt drift, which only lasts for a very short duration. There has been number of debates over blip if it qualifies as noise or is ignore or the model is adjusted to correctly classify the blip drift. Nonetheless, adjusting the model may harm future classifications as following the blip the old model must be restored. An example of blip drift is Cyber Monday where on-line shops might experience an increase in their sales or page hits which is an abnormal pattern (Tsymbal, 2004).

6. Noise: This refers to insignificant fluctuations which are not connected or related to the target concept. If noise is present, it must be filtered out as it's not a concept drift and might conflict with the classification model (Tsymbal, 2004).

### *Detecting Concept drift*

The aim of drift detectors is to raise alarms when concept drift occurs in order to update or rebuild the model. This is a crucial step in models as it increases the robustness and model accuracy. Two simple drift detectors are the Cumulated Sum (CUSUM) (Ross et al., 2012) and the Geometric Moving Average (GMA) (Ross et al., 2012). The main difference between these detectors is that CUSUM raises an alarm when the sum of the inputted data is not zero whilst GMA considers the weighted average in a window and raised the alarm if the average is higher than a given threshold.

### *Dealing with concept drift*

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

1. Instance selection: Is the most common technique for handling concept drift and its aim is to identify those instances that are truly related to the current concept. This is achieved by creating a form of a moving window which considers recent information whilst utilising learnt prediction concepts in the immediate future. Moreover, due to the fact that the classification will be utilising recent data, the model will be automatically excluding old data (Webb et al., 2016).

2. Instance weighting: In this technique instances are weighted depending on their age and relevance to the concept. This allows gradual drift to start being properly handled by the model as new data would have more weight whilst past data would start losing its classification power (Webb et al., 2016).

3. Ensemble learning: allows one to obtain better classification results by processing the data through multiple algorithms. The resultant of each algorithm is combined and results would be considered in the classification decision. Moreover, one could also introduce weights, which weighting is assigned to the different algorithm based on their classification power and accuracy. One downside of ensemble learning is that the data needs to be processed by every algorithm selected for the model classification (Webb et al., 2016).

### *Conclusions*

In this chapter, we introduced and discussed concept drift. Machine learning is increasing its popularity and more real-world application are being released like self-driving cars and fraud detection, which applications do suffer from concept drift. This is due to the fact that data in the real-world is not static, thus concept drift will be residing in such applications and must be dealt with. Moreover, one major limitation in concept drift, is the lack of real-data during the experimentation and testing. This is due to the fact that drift data has to be generated artificially, thus, limiting the experiment efficiency. Example, even though Tesla did rigorous testing and were given the go-ahead of releasing self-driving cars, a number of deadly crashed emerged. Thus, concept drift is an important aspect in machine learning as one needs to enable the model to adapt to new concepts or raise alarms if drastic shifts are encountered (Stewart, 2018).

# *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 (Jurafsky and Martin, 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 $\Xi$ 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, 2015b). 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 24-30) 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 24). 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.

$$ACC = \frac{|TP \cup TN|}{|TP \cup FP \cup TN \cup FN|} \tag{24}$$

*Negative predictive value* (NPV) is the ratio of the correct negative predictions from the total negative predictions (Equation 25).

$$NPV = \frac{|TN|}{|TN \cup FN|} \tag{25}$$

*True negative rate* (TNR), or *specificity*, is the ratio of the correct negative predictions from the total true negatives (Equation 26).

$$TNR = \frac{|TN|}{|TN \cup FP|} \tag{26}$$

*True positive rate* (TPR), also called *sensitivity* or *recall*, is the ratio of the correct positive predictions from the total true positives (Equation 27).

$$TPR = \frac{|TP|}{|TP \cup FN|} \tag{27}$$

Sensitivity and specificity can be combined into a single metric (Equation 28). These metrics are often used in domains in which minority classes are important (Kuhn and Johnson, 2013). For example, the sensitivity of a medical classifier (El-Dahshan et al., 2010) measures how many patients with the condition tested positive, and specificity measures how many did not have the condition and tested negative.

$$Sensitivity \times Specificity = \frac{|TP| \times |TN|}{|TP \cup FN| \times |TN \cup FP|} \tag{28}$$

*Positive predictive value* (PPV), or *precision*, is the ratio of the correct positive predictions from the total positive predictions (Equation 29). The difference between accuracy and precision is depicted in Figure 8.

$$PPV = \frac{|TP|}{|TP \cup FP|} \tag{29}$$

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

$$F\text{-}score = 2 \times \frac{PPV \times TPR}{PPV + TPR} \tag{30}$$

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.

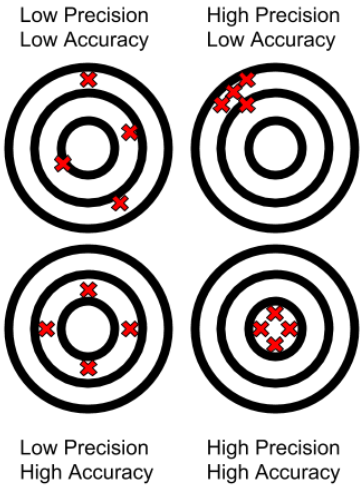



Figure 8: Accuracy vs Precision.

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

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

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

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

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

Using all metrics could be counterproductive due to information redundancy, but none of the metrics is enough on its own (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 9 were trained on the *wines* dataset included with Scikit-Learn.

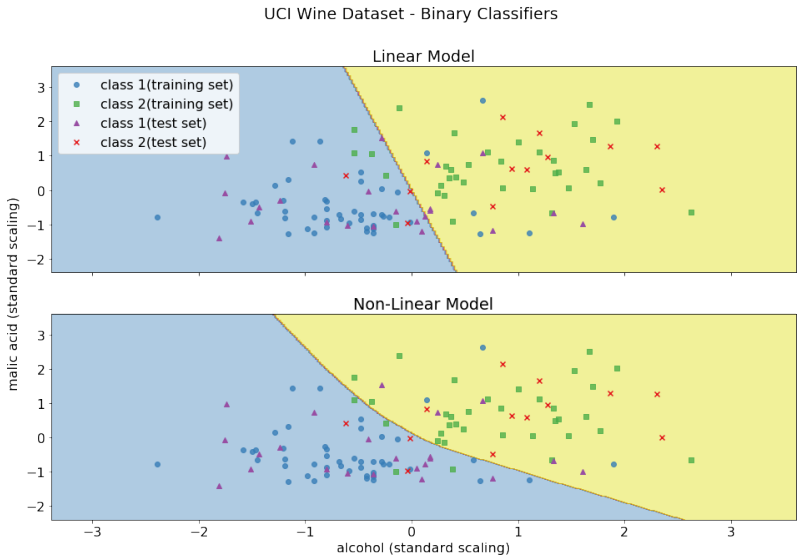



Figure 9: Decision boundary learned by a linear and non-linear binary classifier.

As it can be deduced from Figure 9, the decision boundary of the non-linear model is a better fit than the linear model. The CMs in Figure 10 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 10.

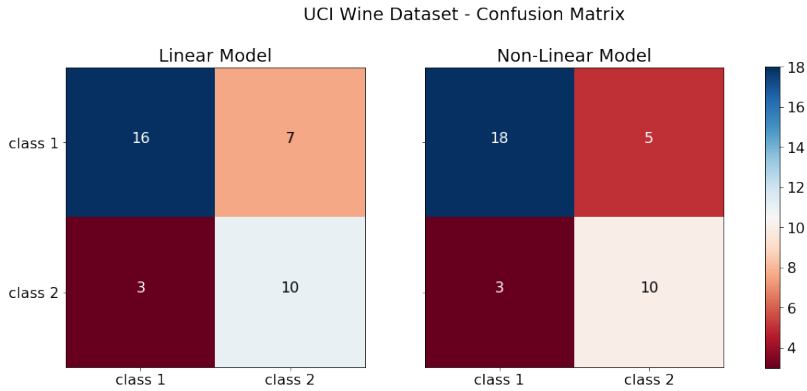



Figure 10: The linear classifier has 16 TP, 10 TN, 7 FN and 3 FP, whilst the non-linear classifier has 18 TP, 10 TN, 5 FN and 3 FP.

# Convolution

## Convolution in Mathematics

### Definition

Convolution is a mathematical operation on two functions to produce a result that reflects how one of the input functions is modified by the other input function.
The convolution of functions $f(x)$ and $g(x)$ (denoted $f * g(x)$) is defined (Bracewell, 2000), for continuous functions $f$ and $g$, as[2]

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{31}$$

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

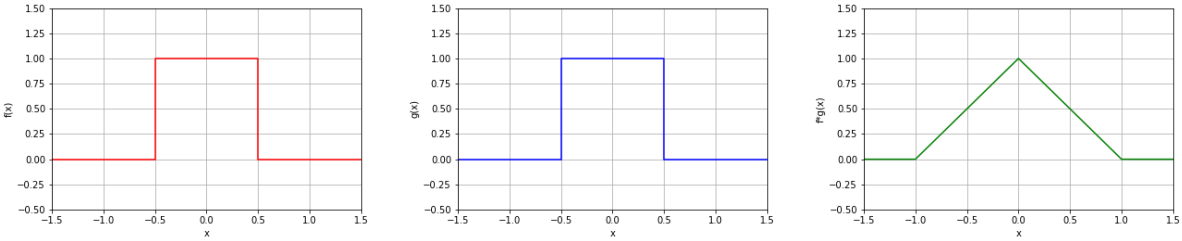



Figure 11: These plots show two continuous functions ($f(x)$ and $g(x)$) and their convolution $f * g(x)$

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

$$(f * g)(n) \equiv \sum_{m=-\infty}^{\infty} f(m)g(n-m) \tag{32}$$

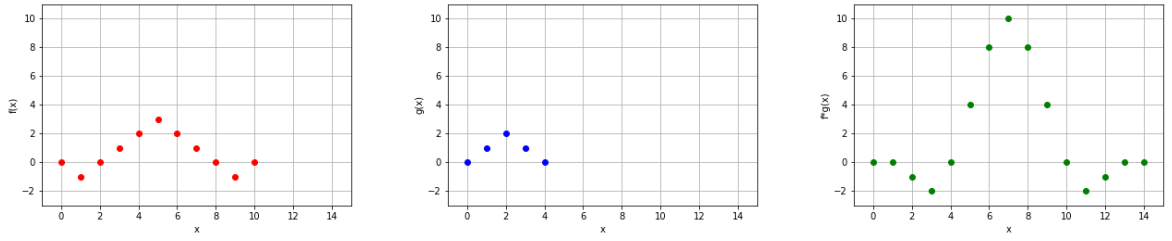



Figure 12: These plots show two discrete functions ($f(n)$ and $g(n)$) and their convolution $f * g(n)$

### Properties

Since convolution is defined as a product of integrable functions on the linear space, the following algebraic properties are satis-

fied (Bracewell, 2000):

**Commutativity**

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

**Associativity**

$$(f * (g * h)) = ((f * g) * h) \tag{34}$$

**Distributivity**

$$(f * (g + h)) = (f * g) + (f * h) \tag{35}$$

**Multiplication by a scalar value**

$$a(f * g) = (af) * g \tag{36}$$

**Multiplicative identity**, where $\delta$ denotes the delta distribution

$$f * \delta = f \tag{37}$$

**Differentiation**

$$\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx} \tag{38}$$

**Integration**

$$\int_{R^d}(f * g)(x)dx = \left(\int_{R^d} f(x)dx\right)\left(\int_{R^d} g(x)dx\right) \tag{39}$$

*Applications*

Listed below are some of the main applications of the convolution operator in various fields of knowledge (Srivastava and Buschman, 2013):

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

- **Probability Theory** - The Probability Density Function (PDF) of the sum of two independent random variables can be obtained by the convolution of the PDFs of the two variables

## *Convolution in Neural Networks*

### *Definition*

Convolutional Neural Networks (CNNs) are a subclass of multi-layer neural networks in which some of its hidden layers are convolutional layers.
Like most neural networks, CNNs can be trained using back propagation algorithms and are mostly used to recognize visual patters with minimal or no preprocessing (Lawrence et al., 1997).

### *Origins and Evolution*

CNNs were initially developed as an attempt to replicate the processing of sight in living organisms.
A seminal paper published in 1968 (Hubel and Wiesel, 1968) noted that two types of neurons took part in the process of identifying images: simple cells (responsible for the detection of straight edges and their orientation), and complex cells (with larger receptive fields, but not affected by the exact position of the edges in the input image).
During the 1980s, CNNs evolved, followed by the introduction of Time Delay Neural Networks (TDNNs) in 1987 (Waibel et al., 1989).
In the early 1990s, CNNs were modified and used for medical image processing and for automatic detection of breast cancer in mammograms (Zang et al., 1994).
In 1998 LeCun lead a team that created LeNet-5, a CNN used by banks to identify handwritten digits in checks.
With the introduction of Graphic Processing Units (GPUs), training of CNNs was greatly improved, allowing for the implementation of efficient Deep Learning Neural Networks with impressive results in image processing and other applications (Ciresan et al., 2013).

### *Layers*

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

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

as each point of the activation map being affect by only a small subset of input points.

- **Pooling layer** - In this layer, simple operations are applied in order to reduce the size of the input data, such as maximum pooling or average pooling.
  In 2x2 average pooling, for example, each result point the the average of 4 input points, reducing the size of the input field by a factor of 4.
- **Rectified Linear Unit (ReLU) layer** - This layer applies the $f(x) = max(0, x)$ activation function to its input, removing negative values from the input field.
- **Fully Connected layer** - In this layer, usually applied after several convolutional and pooling layers, neurons are connected to all activations in the previous layer, as in regular neural networks.
- **Loss layer** - This is the last layer in a typical CNN. In this layer, training is based on the divergence between the desired and predicted labels.

### *Training*

One important feature of CNNs is that, with the exception of the Loss layer, all other layers (Convolutional, Pooling, ReLU, and Fully Connected layers) can be trained using unsupervised training (Arel et al., 2010).
Also, usually each layer or set of layers is trained individually, which also improves the speed and the overall result of the training.
With the introduction of Graphic Processing Units (GPUs), speed of training and processing in the Convolutional layer has been greatly improved (Steinkrau et al., 2005)

### *Applications*

CNNs are currently applied to a wide range of areas, such as (Schmidhuber, 2015):

- **Image recognition and Video analysis** - Image Recognition Systems using CNNs are very efficient, specially if applied to facial recognition. Very good results have been obtained in challenges, and in the ImageNetNet tests performed almost as good as humans. If compared to image recognition, video analysis adds a level of complexity, since, in addition to the analysis of each frame, a time dimension needs to be added to the problem and convolution needs to be performed both on each image as well as on the time domain.
- **Game strategy** - A number of board games have been implements using CNNs, most notably Checkers, in Chess and in Go, the first time Artificial Intelligence beat an expert human player in this game.

- **Drug discovery** - In this area, CNNs are used to predict the behaviour of drug molecules and human proteins.
- **Natural language processing** - For NLP, CNNs are used extensively, including semantic parsing, search query retrieval, sentence modeling, classification, and prediction.

## *Convolutional Filters*

### *Definition*

Convolutional filters (also known as kernels), when applied to image and video processing, are small square matrices with an odd number of rows and columns and used in the Convolutional layer of CNNs in order to detect or enhance specific features in the input image.
The convolution operation performed by CNNs is a special two dimensional case of the discrete convolution operator.

$$R(m,n) = (K * I)(m,n) \equiv \sum_{s=-a}^{a} \sum_{t=-b}^{b} K(s,t) I(m-s, n-t) \qquad (40)$$

where $R(m,n)$ is the convoluted result at point $m,n$, $K$ is the kernel matrix being applied to the input matrix, $I$ is the input matrix, and $a$ and $b$ depend on the dimensions of the kernel used.

### *Practical example: Edge Detection kernel applied*

The figure below shows a practical example of the application of the Edge Detection kernel:

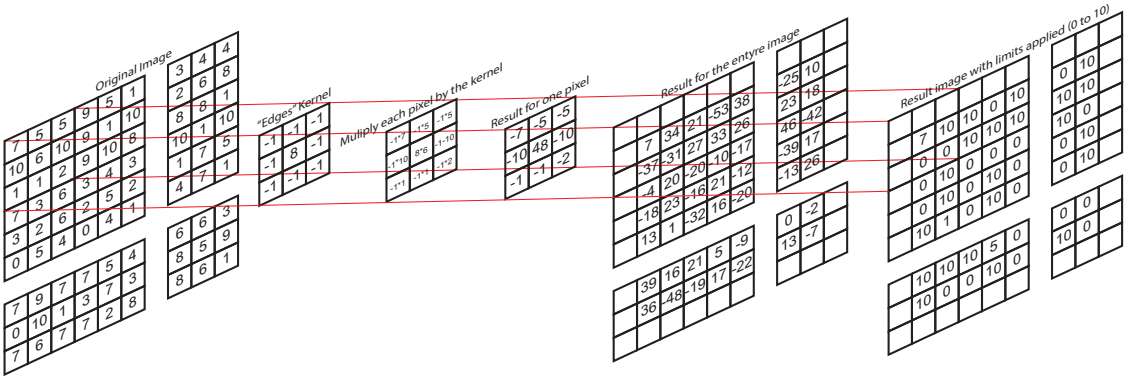



Figure 13: Edge Detection kernel applied

- The kernel is moved over the entire image, pixel by pixel
- At each position, the value for each pixel in the original image is multiplied by the corresponding value in the kernel ($7 * -1 = -7$; $5 * -1 = -5$; $5 * -1 = -5$; $10 * -1 = -10$; $6 * 8 = 48$; $10 * -1 = -10$; $1 * -1 = -1$; $1 * -1 = -1$; $2 * -1 = -2$)
- All results are added ($(-7) + (-5) + (-5) + (-10) + 48 + (-10) + (-1) + (-1) + (-2) = 7$)

- Optionally, the result of the summation may be limited to valid values (colors in images typically range from 0 to 255. In this example we have limited the values to $[0 - 10]$)

### *Common Convolutional Filters*

Convolutional filters can be defined to achieve a large range of effects in the input image.
A small sample of possible kernels is provided below:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Figure 14: **Identity** Kernel and the original image (unchanged, no convolution applied)

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 15: **Gaussian Blur** Kernel and the resulting image

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 16: **Box Blur** Kernel and the resulting image

$$\frac{-1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & -476 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$$

Figure 17: **Unsharp** Kernel and the resulting image

$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Figure 18: **Sharpen** Kernel and the resulting image

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

Figure 19: **Edge Detection** Kernel and the resulting image

# *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. 21a. This is repeated for every combination of test and training set formed from the original data set and the average error is obtained. Therefore, this method trains and tests the algorithm $\binom{n}{p}$ times when the number of samples in the original data set is $n$, becoming inapplicable when $p > 1$ (Arlot et al., 2010).

### *Leave-one-out (LOO)*

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

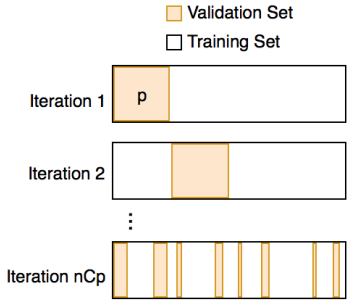



Figure 20: Leave-p-Out Exhaustive Cross Validation

## *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).

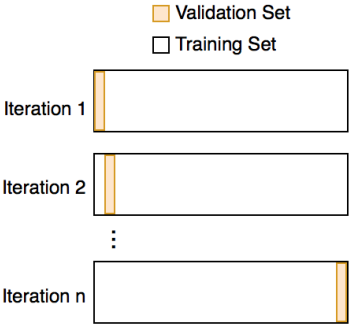



Figure 21: Leave-One-Out Exhaustive Cross Validation

### *k-fold*

This method randomly splits the original data set into $k$ equally sized subsets, as shown in Fig. 22. The function is then trained and validated $k$ times, each time taking a different subset as the test data and the remaining $(k-1)$ subsets as the training data, using each of the $k$ subsets as the test set once. The $k$ results are averaged to produce a single estimation. Stratified $k$-fold cross validation is a refinement of the $k$-fold method, which splits the original samples into equally sized and distributed subsets, having the same proportions of the different target labels (Kohavi, 1995).

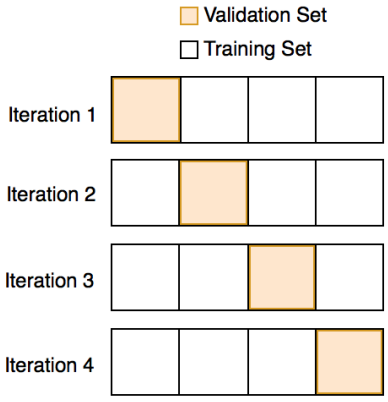



Figure 22: $k$-Fold Cross Validation where $k$=4

### *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. 41, 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.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{41}$$

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. 24a. 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. 24b, overfitting is when the model has a too high degree (e.g. $y = x^{30}$, where the degree is 30) causing it to exactly fit the data as well as the noise and so lacks the ability to generalize (Baumann, 2003), making the model have a high variance. Cross-validation helps reduce this bias and variance since it uses most of the data for both fitting and testing and so helps the model learn the actual relationship within the data. This makes cross-validation a good technique for models to acquire a good bias-variance tradeoff (Arlot et al., 2010).

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

Therefore, lets take an example using the stratified $k$-fold cross-validation method with $k = 10$. Let's say that we are trying to solve age group classification, using eight non-overlapping age groups being 0-5, 6-10, 11-20, 21-30, 31-40, 41-50, 51-60, and 61+. We are using the FG-NET labelled data set, which contains around 1000 images of individuals aged between 0 and 69. Before we can start training our model (e.g. CNN), we must divide our data set into training and test subsets and this is where cross validation comes in. Therefore, we start by taking the 1000 images of our data set and splitting them according to their target class. Let us assume we have an equal amount of 125 (1000/8) images per class[3]. As depicted in Fig. 25, we can now start forming our 10 folds by taking 10% of each age-group bucket, randomly without replacement. Hence, we will

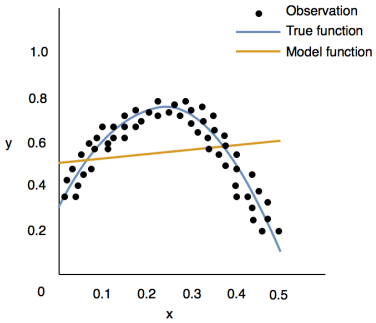



Figure 23: Model Underfitting

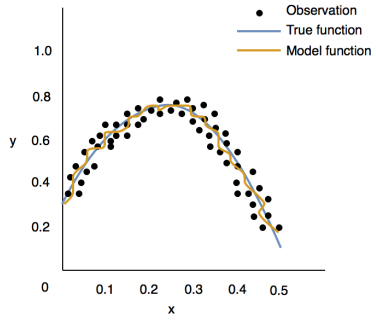



Figure 24: Model Overfitting

[3] 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. 22). For each iteration we use the misclassification error rate to obtain an accuracy value and we finally average the 10 accuracy rates to obtain the global accuracy of our model when solving age group classification, given the FG-NET data set. Hence, we have now estimated the prediction error of the model and have an idea of how well our model performs in solving such a problem. It is important to note that cross-validation is *just* an estimation method and when using our model in real-life applications we do not apply CV but rather train our model with all the data we have.

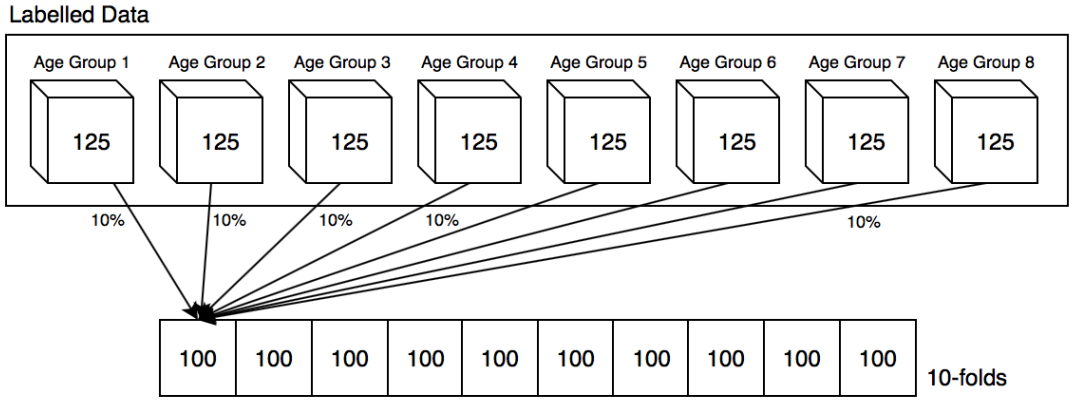



Figure 25: Stratified 10-fold cross-validation on 1000 labelled images of 8 different classes

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

# *Dimensionality Reduction*

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

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

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

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

## *Random Forests*

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

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

and Johnson, 2013).

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

Strobl et al. (2007) discuss that RFs are not ideal for feature selection when the features to be used for prediction constitute of a mixture of categorical and continuous values. Moreover, the variable importances measures are unreliable in cases when the number of categories for categorical predictors vary in the sample set.

### *Genetic Algorithms*

Genetic algorithms (GA) are a commonly used technique in feature selection, which aims at minimizing the feature space by eliminating unimportant variables and keeping those which maximize accuracy in predictions. Introduced by Holland (1962), GA works using a procedure which mimics the theory of evolution, in which a population is manipulated to generate offsprings which form the new generation.

In GA's, a population is made up of a set of individuals (samples from a sample space), which are represented using binary strings, and referred to as *chromosomes*. Each chromosome is composed of a set of features, which are called *genes* (Siedlecki and Sklansky, 1989). A *fitness function* is applied to each individual to determine the fittest individuals within the population. GA then applies selection to obtain the fittest individuals. Pairs of individuals, referred to as parents, are then constructed from the selected individuals, and crossover is performed. Crossover involves generating offsprings from the identified parents by exchanging information amongst the parent pairs. Mutation, which is applied to increase the diversity within the population, entails that one or more bits are randomly flipped to generate a different offspring. GA repeatedly perform this process, also referred to as a generation, until a stopping condition is met (Chaikla and Yulu Qi, 1999).

In feature selection, the individual considered as the fittest from the new population obtained is used to determine the features with the highest importance. Genes with value '1' in the obtained chromosome are considered as the features to be used for predictions (Sivanandam and Deepa, 2007).

### *Principle Component Analysis*

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

Prior to applying PCA, the dataset is adjusted such that the means of the original variables are subtracted from the observations, to produce a dataset with mean zero (Jackson, 2005). PCA works by initially constructing the *covariance matrix* of the data (Guan and Dy, 2009). A high covariance value indicates that the variables are correlated, whereas a zero value indicates that no correlation exists between the variables. Moreover, the covariance between $x$ and $y$ is equal to the variance of $x$, when $x = y$.

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

Figure 26: Covariance Matrix.

The variance between features is indicated by the diagonal values of a covariance matrix (Figure 26), whilst the off-diagonal values refer to the covariance between the features. High values in the off-diagonal terms indicate that high redundancy exists, and thus the covariance matrix must be diagonalized to minimize the correlations between the features.

The resultant diagonalized covariance matrix is used to calculate the *eigenvalues* and *eigenvectors*. The eigenvectors refer to the direction in which the data varies, and these are ordered such that the ones with the highest eigenvalues are chosen as the principal components of the data. Eventually, the selected vectors are kept to form a *feature vector*, which will be used to construct the new dataset with lower dimensionality, as in Equation 42

$$NewDataset = FeatureVector^T x AdjustedData^T \tag{42}$$

Equation 42 multiplies the transpose of the feature vector by the transpose of the mean adjusted data, to obtain the original data with smaller dimensionality (Jackson, 2005).

### *Multi-dimensional scaling*

Multi-dimensional scaling (MDS) is another dimensionality reduction technique which similar to PCA seeks to obtain a data set with lower dimensions. MDS aims at finding a configuration in which the distances between points on a dimensional space are as close as possible to the dissimilarities between the points. Different techniques of MDS exist which differ in the way that points are matched to dissimilarities. In *Metric MDS*, "dissimilarities between objects will be in numerical and distance". Classical scaling is one such technique which aims at finding a configuration in which the dissimilarities are equal to Euclidean distances. Another technique is least squares scaling in which the dissimilarities are transformed into distances using a function,

and the configuration space is obtained by minimising this function. On the other hand, *Non-Metric MDS* seeks a transformation which preserves the rank order of the dissimilarities (Cox and Cox, 2000).

### *Linear Discriminant Analysis*

LDA is a supervised dimensionality reduction technique which produces a lower dimensionality feature space in which a feature vector belonging to a class is distinguishable from that of the other classes (Sharma and Paliwal, 2015). Figure 27 depicts the projection of a two-dimensional feature vector to a one-dimensional feature space. $C_1$, $C_2$ and $C_3$ denote the three different classes being used within the feature vectors. By projecting the feature vectors onto line $\hat{W}$, which represents one dimension, it is noted that a strong relationship exists between the feature vectors of the three classes. LDA seeks an orientation which strongly separates the feature vectors of one class from those belonging to other classes. This orientation can be obtained by rotating $\hat{W}$, resulting in projection $W$, where the feature vectors of the classes are highly separated. Algorithmically, Fisher's criterion is maximized to obtain the required one-dimensional feature vector $W$.

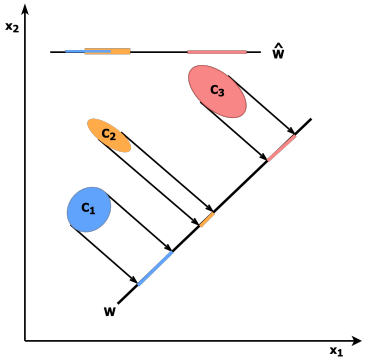



Figure 27: Projection of a two-dimensional feature vector to a one-dimensional feature space. Adapted from Sharma and Paliwal (2015).

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

$$S_B = \sum_{i=1}^{c} n_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{43}$$

This is referred to as the *between-class matrix* and is represented by Equation 43, where $c$ is the number of classes, $n_i$ is the number of samples for the $i^{th}$ class, $\mu_i$ is the mean for the $i^{th}$ class and $\mu$ is the total mean of all samples.

$$S_W = \sum_{j=1}^{c} \sum_{i=1}^{n_j} (x_{ij} - \mu_j)(x_{ij} - \mu_j)^T \tag{44}$$

Following this, the *within-class matrix* is calculated using Equation 44, where $c$ is the number of classes, $n_j$ is the number of samples for the $j^{th}$ class, $x_{ij}$ is the $i^{th}$ sample in the $j^{th}$ class, and $\mu_j$ is the mean for the $j^{th}$ class. This matrix indicates the distance between the mean and sample values for each class. Finally, a *transformation matrix* $W$ which maximizes the between-class variance and minimizes the within-class variance should be constructed.

Fisher's criterion, shown in Equation 45 is used to obtain the maximum ratio of the between-class matrix (denoted by $S_B$) and within class matrix (represented by $S_W$). By transforming the latter into Equation 46, where $\lambda$ denotes the eigenvalues, eigenvectors and the corresponding eigenvalues of $W$ may be calculated. A lower dimensional space is constructed by selecting the eigenvectors having the highest eigenvalues.

$$\arg\max_{w} \frac{W^T S_B W}{W^T S_W W} \tag{45}$$

$$S_W W = \lambda S_B W \tag{46}$$

# *Empirical Risk Minimisation*

## *Introduction*

Empirical Risk Minimisation (ERM) is a principle of statistical theory that is used in supervised learning. Simply put, it states that for a function to make the most accurate predictions possible on an unknown true distribution, prediction errors made on the data belonging to the distribution must be minimised. To give a proper explanation of the principle some prior concepts need to be explained. The next section will discuss loss functions.

## *Loss Functions*

A simple supervised learning scenario comprises of the following. Typically there would be one or more inputs or features denoted as $x$ and the target output or labels denoted as $y$. Using the combination of features and output $(x, y)$, a prediction function $f(x)$ is built where, given a new input $x'$ gives the best estimate for the value of $y'$, assuming that the data follows an underlying distribution that is not known. Once a set of predictions is generated, a way of measuring how good these predictions are is required. For a single data point this is simply the distance between $y$ and $y'$. Repeating this process for multiple points will give us a loss function (Wehenkel, 2018):

$$L(f(x), y) \tag{47}$$

The loss function gives us the cost our prediction function (f(x)) which gives a clear indication of how efficient the function is at predicting the expected output. Common loss functions are squared-error loss and mean square error. This calculation of "cost" of the function can give us an indication of the risk of the prediction function. Therefore, the goal of finding the most optimal prediction function turns out to be the goal of finding the function which minimises the true risk.

## *True Risk*

True Risk is the prediction error on the entire population of data. Unfortunately in most cases, it is impossible to collect the entire population dataset. For example, in the case of predicting the gender of a person based on their height and weight, every single person

across the globe would need to have their height and weight collected into one set of data. Since the entire population of data is unavailable, there needs to be a way of minimising the true risk, without the true distribution being known (Vapnik, 2000)

### *Empirical Risk Minimisation*

This is where Empirical Risk Minimisation comes in. When constructing the supervised learning model, the predictive function chosen is the one with the least discrepancy between predicted values and actual values. In other words, the one with the least risk. This risk is Empirical Risk and the process of finding the function that minimises this risk is called Empirical Risk Minimisation. Note the risk is empirical risk and not the true risk as the data is only subset of the entire population. In an ideal scenario the end goal would be to minimise the true risk as much as possible. Due to the fact that the data for calculating the true risk is unavailable this cannot be achieved, however it is said that the empirical risk is almost identical to true risk. Therefore by minimising the empirical risk the true risk will be minimised also.

### *Equation of Empirical Risk Minimisation*

The following is the equation for the Empirical Risk Minimisation as proposed by Vapnik (1991):

$$R_{emp}(f) = \frac{1}{m}\sum_{i=1}^{m} L(f(x_i), y_i). \tag{48}$$

Here $R_{emp}$ denotes the empirical risk or empirical error for a particular prediction function $f(x)$. The number of input and output pairs is denoted as m. As can be seen in the equation above, empirical risk equates to the sum average of the of the loss function on each pair of $x$ and $y$ points generated. The induction principle behind empirical risk minimisation makes the assumption that the prediction function $f(x)$, which minimises the risk $R_{emp}$, will result in a risk $R_{emp}*$ which is close to its minimum.

### *Uniform Convergence*

The convergence of the empirical risk to the true risk can only be achieved under specific conditions and bounds (Clémençon et al., 2017). These bounds are independent of the prediction function and are based on the VC-dimension of the dataset. This dimension basically measures the complexity of the prediction function. The VC-Dimension set also measures ability of the function to perform correct predictions on unseen data, using the least training data possible. A low VC Dimension indicates a high bias in the function, which will lead the function to perform poorly (especially on training data) as it will cause underfitting. On the other hand, a high VC-Dimension

which indicates that the function has a high complexity. The complexity could be high to the point that the function would memorise the data rather than idenitfy the underlying pattern which causes overfitting. Based on this, prediction function with high complexity should be avoided. When comparing two fcuntions which have roughly the same empirical error, the least complex one should be chosen in order to avoid the scenario of overfitting.

### *Problem of Empirical Risk Minimisation*

The most common problem of ERM as mentioned above is overfitting. As the goal is to reduce the empirical error as much as possible, the result could be a prediction function that fits the training data perfectly. However when given unseen data this will perform poorly causing overfitting (Shalev-Shwatz, 2014).

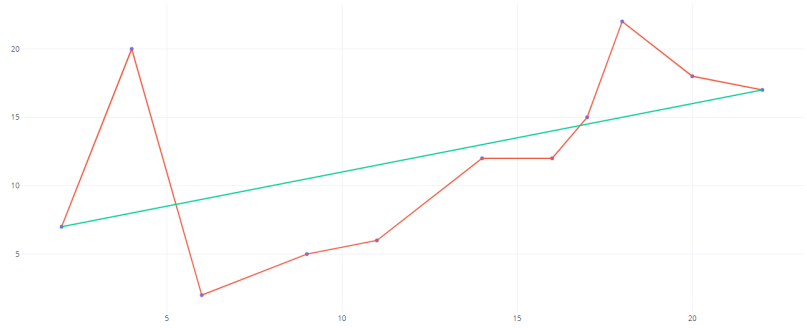



Figure 28: Example of Overfitting

Figure 28 above is a perfect example of overfitting as a result of Empirical Risk Minimisation. The green line shows the line of best fit of the predictive function which indicates how the function would best represent the data. The orange line is the same predictive function after attempting to minimise the empirical risk to it's lowest point. For the training data, the function will perform extremely well with almost no error however this is due to overfitting, resulting in the function to perform extremely poorly on the testing or real world data. For this particular case the Empirical Risk Minimisation would be giving a false representation of the true risk as it would appear to be a low empirical error when in reality it would be extremely high on unseen data.

### *Structural Risk Minimisation*

The principle of Structural Risk Minimisation (SRM) was introduced as an effective method of tackling the problem of overfitting in Empirical Risk Minimisation. This does so by making use of the VC-Dimension mentioned previously to analyse a prediction function's complexity and by using Empirical Risk Minimisation. The aim of this minimisation principle is to find the most optimal predictive function, which can be defined as the function with a low value of empirical

risk and a low VC-Dimension (low complexity). This balance between these two bounds is the essence of Structural Risk Minimisation.

### *Factors affecting Empirical Risk Minimisation*

From what can be seen there are several factors that indicate how good approximation of empirical risk will be. Firstly, the size of the dataset. A small dataset will most likely lead to high bias and overfitting meaning a large dataset is preferred as it will provide a better approximation of the true distribution. The underlying distribution of the data will also affect the approximation as complex and irregular distributions will cause the function to overfit as previously mentioned. Another factor is the loss function used. The loss function should be consistent with the data and not show irregular spikes in loss values at certain data points. Lastly, another factor that affects empirical risk minimisation is the prediction function. If the function considered is large and complex, it will be a much more difficult task in finding the empirical risk and the true risk.

### *Conclusion*

In conclusion, the principle of Empirical Risk Minimisation is an effective way of minimising the empirical error of a prediction function. Under the appropriate conditions, by reducing this risk on a subset of the population data then the true risk will be reduced to the same extent. However, caution needs to be taken as to prevent the case of overfitting.

# *Ensemble Methods*

An ensemble is a method used in machine learning to combine predictions made by multiple learning algorithms to achieve better predictive performance (Clemen, 1989; Perrone, 1993). This technique works with classification problems, for example predicting whether a company will go bankrupt (Zięba et al., 2016), and it also works with regression problems, for example predicting crude oil prices (Yu et al., 2008). The simplest ensemble technique used in classification problems is to take the maximum vote of all outputs made by each classifier. On the other hand, averaging the predictions made by each predictor is the most basic technique used for regression problems.

The general idea behind ensembles is to merge each learner's hypothesis into one with the intention of obtaining better predictions. There are various ensemble methods, some of which use a single base learner to produce homogeneous learners while others use individual learners to produce heterogeneous learners. Generally, ensemble methods are applied to supervised learning algorithms. The methods described below are the most common used ensemble techniques in machine learning.

## *Bagging*

*Bagging*, also referred to as *Bootstrap Aggregating*, is a technique introduced by Breiman (1996a). In his work it was shown that classification and regression trees (Breiman et al., 1993) can have a significant increase in accuracy when combined with *Bagging*.

This method works by sampling the training dataset into multiple subsets or bags of equal parts. These subsets are sampled using replacement, meaning that an instance can be found in multiple subsets. The size of each subset is defined as a percentage of the total size of the training dataset. Once the training set is sampled, each subset is utilized to fit a different learner. A prediction $\hat{y}$ on an unforeseen instance $x$ is made by taking a maximum vote made by each learner as the predicted classification as shown in Equation 49. If it is a regression problem an average of the predicted values is taken as the final output as shown in Equation 50. This procedure can be better visualized in Figure 29. The idea behind this technique is to

avoid overfitting by reducing variance.

$$f(x) = argmax \sum_{b=1}^{B} f_b(y|x) \tag{49}$$

$$f(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x) \tag{50}$$

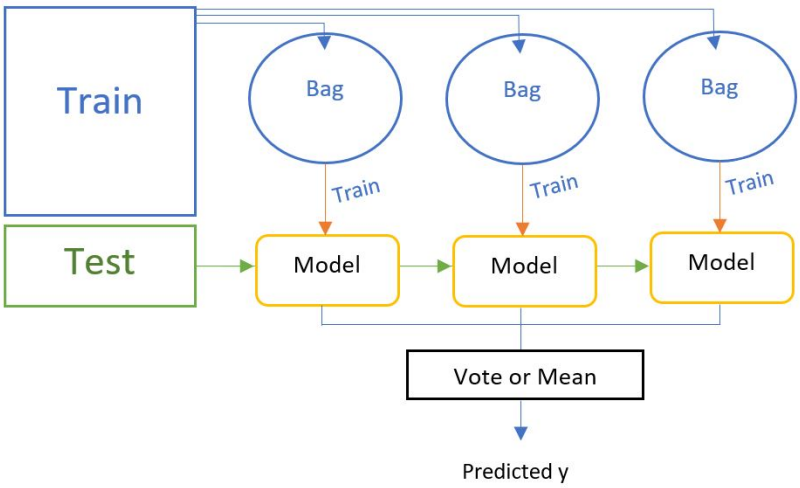



Figure 29: Dataset Bagging using 3 base learners. Each bag is a subset of the training set. Instances are selected randomly and with replacement. Each base learner is fitted with the respective bag. Predictions are then made by each base learner and the final output $\hat{y}$ is the maximum vote or an averaged value of all predictions.

*Bagging* can also be applied to the feature set where subsets of the features is selected at random and used to fit the base learners. This technique is referred to as *Feature Bagging*. One popular model that utilizes *Feature Bagging* is *Random Forests* introduced by Ho (1995). It uses multiple *Decision Trees* which are trained using subsets of the total feature set selected at random. It was shown that when applying this method with multiple *Decision Trees*, the model was able to generalize more when classifying handwritten digits. An extension to *Random Forests* was later developed which uses both *Bagging* and *Feature Bagging* to further control variance (Breiman, 2001b).

## *Boosting*

This technique is a variation on *Bagging* which aims to improve the learner's predictive performance by correcting the errors made by the previous learners in the ensemble, hence the name *Boosting*. The idea of having multiple *weak learners* that adapt by correcting errors was introduced by Freund and Schapire (1997) and this technique is called *Adaptive Boosting* or *AdaBoost*, which is the most common *Boosting* method. It was shown that this technique can reduce both bias and variance (Breiman, 1996b). *AdaBoost* does not work well when having insufficient data or very weak hypotheses (Freund et al., 1999). It also suffers when having a large number of outliers in the dataset and can significantly reduce the predictive performance (Dietterich, 1998). Variants of *AdaBoost* were developed to combat large amount of outliers in the dataset such as *'Gentle AdaBoost'* which gives less

importance to outliers (Friedman et al., 2000) and *'BrownBoost'* which uses a combination of Brownain motion, *Boosting* and repeated games to handle such cases (Freund, 2001).

*Boosting* works by taking a subset of the training set to train the first learner. Once the learner is trained, it is validated using the whole training set. When validation is done, each instance in the original dataset is given a weight based on the error made by this learner. A new learner is now trained on a new subset, but this time instances with higher weights are more likely to be selected. Again, this learner is validated using the whole training set but the weights for each instance is now recalibrated using a combined prediction of the two learners as shown in Figure 30. This process goes on for the number of learners specified to be used in the ensemble and the idea behind it is that each learner corrects the previous learner's errors. Once this procedure is finished, a prediction is made on the weighted mean of each learner if it is a regression problem as shown in Equation 51. If it is a classification problem, it takes a weighted vote to come up with the predicted classification as shown in Equation 52.

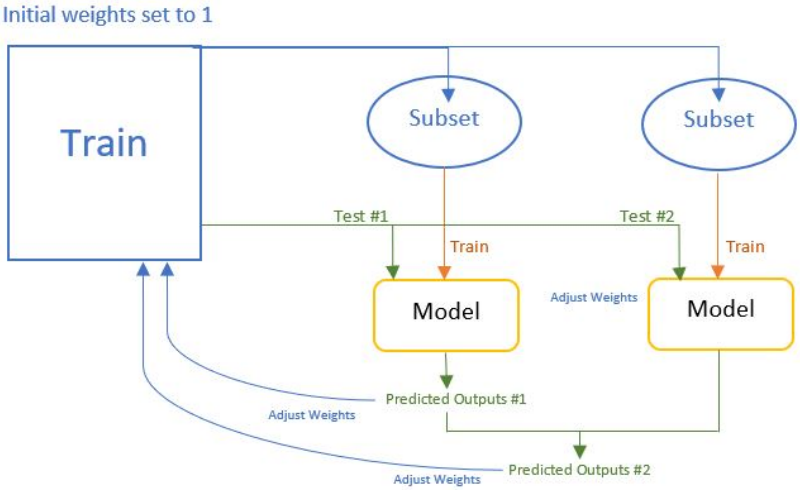



Figure 30: Boosting using 2 base learners. Initially weights are set to 1, so all instances have the same probability to be selected in the subset. The first learner is trained using the first subset and then validated using the whole training set. Weights are recalibrated depending on the errors of the predictions. On the second run, instances which had significant errors are more likely to be selected. The second learner is now trained and tested using the whole training set. This time weight are adjusted using the combined predictions of the two learners.

$$f(x) = \frac{1}{B}\sum_{b=1}^{B} \alpha_b f_b(x) \tag{51}$$

$$f(x) = sign(\sum_{b=1}^{B} \alpha_b f_b(y|x)) \tag{52}$$

This technique assumes that the learners are 'weak' and homogeneous. *Boosting* aims to build a stronger learner by combining a collection of *weak learners*. *Boosting* works in sequence and cannot be parallelized since the learners are dependent on each other. There are various other variations of boosting but one other popular implementation worth mentioning is *Gradient Boosting* 'GBM' which focuses on optimizing the loss function using boosting (Breiman, 1997).

## *Stacking*

*Stacking* or *Stacked Generalization* is an ensemble method introduced by Wolpert (1992), it aims to reduce the generalization error of the combined model. Unlike the previous techniques it can be used to combine heterogeneous learners. In fact, this technique works better when the learners are different from each other and are known to be strong learners.

In *Stacking*, the dataset is first split into two; the training set and the test set. The training set is further split into K-folds like the K-fold cross validation technique. Each learner is then trained on $K-1$ folds and validation is done on the holdout fold. For prediction made by each learner at the holdout fold is recorded and stacked together to form a new dataset. The actual value/label is also stacked at the end. This procedure continues for $K$ times. Once all the learners have been trained using this method, a new dataset would be created with the predictions made by each learner which will be used as features. This dataset is then used to train the *combiner learner*, which is also referred to as the *meta-model* or the *aggregator model* as shown in Figure 31. After this model is trained using this newly generated dataset it will be validated using the original test set.

When using *Stacking* for classification it is advised to use class probabilities instead of the predicted outcomes as it gives a better measure of confidence for each model, thus enhancing the meta-dataset (Ting and Witten, 1999).

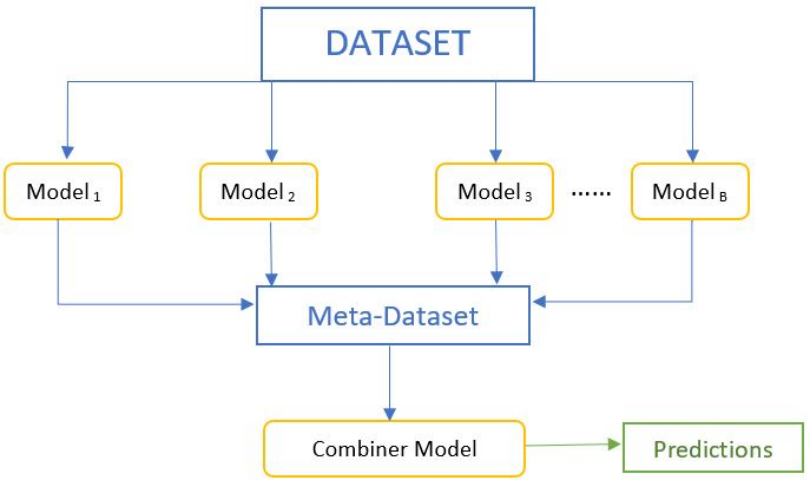



Figure 31: A visual representation of the steps required when Stacking multiple machine learning models.

The general idea behind this ensemble technique is to combine multiple different learners together, but unlike the previous methods, *Stacking* combines the models using another intermediate model which is trained on the predictions made by individual learners. *Stacking* technique was also studied to identify Higgs bosons at the Large Hadron Collider where *Stacking* outperformed *Boosted Decision Trees* (Alves, 2017).

# *Evaluation Methods*

Although modern machine learning is exceptionally diverse and multidisciplinary, what unifies the field is standard experimental evaluation methodologies that are adopted based on the task at hand (Kibler and Langley, 1988). Evaluation is involved both in model selection, i.e. during hyperparameter tuning, and in comparing multiple models against each other or a random baseline. Ideally, evaluation is aimed at interpreting a model's behaviour and analysing its errors (Doshi-Velez and Kim, 2017). Japkowicz and Shah (2011) outline four essentials of evaluation methodology: a test set, performance measures, error estimation, and statistical significance tests.

When evaluating machine learning models, it is common to rely on some annotated *gold standard* dataset that serves as a proxy of real-world phenomena (Manning et al., 2008). Examples of such datasets can be found in centralised collections such as the UCI Machine Learning repository (Dheeru and Karra Taniskidou, 2017) or online communities like Kaggle[4]. However, one needs to be aware that a dataset may not represent the whole domain (Japkowicz and Shah, 2011), contain annotation artefacts, or might have been constructed with vague class definitions (Hand, 2006).

[4] https://www.kaggle.com

While there is an assumption that training and testing sets come from the same distribution in order for a model to work as expected (Japkowicz and Shah, 2011), it is essential that the model is tested on previously unseen data. Otherwise, it yields overoptimistic results (Kibler and Langley, 1988). Error estimation techniques such as a held-out set or cross-validation are detailed on page 37, so the following is focussed on evaluation metrics and statistical tests.

## *Performance Measures*

Stapor (2017) notes that according to the *no free lunch theorem* (Wolpert, 1996), it is unreasonable to expect that one classifier is generally better than another since there will always be problems where one outperforms the other and vice versa. Evaluation is also highly task-specific and selecting a performance measure should be done with the end goal in mind (Goodfellow et al., 2016). For example, natural language generation models used in image captioning are penalised for "hallucinating" extra information, i.e. such models are considered worse, but this aspect can be encouraged when applied to poetry generation.

### *Automatic Evaluation Metrics*

For supervised classification, the simplest measure is *accuracy* which is the ratio of correctly predicted examples, and its reverse metric *error rate*, i.e. the fraction of errors. More fine-grained classification metrics are described on page 25.

For machine-learned ranking algorithms that infer an ordered list of multiple outputs, standard metrics can be reported *at N* (Herlocker et al., 2004), e.g. accuracy at 5 (Acc@5) illustrating a relaxed accuracy that looks for a match among the top five outputs for each example.

For cluster analysis, despite it being unsupervised, typical evaluation metrics still require some labelled gold standard dataset. The simplest measure is *purity* that indicates the degree a model is able to cluster together instances of the same ground truth class (Equation 53):

$$p(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|, \tag{53}$$

where $\Omega = \{\omega_1, \ldots \omega_k\}$ is the set of predicted clusters, $C$ is the set of reference classes, $N$ is the size of the test set (Manning et al., 2008). For other metrics used in clustering see, e.g. Friedman et al. (2001).

For regression methods, a common metric to report is the *root mean squared error* (RMSE) that characterises the discrepancy between predicted and true values (Equation 54):

$$RMSE(f) = \sqrt{\frac{1}{N} \sum_i^N (f(x_i) - y_i)^2}, \tag{54}$$

where $f(x_i)$ is a prediction for input $x_i$, $y_i$ is the target value, and $N$ is the test set size. Harrell (2015) overviews other regression metrics.

### *Manual Evaluation Metrics*

It is not always possible to use the aforementioned automatic metrics to assess a machine learning model in a task-specific context, e.g. when there are no ground truth data available or there are yet no reliable metrics correlating with human judgement or other real-world phenomena. Therefore, *manual* evaluation is sometimes required and is typically done either by experts or via crowdsourcing when raters are performing small paid evaluation tasks (Alonso et al., 2008) using specialised platforms such as Amazon Mechanical Turk[5]. For example, assessors can be provided with Likert scales (Likert, 1932) to rate various facets of a model's output, e.g. fluency of machine translation on a scale from 0 to 5; or they can be asked to rank outputs according to some aspect, allowing comparing multiple models. There have been attempts to replicate such judgements automatically, e.g. using *adversarial* evaluation (Jurafsky and Martin, 2018) that involves training Generative Adversarial Networks (Goodfellow et al., 2014a) to distinguish between human and machine-generated outputs.

[5] https://www.mturk.com

Another way to overcome the lack of metrics and data is to shift the task on which a model is evaluated, i.e. conduct *extrinsic* evaluation (Belz and Gatt, 2008) that commonly involves assessment of a

model's performance in a "downstream" application rather than the training task, as opposed to *intrinsic* evaluation. For instance, if the goal is to evaluate a feature reduction mechanism, one can measure its effect on the accuracy of sentiment classification (Schnabel et al., 2015). An example of manual extrinsic evaluation is A/B studies (Hofmann et al., 2016) aimed to quantitatively measure a change in user experience before and after deploying a new machine learning model. A/B testing consists of dividing users into two groups A and B and comparing them against each other according to some metrics. A/B tests can be employed to investigate the average time it takes users to find the desired webpage depending on the search algorithm (Manning et al., 2008) or the change in time users spend on a website after introducing a new recommendation system (Davidson et al., 2010).

### *Statistical Significance Testing*

Whenever a new algorithm is proposed, it needs to be compared against either a random baseline or previously published results for the task and dataset at hand. However, often improvement is rather incremental (Hand, 2006), so it might not be evident if it is of any importance. To decide whether a change is significant, *statistical significance tests* are encouraged, at least in case of scarce testing data (Drummond and Japkowicz, 2010).

Details about general statistical tests can be found in (Urdan, 2011), whereas Japkowicz and Shah (2011) provide recommendations on the correct use of such tests for classifier comparison. In evaluating machine learning techniques, one needs to be cautious when using standard parametric tests such as the paired $t$-test (Urdan, 2011) because of the assumptions expected to be met, e.g. equal variance of obtained metrics (Japkowicz and Shah, 2011). In contrast, *non-parametric* tests do not impose such restrictions on the population distribution which makes them more applicable but less powerful (Colquhoun, 1971). What follows is an illustration of applying a non-parametric test.

#### *An Illustration of Statistical Significance Testing*

Given two trained classifiers $A$ and $B$ and a set of ground truth labels of size $N$ against which predictions can be compared, it is possible to construct a $2 \times 2$ contingency table such as Table 7. Assuming that accuracy is selected as a performance metric, it can be seen for model $A$ it is $(CC + CI)/N = 1210/1243 = 0.973$, while for model $B$ it is 0.986. Based on the accuracy alone, the improvement of classifier $B$ appears rather small, so one would want to report statistical significance. A *null hypothesis* $H_0$ is formulated that there is no difference in accuracy between classifiers, whereas the alternative hypothesis $H_1$ states that these are significantly different. A *significance level* $\alpha$ representing the probability of wrongly rejecting $H_0$ is specified, e.g. 0.05. Supposing that it has been found that assumptions for parametric tests do not hold, a non-parametric alternative is used, e.g. *McNemar's test* (McNe-

| | | **B** | |
|---|---|---|---|
| | | Correct | Incorrect |
| **A** | Correct | $CC = 1200$ | $CI = 10$ |
| | Incorrect | $IC = 25$ | $II = 8$ |

Table 7: Contingency table breaking down predictions of two classifiers $A$ and $B$ into different cases based on comparison against the ground truth labels (correct or incorrect).

mar, 1947) with the following statistic:

$$\chi^2_{MC} = \frac{(|IC - CI| - 1)^2}{IC + CI}, \tag{55}$$

where $IC$ and $CI$ correspond to the cases described in Table 7. The $\chi^2_{MC}$ statistic from Equation 55 is compared against the $\chi^2$ distribution table value (or a binomial distribution in case of $CI + IC < 20$) (Japkowicz and Shah, 2011) that allows to find out the probability of obtaining such $\chi^2_{MC}$ under the assumption that $H_0$ is true. If this probability, referred to as *p-value*, is less than $\alpha$, $H_0$ is rejected.

For the example in Table 7, $\chi^2_{MC}$ is 5.6 and the $p$-value is 0.017, which is less the selected $\alpha = 0.05$. Therefore, it is said that the improvement in accuracy is significant.

## *Conclusions*

In conclusion, each component of a machine learning evaluation framework, i.e. selecting a testing dataset and performance measures, conducting error estimation and statistical significance tests, requires careful consideration with respect to the end goal; however, rigorous evaluation can pave the way to interpretable and more reliable machine learning systems (Doshi-Velez and Kim, 2017).

# *Feature Selection*

*Feature Selection* (FS) is a process that chooses an optimal subset of features according to certain criterion (Liu and Motoda, 2012). According to Sammut and Webb (2017), in many real-world applications such as: data mining, machine learning, computer vision, and bioinformatics we need to deal with high-dimensional data. Lately, the dimensionality of the data involved in these areas has increased exponentially. The trend of this growth is also reflected in the *UCI machine learning repository*[6] as shown in Figure 32.

[6] https://archive.ics.uci.edu/ml/index.php

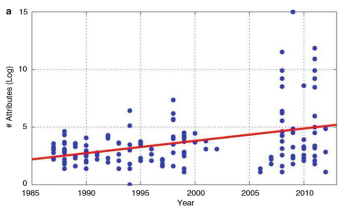



Figure 32: Growth of the number of features in the UCI ML repository. Reproduced from Sammut and Webb (2017).

Guyon and Elisseeff (2003) state that the objective of FS is threefold: improving the prediction performance of the predictors, providing faster and more cost-effective predictors, and providing a better understanding of the underlying process that generated the data. Furthermore, the aim of the FS is to select features that are *independent* of each other but *highly dependent* on the value to be predicted.

FS keeps the original feature, that is, it maintains the physical meanings of the original feature without any transformation. Furthermore, Masaeli et al. (2010) found that this property has its significance in many practical applications such as finding relevant genes to a specific disease and building a sentiment lexicon for sentiment analysis.

According to Zhao et al. (2010), FS is an optimization problem that can be split into two perspectives:

1. searching for the best subset of features,

2. principle model for: selecting, adding, removing or changing new features during the search using a certain evaluation measure.

Mainly there are three models used for FS: filter, wrapper and embedded.

## *Filter models*

*Filter models* operate independently of the learning algorithm. A filter algorithm usually consists of two steps. First, each feature is given a weight then, the features are ranked and the top-ranking features are chosen whilst the others are discarded. Selection criteria section describes some popular measurements used to calculate feature weight.

*Selection criteria*

- *Correlation*: this measures the ability to predict the value of one variable from the value of another. In FS for classification, we look for how strongly a feature is associated with the class. A feature $x_1$ is preferred to another feature $x_2$ if the association between feature $x_1$ and class $Y$ is higher than the association between $x_2$ and $Y$. In FS for clustering, the association between two random features measures the similarity between the two (Liu and Yu, 2005). According to Guyon and Elisseeff (2003), one of the simplest dependency measure function is *Pearson's correlation coefficient* defined in Eq. 56, where $x_i$ is the $i_{th}$ feature, $Y$ is the output (class labels), *cov* is the covariance and *var* is the variance.

$$\rho(i) = \frac{cov(x_i, Y)}{\sqrt{var(x_i) * var(Y)}} \tag{56}$$

- *Information gain*: this is one of the most popular measures in FS due to its computational efficiency and simple interpretation (Tang et al., 2014). It is used to measure the dependence between features and labels. The information contained in label $Y$ can be defined by *Shannon's entropy* $H(Y)$. This is defined in Eq. 57 where $y_j$ are the labels and $P(y_j)$ is their probability. If we observe feature $X$ then the conditional entropy $H(Y, X)$, can be defined by Eq. 58 where $P(y_j, x_i)$ is the joint probability. Finally, *mutual information* $MI(Y, X)$ between label $Y$ and feature $X$ can be defined by Eq. 59.

$$H(Y) = -\sum_j P(y_j) \log_2 P(y) \tag{57}$$

$$H(Y, X) = -\sum_i \sum_j P(y_j, x_i) \log_2 P(y_j, x_i) \tag{58}$$

$$MI(Y, X) = H(Y) - H(Y|X) \tag{59}$$

Filter models normally are very fast because usually the evaluation measurement complexity is simple and cheap, both in terms of performance and memory. Having said this, García et al. (2015) found that due to it's simplicity and low time complexity it can handle large data. On the other hand, Guyon and Elisseeff (2003) found that due to the fact that filter models score features independently of each other, the best features are not always chosen. This is because, a feature that is completely useless by itself thus ranked very low with this model, can provide a significant performance improvement when taken with others. *Wrapper models* address this limitation by evaluating subsets of features rather than individual features.

## *Wrapper models*

The aim of the *Wrapper model* is to achieve the highest predictive accuracy possible by selecting the features that accomplish this for a fixed learning algorithm. It uses a learning algorithm as a black-box

to select the best feature subset, agreeing on a predictive measure (Kohavi, 1995). Figure 33 shows a conceptual view of a wrapper model. The *Feature Selector* component takes on the original features and uses a search strategy to generate subsets of features for evaluation. The selected feature subset uses the learning algorithm to *measure* and evaluate its performance. These steps are repeated until a *stopping criterion* is met. Typically, the stopping criterion is that refinements to the subset do not yield any performance improvements.

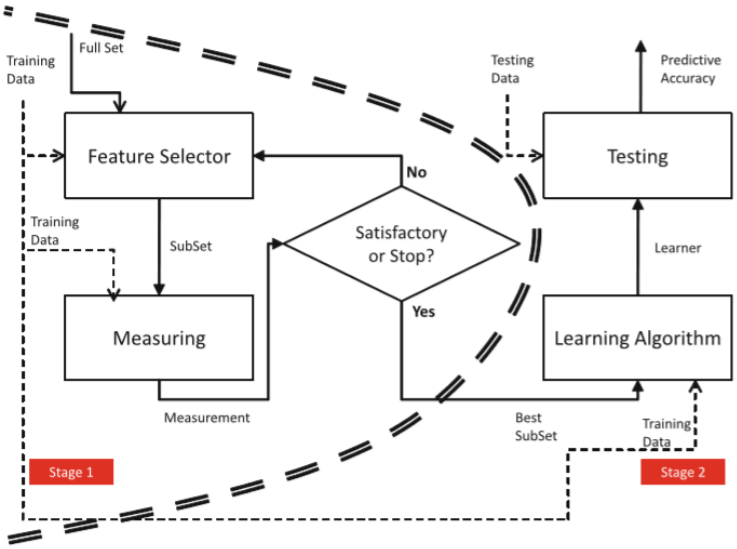



Figure 33: A general framework for Wrapper Methods of FS. Reproduced from Tang et al. (2014)

*Feature Selector* plays an important role in wrapper models because the order of the feature search space is $O(2^N)$ (Figure 34) making exhaustive search methods computationally unfeasible for large datasets. Therefore, search algorithms such as *Sequential selection algorithms* and *Heuristic selection algorithms*, described in section Search algorithms, need to be used (Chandrashekar and Sahin, 2014).

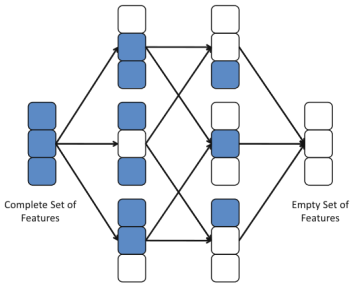



Figure 34: Search space for FS. Reproduced from Liu and Motoda (2012)

### *Search algorithms*

- **Sequential selection algorithms**: These algorithms are called sequential due to the iterative nature of the algorithms. Sequential forward feature selection (SFFS), sequential backward feature elimination (SBFE) and sequential bidirectional selection are greedy search algorithms that add or remove features one at a time (Liu and Yu, 2005). SFFS initiates with an empty set and SBFE initiates with a full set where as a bidirectional search initiates a SFFS and SBFE simultaneously ensuring that the features selected by SFFS are never eliminated by SBFE. The mentioned algorithms suffer from producing nested subsets since the forward inclusion is always unconditional which means that two highly correlated variables might be included if it gave the highest performance in the SFFS evaluation (Chandrashekar and Sahin, 2014). Sun et al. (2006) developed the Adaptive Sequential Forward Floating Selec-

tion (ASFFS) algorithm, an adaptive version of the SFFS, to avoid the nesting effect and produce a less redundant subset than the SFFS algorithm.

- **Heuristic algorithms**: One of the widely used *Heuristic algorithm* in FS is *Genetic algorithm* (GA); a general adaptive optimization search methodology that mimic Darwinian forces of natural selection to find optimal values of some function (Chandrashekar and Sahin, 2014). The algorithm starts by creating a set of candidate solutions referred to as chromosomes which form the first population. For each chromosome the fitness value is calculated. The fitness value is randomly impacted mainly by the *crossover* and *mutation* functions. Then based on the Darwinian principle of 'survival of the fittest' referred to *offspring*, the chromosomes with the best fitness values are combined randomly to produce the next population. This process is repeated again until acceptable results are obtained. For FS, a simple GA approach is to encode features in the chromosome (e.g. the chromosome 01001000 could mean that the 2nd and 5th features are selected), and the fitness value can be some measurement of model performance (De Silva and Leong, 2015). Huang and Wang (2006) used a chromosome with parameters of a *Support Vector Machines* SVM and the resultant solution provided the best features and the appropriate SVM parameters.

According to Sammut and Webb (2017), wrapper models generally achieve better recognition rates than filter models since the learning algorithm is involved in the selection of the subset, thus any bias introduced by the chosen algorithm is removed. The drawback of this model is that, wrapper models must train a classifier for each feature subset thus the method might become unfeasible.

### *Embedded models*

Some learning algorithms have in-built mechanisms to deal with feature selection (Guyon and Elisseeff, 2003; Sammut and Webb, 2017). This makes the selection of the features more efficient than the wrapper method and is specifically designed for the algorithm of choice. Another improvement on the other methods is that whilst the filter and wrapper methods select the features as part of the pre-processing of the data, the embedded models cater for feature selection as part of the learning process. One such example is the use of *L1-norm* regularisation in (SVM) (Guyon and Elisseeff, 2003; De Silva and Leong, 2015). Here the coefficients of less important features are reduced to zeros thus eliminating them. *L2-norm* regularisation can also be used for feature selection but was found to be less effective than *l1-norm* regularisation (De Silva and Leong, 2015). *Decision trees* also have a feature selection strategy embedded within the algorithm (García et al., 2015).

# *Generative Models and Networks*

*Definition*

A generative model is an unsupervised statistical model that can be used to generate similar samples from the learning of observable examples (Shin et al., 2017). These types of models make use of Naive Bayes rules in joint probability to calculate Equation 60:

$$p(y, x) = p(x)p(y|x) \tag{60}$$

With $x$ being the input feature and $y$ being the given class (Ng and Jordan, 2002).

Generative models are a concept which are currently limited in use due to the challenge of productively training them, where article such as Goodfellow et al. (2014b) state the difficulty of finding the best way to train them as the probabilistic computation required tends to be hard to gather the linear units in a generative context.

They are still sought after due to their heavy use in computer vision such as image caption generation (Lin et al., 2014; Touretzky et al., 1996); projects related to speech and language for example speech synthesis (Ou and Zhang, 2012) and conversation with dialogue generation (Sordoni et al., 2015). Generational models have the capability of data generation from the given sample with the understanding of the structure of the input data even without labels (Tu, 2007).

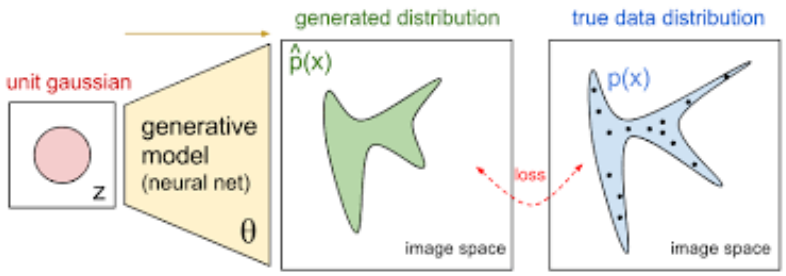



Figure 35: The layout of a generative model that shows the production of a generated product that is a variation of the original image space [7].

[7] blog.openai.com/generative-models/

*Generative Adversarial Networks*

Generative Adversarial Networks (GAN) are a type of generative approach that makes use of the creation of a network which consists of, the generator model which maps the latent space to the input space $x$ and with the introduction of a discriminative model. It will

be used to classify the generated data sample of the first model and calculates the probability of the sample that it came from the product of the first model or the actual training data. These models are trained simultaneously where the performance of the generative model is based on the probability of the discriminative model to make a mistake (Goodfellow et al., 2014b).

GAN is defined by the following objective function:

$$min_D max_G V(D,G) = \mathbf{E}_{x\sim\ pdata(x)}[logD(x)] + \mathbf{E}_{z\sim\ pz(z)} + [log(1 - D(G(z)))] \tag{61}$$

Where $D$ is the discriminative model and it is represented by a multilayered perceptron with parameters $D(x; \theta d)$ including a single scalar output. D($x$) is the probability that $x$ came from the data rather than the output of $G$, it represents the generative model which is also represented by a multilayered perceptron contains parameters $\theta g$. The generator's distribution $pg$ is learned by defining the input noise variables represented by $pz(z)$. This is then represented by the data space as G(z;$\theta$g). The two models are trained simultaneously where $D$ is trained to maximize the probability of labelling the data $x$ as either if it came from the provided sample rather than from $pg$, while $G$ is trained to minimize $log(1\text{-}D(G(z)))$, therefore GANs follow a minmax game (Goodfellow et al., 2014b).

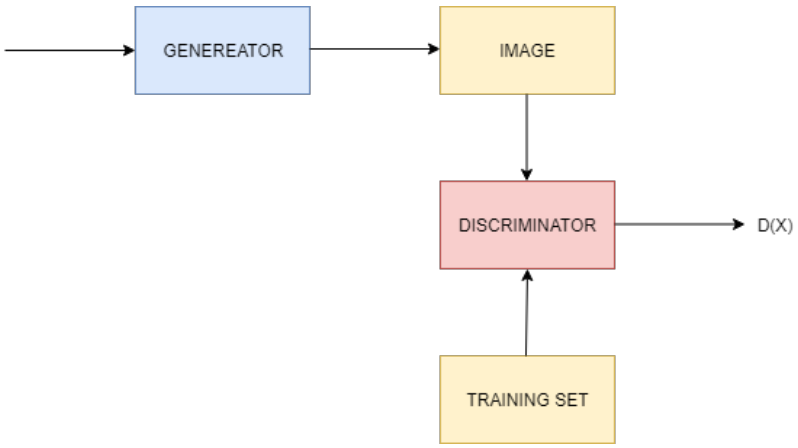



Figure 36: The components of a typical GAN system which includes a generative component that takes the dataset as inputs and the output is fed to the discriminator [8].

GANs is an area which can be used for reinforcement learning in several different ways,one example is the learning of a conditional distribution for the generation of future possible states based on the current state of the samples and with hypothetical actions which can be taken as input (Goodfellow, 2016).

## *Variational Autoencoders*

Variational Autoencoders are models that consist of neural networks of an encoder, a decoder with a loss function which can be trained using stochastic gradient descent. The encoder receives input data and produces a hidden representation of the same output with weight and biases. The encoder produces a latent vector space to lower the

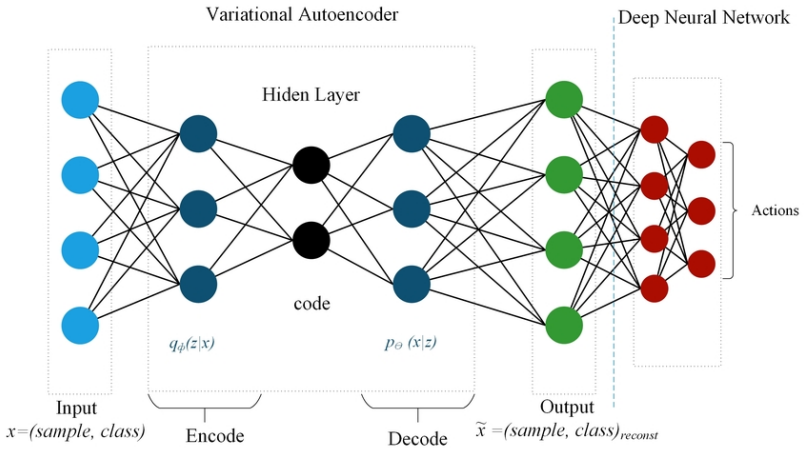



Figure 37: The structure of a Variational Autoencoder that shows the process of encoding and decoding with the production of the output.

dimensional space. The decoder consists of another neural network where the input is the output of the encoder and the product of this network is the parameters of the probability distribution of the data with its constituent weights and biases (Doersch, 2016).

Variational Encoders have become a very popular approach to tackle distributions with unsupervised learning and these models have already proved to tackle complicated distributions by generating data of different kinds of projects (Doersch, 2016).

The article by Jain et al. (2017) works on the generation of diverse visual questions wrapped around given images, this can be used for educational systems to AI assistants such as chat bots, driving assistance or for self entertainment. The author of the article thinks that this type of task is important for the reason of that it is wrapped around similarly to 'future prediction' and that it will solve the task of visual question and answering.

### *Boltzmann Machine*

A Boltzmann Machine is a type of stochastic recurrent neural network where it has a straightforward learning algorithm that provides the ability of using data sets that contain binary vectors to work out compelling features. They are heavily used to solve search problems where the dynamics of a Boltzmann machine allow the weights of a connection to be fixed so that they can represent the cost function of an optimization problem and binary state vectors will then be sampled to provide adequate solutions to the provided optimization problem (hinton, 2007).

The article by Nie et al. (2015) makes use of a generative restricted Boltzmann machine to be implemented and tested on two computer vision applications, mainly facial expression recognition and human action recognition. The model is made up of a Gaussian-Binary restricted Boltzmann machine to model high-dimensional motion data to obtain the likelihood of the model.

A Multimodal Deep Boltzmann machine was used in the article by

(Srivastava and Salakhutdinov, 2012) for a series of experiments and its performance was compared with other classifiers. This generative machine works by having two types of Deep Boltzmann machines which each include a number of hidden layers that are interconnected with each other only. These two types of DBMs have their own functionality and they are linked together by a common layer that joins the image and text inputs. The task at hand was the classification of images that contain different tags according to the content of the image. The DBM outperformed other machine learning algorithms such as an SVM and even an Autoencoder which was based on the works by Ngiam et al. (2011) and even other deep learning methods for example deep belief networks.

Generative models are a highly researched and powerful way of understanding and training a model with the data distributed. Articles such as Georghiades et al. (2000) state how their results and performance surpass those of other state of the art models in the implementation of facial recognition in different orientations.

# *Hyperparameters*

Li et al. (2017) state that 'performance of machine learning algorithms depends critically on identifying a good set of hyperparameters'. Hyperparameter tuning is a topic often discussed in literature describing any combination of machine learning algorithms. Adequate values set for hyperparameters will massively influence training and testing of a model, be it in terms of processing time, as well as the actual robustness of the model in question.

## *Introduction*

### *Defining Hyperparameters*

As explained by Bergstra and Bengio (2012), consider a learning model $A$ attempts to map a set of training data $X_{train}$ to a function $f$ which minimizes generalization loss. The model has a set of parameters which may be optimized during training, which are referred to as *model parameters*. The learning algorithm also has a set of parameters which define its architecture and how the system will behave. These may not be changed at runtime, and are referred to as *hyperparameters*. The training and testing performance of a function in relation to minimizing validation error depends heavily on selecting appropriate hyperparameter configurations.

## *Approaches for Hyperparameter Optimization*

Various methods exist to find optimal hyperparameter sets. A multitude of published works go into methods such as *grid search*, where one may refer to Duan and Keerthi (2005). *Random search* is another approach, explored in detail by Bergstra and Bengio (2012). Other approaches make use of *Bayesian Optimizations*, such as Snoek et al. (2012) and Snoek et al. (2015). *Gradient-Based* methods have also been used for hyperparameter tuning, in works such as Keerthi et al. (2006). An overview and comparison of a number of these approaches will be provided in the sections to follow.

### *Grid Search vs. Random Search*

Two common approaches for tuning hyperparameters are grid search and random search. Grid search is an exhaustive method which will

go through every possible combination of a set of hyperparameters, similarly to a brute-force methodology. The hyperparameter set with the best results is then kept and used for the model. Figure 38 shows grid search being applied on a system consisting of two hyperparameters.

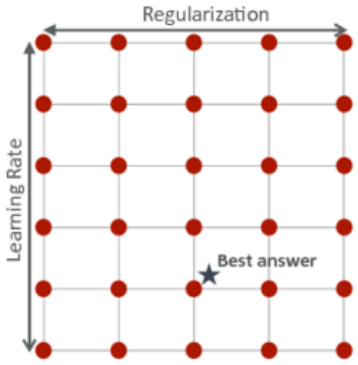



Figure 38: Grid search on a 2D hyperparameter space, adapted from http://prog3.com/sbdm/blog/google19890102/article/details/50276693 and researchgate.net

By observing Figure 38, it is clear that grid search is guaranteed to find a correct answer due to its exhaustive nature. In order to compare this method with a different approach, Bergstra and Bengio (2012) conducted a study to compare randomly sampled hyperparameters with those obtained through a grid search. Here they refer to a set of model coefficients $\theta$ and hyperparameters $\lambda$. As stated previously, the objective for a learning algorithm is to find the $\lambda$ which minimizes a loss function $f$. Figure 39 shows random search working within the same space as the previous image.

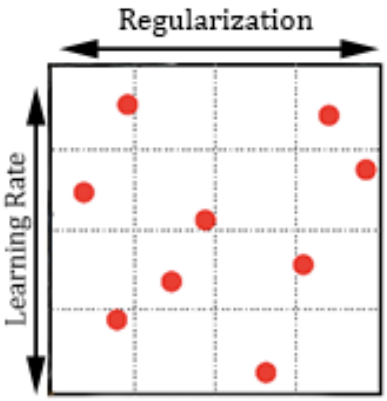



Figure 39: Random search on a 2D hyperparameter space, image adapted from Figure 38

Weighing both approaches together, random search could land on the best answer in the first trial, or the last. Consider the following example:

An SVM has several hyperparameters, one may consider $\gamma$, $C$, and the kernel type for the sake of this example. $\gamma$ defines the influence of observations on the model, $C$ affects the strictness of classification, and the kernel will affect the shape of the SVM's decision boundary.

These may be represented as follows:

$$kernel = [linear, polynomial, RBF]$$

$$\gamma = [0.0001, 0.001, 0.01, 0.1, 0.5, 1, 1.5]$$

$$C = [0.001, 0.01, 0.1, 1, 1.5]$$

For this very simple example, grid search would take $3 * 7 * 5 = 105$ runs of an SVM to check each and every hyperparameter combination. With random search, there is a possibility to discover a good configuration much faster, perhaps even instantly. This is particularly the case in real life examples where there may be many more hyperparameters than the ones presented. Since grid search makes use of every possible combination, the system suffers from the *curse of dimensionality*, described by Chen (2009) and Bellman (2015). This results in the algorithm complexity growing exponentially as the number of hyperparameters increase. This constrains grid search to being competitive only in lower dimensional spaces.

Thus, once a problem exceeds two hyperparameters, and sometimes even with only two, random search becomes vastly more efficient according to Bergstra and Bengio (2012). Nevertheless, studies such as Duan and Keerthi (2005) and machine learning libraries such as *libsvm* by Chang and Lin (2011), employed a combination of grid search and manual search. Knowing this, Bergstra and Bengio showed that random search actually generally outperforms grid search for hyperparameter optimization. This is true even when taking into account the element of luck required by random search, since hyperparameter trials are randomly sampled without any selection criteria. Snoek et al. (2012) states that this property provides random search yet another advantage grid search, since the algorithm can operate without caring about which hyperparameters are important to the model.

### *Bayesian Optimization of Hyperparameters*

Press (2005) defined Bayesian techniques as methods making use of Bayes' theorem to combine observational data with subjective beliefs. Bayes' theorem states that considering a hypothesis $H$ and evidence $E$, the probability of $H$ happening given that $E$ has occurred is calculated as follows:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

Bayesian optimizations provide a reasonable approach to modelling uncertainty. According to Snoek et al. (2015), they also provide a natural framework for optimizations of expensive black-box functions.Returning to the scope of hyperparameter tuning, one may

consider a hypothesis as a set of hyperparameters $\lambda$ and the evidence as data $D$. Bayesian approaches will keep $D$ static, and modify hyperparameter configurations $\lambda$ instead to see how the output changes. Essentially, the resulting data obtained through a certain hyperparameter set is then used to predict an expectation for different hyperparameter values. This may be repeated until convergence to a local optimum is reached. Regarding performance, Snoek et al. (2012) compared Bayesian approaches to an exhaustive grid search for optimizing hyperparameters of M3E models with three hyperparameters. The conclusion of the study was that the Bayesian approach was reasonably more efficient than grid search. This also makes sense

### *Gradient-Based Hyperparameter Optimization*

A particular project by Keerthi et al. (2006) investigates hyperparameter values in SVMs and Bayesian models. In this system, the variables $C$ and $\gamma$ are considered. $C$ is the regularization coefficient, influencing how leniently the SVM hyperplane classifies observations. Figure 40 and Figure 41 show how different values of $C$ influence a decision space for classification.

As previously stated, $\gamma$ defines the influence of points on the variance of a model. As expected, experiments in this study showed that performance of the SVM varied considerably as a function of hyperparameter values. Gradient-based optimization techniques were used, computing the slope of a performance validation function with respect to the vector $h$ of hyperparameters. The gradient-descent approach attempts to minimize the cost function of the hyperparameter problem by converging to local minima. To reach this goal, the gradient of the function decreases with each iteration of the algorithm. This will update the coefficients $\beta$ of the system, which will then be applied to the feature vectors in the next iteration. This process is repeated until convergence towards a local minimum is reached. This system allowed for a fast method to determine optimal sets of hyperparameters, which also outperformed grid search even for models with a low dimensionality of two hyperparameters.

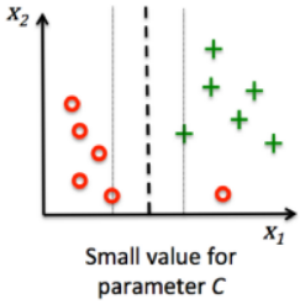



Figure 40: Small $C$

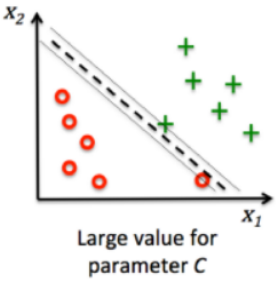



Figure 41: Big $C$, both images from https://www.bogotobogo.com/python/scikit-learn/scikit_machine_learning_Support_Vector_Machines_S

# *Long Short-Term Memory*

Some problems in Machine Learning, especially NLP-focused tasks, generally require some sort of **context** from previous iterations to form a more comprehensive understanding of the problem it is being given.

When facing context-heavy sentences such, as, "I am from Malta, and I speak *Maltese*", most models became unable to forge enough contextual clues in order to determine the nationality despite such clues, or **long-term dependencies**, being given earlier in the sentence (Bengio et al., 1994).

In order to solve this problem, Hochreiter and Schmidhuber (1997) proposed the Long Short-Term Memory network, or LSTM.

## *LSTM Architecture*

This module is a type of Recurrent Neural Network (RNN), that is, a repeated chained module, with a gradient-based unit applied. The main difference is that the repeated module itself is different, with the LSTM module being composed of a series of gates named the **input gate**, **hidden gate**, **forget gate**, and **output gate**, as described in Figure 42.

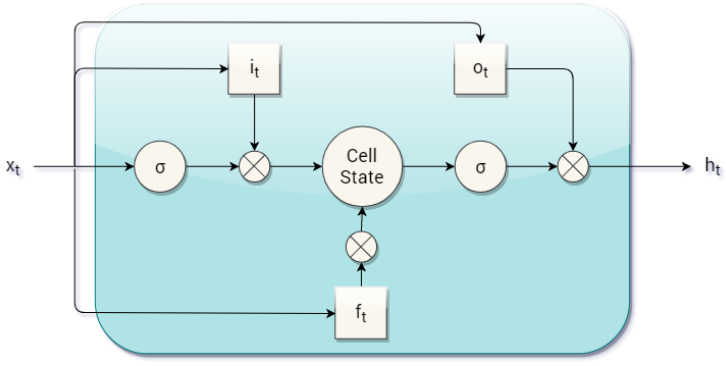



Figure 42: Architecture of standard LSTM module. Adapted from: Olah (2015)

The traversal from one gate to another is handled through a number of **activation functions**, which have been discussed in a previous chapter. The LSTM framework uses three main functions:

$\sigma_g$ - denoting the Sigmoid Function (Rumelhart et al., 1986). $\sigma_c$ - denoting the Hyperbolic Tangent Function. $\sigma_h$ - also denotes the Hyperbolic Tangent Function in the case of the hidden gate, and implies that $\sigma_h(x) = x$.

In order for the LSTM module to learn, backpropagation is implemented in the form of the Constant Error Carousel (CEC) function, which updates the hidden gate $h_t$ through the following equations:

- $h_t$ - the hidden state, denoted by

$$h_t = o_t \circ tanh(c_t) \tag{62}$$

  where $\circ$ implies the Hadamard Product (element-wise multiplication).
- $o_t$ - the output gate, denoted by

$$\sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \tag{63}$$

- $c_t$ - the cell transfer state, denoted by

$$f_t \circ c_{t-1} + i_t \circ tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{64}$$

- $f_t$ - the forget gate, denoted by

$$\sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \tag{65}$$

- $i_t$ - the input gate, denoted by

$$\sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \tag{66}$$

These functions are not only the foundation of the LSTM module, but it is processed in such a way that it also reduces the Vanishing or Exploding Gradient Problem - the problem in which gradient descent begins to converge to zero or infinity, leaving the results almost unchanging in value (Hochreiter et al., 2001). They are typically trained by Connectionist temporal classification (CTC) Score functions (Graves et al., 2006), similar to how training and testing work in normal Neural Networks with the additional function of handling time-variable problems. It works by taking the input as the recorded observations, sequential labels as an output, and proceeds to estimate a probability distribution for the sequence of inputs against outputs for time. These processes take place for every LSTM module present in the global network, sending their output to the next LSTM module which can be seen in Figure 43.

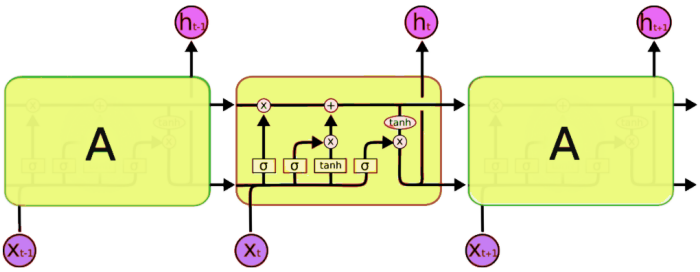



Figure 43: Typical LSTM Network. Retrieved from: Strivastava (2017)

## *Structural Variants*

Since their introduction in 1997, many researchers have refined and improved this structure to suit different applications, with the following variants being the most notable amendments.

*Peephole LSTM*

First introduced by Gers et al. (2003), the Peephole LSTM variation is very similar to the typical LSTM structure, with the only core difference being that each cell is able to look into the current CEC values for each gate, allowing for more control into the network as show in Figure 44.

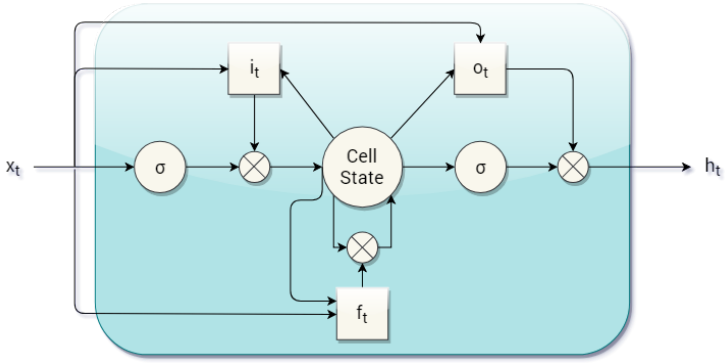



Figure 44: Peephole LSTM module variation, Adapted from Graves et al. (2013)

*Convolutional LSTM*

Another variation of LSTM first proposed by Shi et al. (2015), which used the LSTM's long-term dependency property in conjunction with a Convolutional Neural Network in order to process multiple subsequent image frames and retain contextual knowledge of different scenes. Figure 45 demonstrates the operation of Convolutional LSTMs.

The operations which take place are the following:

- $h_t$ - The hidden gate, denoted by $o_t \circ \sigma_h(c_t)$
- $c_t$ - The current cell state, denoted by $f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c * x_t + U_c * h_{t-1} + b_c)$
- $o_t$ - The output gate, denoted by $\sigma_g(W_o * x_t + U_o * h_{t-1} + V_o \circ c_{t-1} + b_o)$
- $i_t$ - The input gate, denoted by $\sigma_g(W_i * x_t + U_i * h_{t-1} + V_i \circ c_{t-1} + b_i)$
- $f_t$ - The forget gate, denoted by $\sigma_g(W_f * x_t + U_f * h_{t-1} + V_f \circ c_{t-1} + b_f)$

*Gated Recurrent Units (GRU)*

This variation, first proposed by Cho et al. (2014), is very similar to the LSTM architecture in that it also a gated network, depicted in Figure 46. It had been proposed as an alternative to LSTM's to handle problems within the same domain (i.e. Speech Synthesis, for example). The main architecture of a GRU consists of a fully gated unit, where for time $t = 0$ and output $y_t = 0$, the output vector $h_t$ is defined as:

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ (W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \tag{67}$$

where

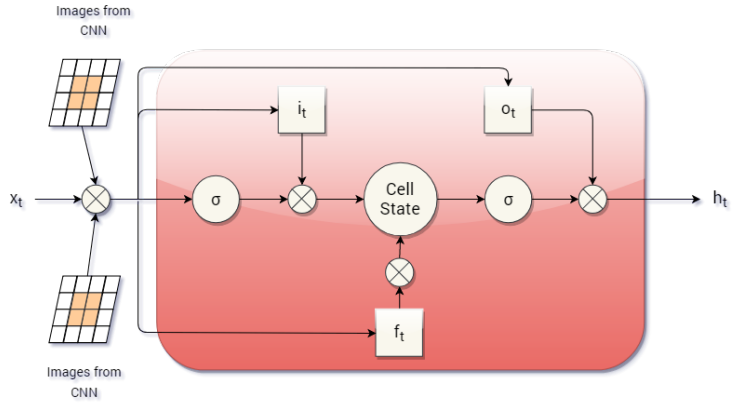



Figure 45: Peephole Convolutional LSTM module variation. Adapted from Gers et al. (2003)

- $x_t$ is the input gate
- $z_t$ is the update gate, denoted by the function $\sigma_g(W_z x_t + U_z h_{t-1} + b_z)$
- $r_t$ is the reset gate, denoted by the function $\sigma_g(W_r x_t + U_r h_{t-1} + b_r)$

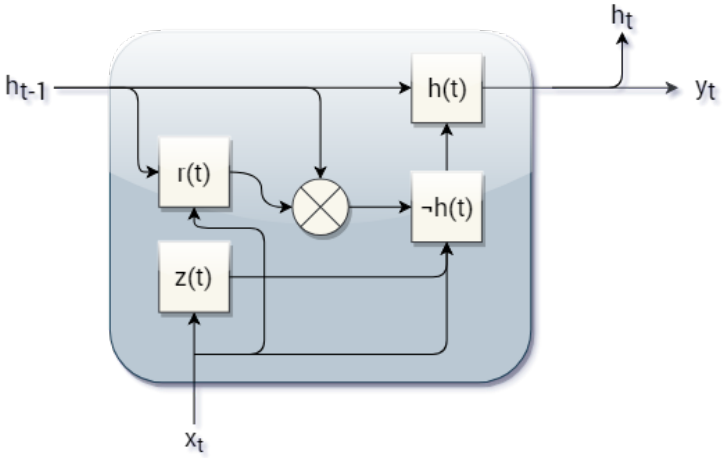



Figure 46: Structure of GRU, based on figure by Le et al. (2016).

In conclusion, LSTMs are an excellent deep learning tool for time-series problems such as Natural Language Processing, where sentence fragments require memory (Graves et al., 2006; Gers and Schmidhuber, 2001; Schmidhuber et al., 2005; Schäfer et al., 2006). However, those are not the only applications of LSTMs. Other applications can involve speech recognition (Saon and Picheny, 2017), handwriting recognition (Graves et al., 2009), patient subtyping (Baytas et al., 2017), and many more applications. Handling context is very important in many domains, and as such, LSTMs have been and will continue to be used and improved on to support the ever-growing problems in Machine Learning.

# *Loss Functions*

The driving force of any mathematical optimization problem, especially in machine learning, is the loss function, measuring how well an algorithm is performing with regards to its target. In general terms, the evaluation for any learning problem is done by taking a predicted or estimated output and compare it to the relative real recorded outcome, typically in a dataset. The expected value of the loss is called the statistical risk. The formulation of the loss function is crucial and must be accurately defined according to the selected algorithm and the selected data for the particular optimization problem. The loss function must reproduce a numerical value that somehow reflects the level of achievement desired by the algorithm. In machine learning we are typically attempting to find the parameters that fit our model and that minimizes the loss function. Minimizing the loss function is one of the fundamental foundations of statistical machine learning (Yang et al., 2014).

The three canonical loss functions as described by Vapnik (Vapnik Vladimir, 1998) are:

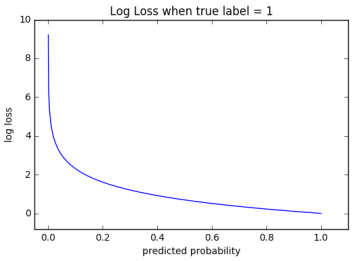



Figure 47: Log Loss Function

- The 0-1 Loss âĂŞ the simplest loss function mainly used in binary classification

$$L(y, \hat{y}) \tag{68}$$

, resulting in an error of 0 if and only if $y == \hat{y}$. Otherwise the function will produce the value of 1. This does not however differentiate between different classes and types of errors. Variations of this function may thus be formulated by extending the simple loss function and make the resulting loss dependent on another function.

- The squared loss âĂŞ the equation used for regression defined as

$$L_{sq}(y, \hat{y}) \tag{69}$$

where the resulting error value is given by

$$(y - \hat{y})^2 \tag{70}$$

This is often the loss function of choice used in estimators such as linear regression and in many areas of machine learning (Davidson-Pilon, 2015). One disadvantage of using the squared loss function is that it is sensitive to outliers, since the loss increases quadratically as the estimate moves away from the true value (Davidson-Pilon,

2015). A loss function which is less sensitive to outliers is the absolute-loss function, given by:

$$L(y, \hat{y}) = |y - \hat{y}| \tag{71}$$

In this case, only the absolute value is taken of the difference between the estimated and real value. However this equation is not smooth where $(y - \hat{y})$ and thus may prove to be difficult to differentiate.

- Finally the probability estimation via the log loss is given by:

$$L_{log}(\hat{p}) = -ln(\hat{p}) \tag{72}$$

In this case rather than simply classifying the outputs into the different classes, we may also require the confidence level that a certain input belongs to a particular class. A variation of the log loss is the cross entropy which provides the possibility of classifying input into more than two classes. A depiction of the log loss function is given in Figure 47

In general loss functions must satisfy certain properties. Primarily they should not be expensive to compute (Scholkopf and Smola, 2001) have only a small number of discontinuities in the first derivative and be convex in order to able to find a unique solution. In the case of regression problems another desirable property is resistance to outliers.

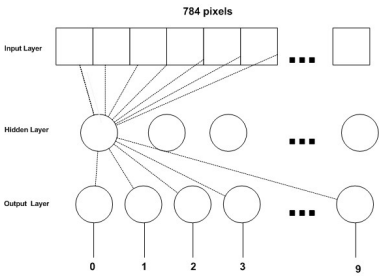



Figure 48: Example Neural Network with Cross Entropy Loss Function

### *Cost Functions*

Cost functions are generally based on loss functions which have been adapted to calculate the loss over a number of training examples. In this case it is assumed that there are $n$ samples or training examples for which an approximation for each of the errors $e$ in our model is to be calculated. Thus a cost function can be interpreted as representing a total error over all corresponding predictions for the training examples. Cost functions may also include penalty terms, such as L1 or L2, for avoiding overfitting or underfitting. Some of the most widely used cost functions in regression are briefly explained here (Chai and Draxler, 2014).

### *MSE*

The Mean Squared Error for a dataset with $n$ training samples is given by summing the error $e$ for for all examples and calculating the squared difference from the true value $y$ and the prediction value $\hat{y}$.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y - \hat{y}) \tag{73}$$

### *RMSE*

RMSE has been widely used as a metric for model performance in meteorology, air quality and climate research study. The calculation

is done by firstly obtaining the total square error which is the sum of the individual squared errors. Total square error is then divided by $n$. Large errors in this case have a greater influence on the total square error rather than smaller errors. While the rationale for squaring the individual error is to eliminate the sign (Willmott and Matsuura, 2005). The formula is given below :-

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y - \hat{y})} \tag{74}$$

*MAE*

Calculation of MAE is simple, involving the summation of the absolute errors to the obtain the total error, then dividing by the total $n$. The formula is given below :

$$MAE = \frac{1}{n}\sum_{j=1}^{n} \mid y - \hat{y} \mid \tag{75}$$

*Cross Entropy Loss*

In terms of classification cost functions, Cross Entropy is the most frequently used.

$$L(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{76}$$

A practical example of Cross Entropy function as applied to Deep Neural Networks is now described. Deep Neural networks are commonly good performers when used as classifiers, especially since these can be easily manipulated by changing their architecture, interconnections and activation functions according to specific requirements. However for such implementations the loss function of choice is almost always log loss, which is applied through softmax at the output layer (Janocha and Czarnecki, 2017). Lets assume, as an example, a dataset containing a large number of hand written digits from 0 to 9 as images and it is required to build a model to recognize the hand written digits and categorize each input into 10 different classifications. The structure of the neural network used for this example is explained in Figure 48. Each image of hand written digits is 28 x 28 pixels large, thus the input layer contains 784 neurons, while at the output layer we would require 10 neurons, each representing a digit from 0 to 9. The cost function that will be used in this example is the Cross Entropy function, as described in 76.

However, this formula is only suitable for one training example. Generally the weights for a deep neural network are updated after calculating the loss of a batched number of training examples. In this case the cost would need to be averaged over all training examples, and thus the function would be rearranged as follows:

$$L(Y, \hat{Y}) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right) \tag{77}$$

In neural networks, it is required to compute how much the weights $w$ for each layer $j$ affect the loss $L$. This can derived by finding the partial derivate of the cost function with respect to the weights, as follows:

$$\partial L / \partial w_j \tag{78}$$

However it is generally not possible to find the partial derivative as described above directly. Instead by applying the chain rule it can be found how much the weights affect the activation functions for each layer and in turn how much these activation functions ultimately affect the loss. This can be formally described as follows:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_j} \tag{79}$$

After finding the partial derivative of each function as described, the partial derivative of the cost function with respect to the weights can be substituted and finalised:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_j} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \hat{y}(1 - \hat{y}) w_j = (\hat{y} - y) w_j. \tag{80}$$

# *Neural Networks*

Neural Networks (NNs) are the product of various academic facets, the most prominent one being neuroscience as the structure of a neural network mimics the computation done by the human brain, which makes use of neurons to perform complex calculations such as object recognition from visual input of the eyes. This process is very inefficient to be performed on a computer due to its computational difficulty, which is remedied by the neural network's parallel nature. Neural Network learns with a supervised approach, supplying the model with training datasets in order to adjust the synapses to match the output. The trained model can then be used on unseen data to produce predictions (Guresen and Kayakutlu, 2011).

The way the brain trains its neurons is through experience and thus creating a learning process, and neural networks are designed to work the same way by creating synthetic nodes which will behave like organic neurons. Since this structure makes use of learnt experiences, the progress is stored in the form of weighted values at the synapse of each node (Haykin, 1994).

## *Types of Neural Networks*

### *Artificial Neural Network*

An Artificial Neural Network (ANN) (see Figure 51) can be defined as a collection of Machine Learning techniques that together are able to process complex inputs into meaningful spaces or classifications without any further input from the user (such as rules). The ANN is a network of layers, where every node is connected to each node of the previous and next layers. A common setup for an ANN will have an input layer, hidden layer and output layer. Neural Networks are capable of solving both regression (continuous label) and classification (discrete classes) problems, although the main purpose of using these models are for classification purposes, which is highly influenced by the number of nodes in the output layer (Devulapalli, 2015).

With regards to hyperparameters, the neural network contains parameters that require many trial and error to discover their optimal value. One of which is the number of hidden layers and nodes, as a small value can result in underfitting and the model would not have enough complexity to learn the dataset, and a large number will result in overfitting, where the model would be able to learn the dataset

by rote. The latter is a very common issue with neural networks and can be mitigated with dropout procedure. This method assigns a probability to each node during training in order to periodically deactivate it, thus eliminating the possibility of the next node to base its calculations on the deactivated node, thus eliminating overfitting (Srivastava et al., 2014a).

Another hyperparameter is the learning rate, which is used as a scale to which the model updates its weights. The learning rate should not be very large, as this could overshoot the local minimum and never converge. The ideal way is to use a decaying learning rate in order to alleviate the problems of having a large learning rate as well as small, while still keeping their benefits Jacobs (1988). One method to find out hyperparameters is to perform Bayesian Optimization, which builds a probability model on the dataset and selects the hyperparameters with the best score Eggensperger et al. (2013).

As the ANN is traversed, an activation function is fired at each step. Activation functions form the output behaviour of each of the neurons, by applying a specific function to the input nodes and producing the desired output. The activation function may contain a numerical bias, which will transform its output upwards or downwards (Jain et al., 1996). Examples of activation functions are as follows:

- **Sigmoid function**:

  The Sigmoid function is an improvement over the step function as it smoothens out the steep change between the two states. This function maps the input to a value between 0 and 1, both ends being an asymptote. The Sigmoid function shown in Figure 49 is as follows (Haykin, 1994):

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

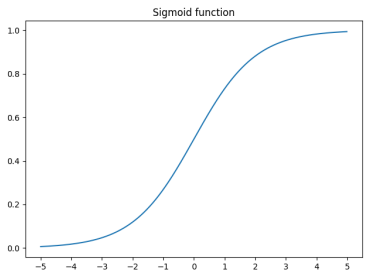



Figure 49: The sigmoid function.

- **Hyperbolic Tangent function**:

  The Hyperbolic Tangent function (Tanh) is very similar to the Sigmoid function, with a more stable gradient and an asymptotic range between -1 and 1, thus including negative values. The function shown in Figure 50 is the following (Abdelsalam et al., 2017):

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

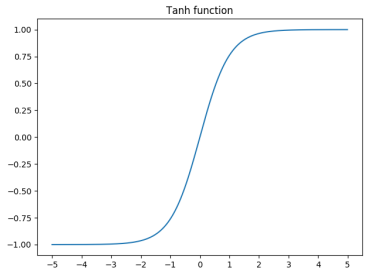



Figure 50: The Tanh function.

The traversal goes from input nodes to the output nodes, and the output will be a mapped function of the input, firing activation functions at each step. Before performing any predictions, the ANN needs to be trained with labelled data in order to adjust the synapses to their optimal values to form the mapping (Guresen and Kayakutlu, 2011).

The goal of the learning process is to minimize the output of a cost function, taking into consideration the error value. Each time the input is fed forward through the network, the error of the predicted

output is compared to the actual output and thus produce an error value. The cost function $C(n)$, in terms of error $e_k(n)$ is defined as $C(n) = \frac{1}{2}e_k^2(n)$.The error delta in terms of weight $w_{kj}$ is calculated with the following formula: $\Delta w_{kj}(n) = \alpha e_k(n)x_j(n)$ where $\alpha$ is the learning rate, $e_k(n)$ is the error and $x_j(n)$ is the input. (Dawson and Wilby, 1998).

Back-propagation is the process where the biases are updated by distributing the error from the layer closest to the output inwards, applying the derivative of the activation function in the process. The synapses are updated by adding the calculated delta to the weight, formally, $w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$ where $w_{kj}(n)$ is the previous weight and $\Delta w_{kj}(n)$ is the calculated weight difference (Rosen and Goodwin, 1994). Each layer will calculate the error based on the previous error calculations and this will cause a diminishing effect on the error value, leading towards the vanishing gradient problem. Hochreiter (1998) points out that this will cause the calculated error delta to be so small that the improvement will be nearly negligible, thus causing the model to be under-trained. One of the remedies to the above phenomenon is to implement Long short-term memory to the model. As the ANN performs epochs, the error of the synapses will converge towards the optimal value, which once reached, the model will be trained. Once the model is trained, one can perform predictions by passing an input through the network and observing the output (Taşdemir et al., 2008).

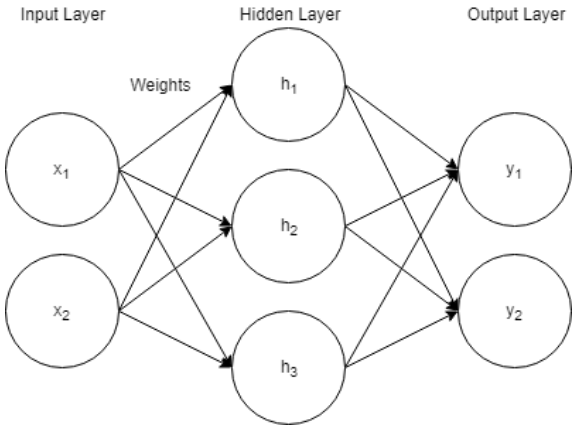



Figure 51: This diagram is a visual representation of an ANN.

### *Convolutional Neural Network*

A Convolutional Neural Network (CNN)(see Figure 52) is a variant of ANN and are commonly used for image processing and recognition. A CNN usually is a multi-layered network, where each layer will apply a specific function to an image (Krizhevsky et al., 2012b). A CNN can use different kinds of layers, each with their own function and output, for example:

- **Features**: The images are reduced to similar patterns, where each pattern is a small subset of the image and is present in both images

(Rohrer, 2016).

- **Convolutions**: A filter is applied across every pixel to calculate the similarity between the filter (also referred to as a kernel) and the sample with a convolution function. This will output a value that represents the likeness of the feature to the image; the higher the number, the higher the similarity (Alwani et al., 2016).
- **Average/Max Pooling**: This process shrinks the images while still keeping its important features. A small window which is typically 2x2 iterates over pixels, stepping a specific number of pixels each time to possibly avoid overlapping calculations. The output of this filter is the maximum or average value inside the window Ṗooling greatly helps to reduce the computational overhead of processing a large number of pixels and thus converge in less time (Fei et al., 2018; Cireşan et al., 2011).
- **Rectified Linear Units**: A Rectified Linear Unit (ReLU) is considered as an activation function, in that the output is filtered to contain only positive values. This function sets all negative values to 0 and ignores positive ones. This is used to prevent learned values from lingering close to 0 or becoming such a large negative number that it will inhibit the performance (Han et al., 2017).
- **Fully-Connected Layer**: A fully-connected layer (FC) treats the input as a singular list of nodes where each node is connected to all the previous nodes. Each node contains a weighted value which determines the contribution to a specific category when the node is fired. When the layer is activated, the nodes go through a voting process, where each node will vote to which category the input image falls under. The weights will determine which nodes will have more impact in the voting process, and the category with the bigger value of votes will classify the image (Xie et al., 2017).

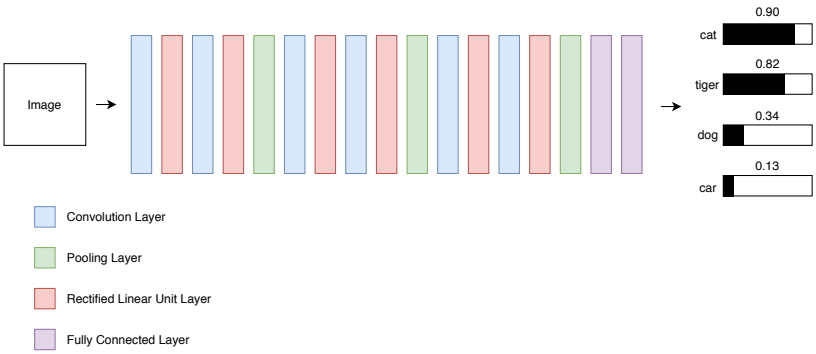



Figure 52: A diagram of a CNN with deep learning.

A CNN can have multiple repetitions of a set of layers, which will form a deep learning model with a multitude of layers that will eventually produce a final classification of the image. Training of a CNN occurs by feeding the network labelled data and adjusting the weights and features during a backpropagation step, distributing the error and converging to a set error threshold, similar to how ANNs perform their training (Iizuka et al., 2016).

# *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 8 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 8: 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 53 that shows the effect of noise on the classification accuracy of various classifiers.

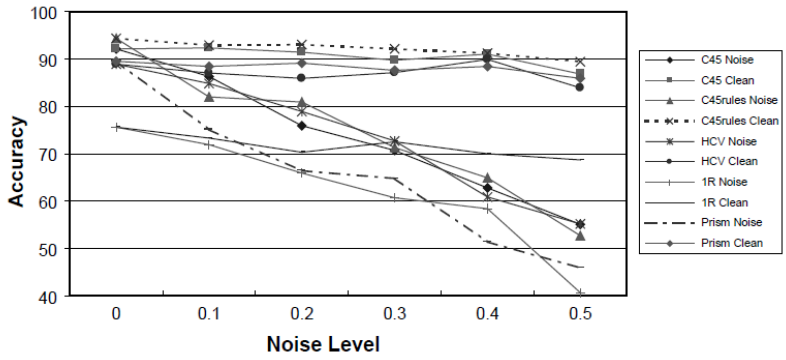



Figure 53: A graph showing the effect of noise on the classification accuracy of various classifiers. Each model was trained on a noisy dataset and on a manually cleaned dataset. These are represented as *XXX Noise* and *XXX Clean* respectively, where *XXX* denotes the classifier in question (Zhu and 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 54 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 81) and the number of incorrect instances that are not filtered (*Type II errors* -

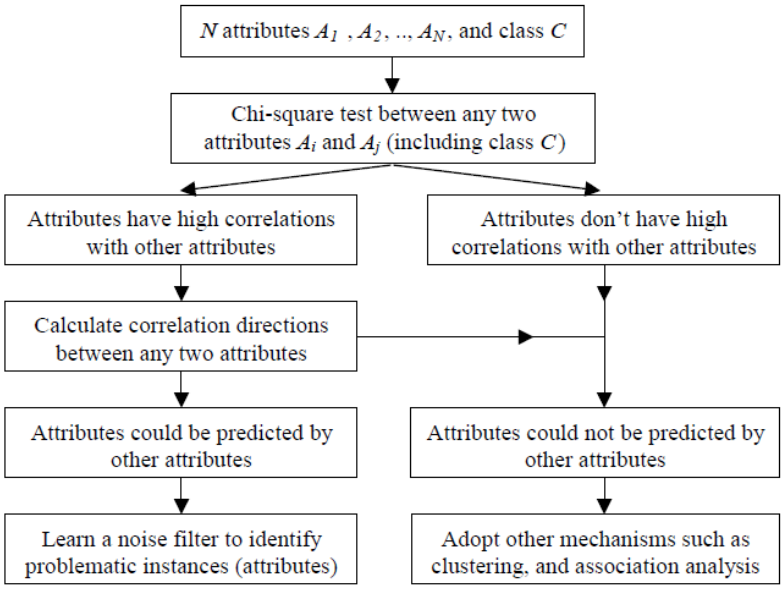



Figure 54: The approach adopted by Zhu and Wu (2004) to filter and correct attribute noise.

Equation 82). These are calculated as follows:

$$ER_1 = \frac{\text{\# of correctly labelled instances which are removed}}{\text{\# of correctly labelled instances}} \tag{81}$$

$$ER_2 = \frac{\text{\# of mislabelled instances which are not removed}}{\text{\# of mislabelled instances}} \tag{82}$$

Consequently, the *Noise Elimination Precision* (NER) can be calculated through Equation 83 (Frénay and Verleysen, 2014).

$$NER = \frac{\text{\# of mislabelled instances which are removed}}{\text{\# of removed instances}} \tag{83}$$

*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*

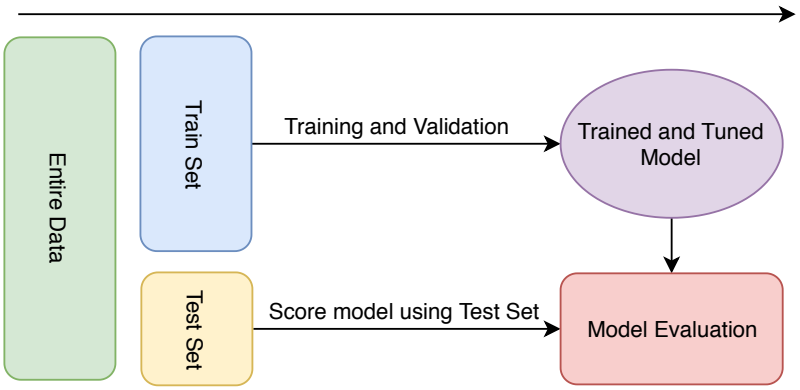



Figure 55: The traditional batch train-test machine learning approach workflow.

In the traditional machine learning approach depicted in Figure 55, 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

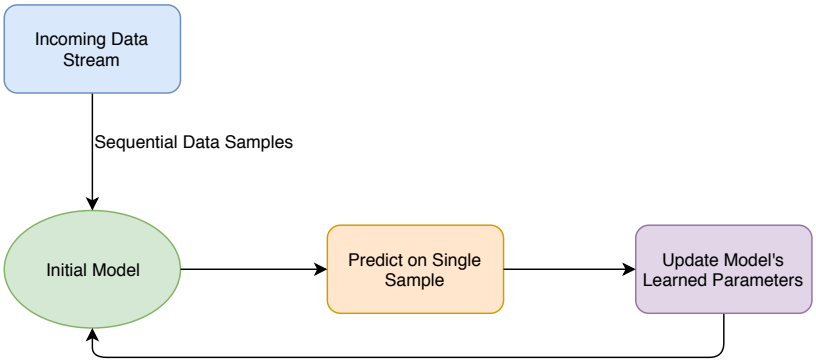



Figure 56: Basic workflow architecture of online learning.

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 56), 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 57:

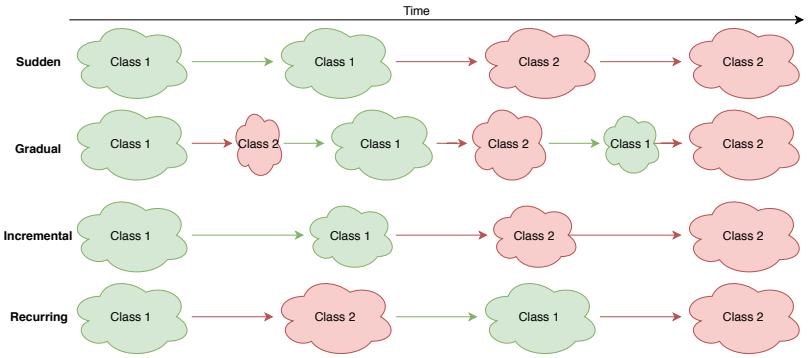



Figure 57: The four types of Concept Drift.

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 58 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.

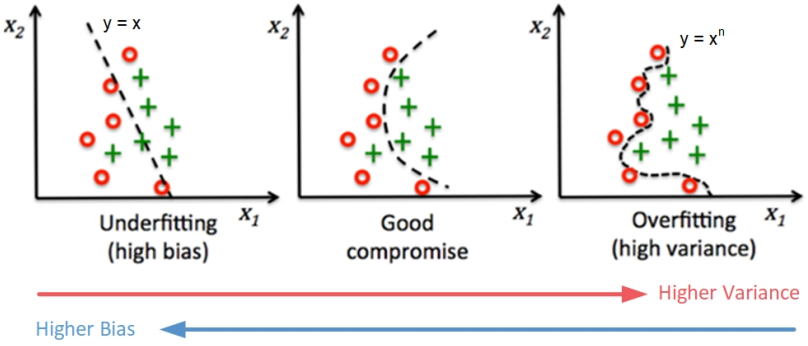



Figure 58: Overfitting and Underfitting: Challenge for a good bias-variance trade-off (Adapted fromTaschka (2015)).

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 58. 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 $\lambda$ to add the bias or penalty to the error function as the model complexity increases. Function 84 defines a simplified loss/cost function with regularisation.

$$minimize(costfunction, c(x) + \lambda(penaltyfunction)) \tag{84}$$

Function 84 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 $\lambda$ is determined empirically or through cross validation, and it is used to control how to balance out the two terms in function 84 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 84 can be re-written as:

$$minimize(\sum_{i=1}^{n}(y_i - w_0 - \sum_{j=1}^{p}(x_ij.w_j))^2 + \lambda \sum_{j=1}^{p}(w_j^2)) \tag{85}$$

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 $\lambda$ 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:

$$minimize(\sum_{i=1}^{n}(y_i - w_0 - \sum_{j=1}^{p}(x_ij.w_j))^2 + \lambda \sum_{j=1}^{p}(w_j)) \tag{86}$$

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. (2014b). 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 59,60.

The random dropout action is performed during the training phase only, and it creates a different thinned NN (Figure 60) for each epoch. Therefore the minimisation, through back propagation, of the NN loss function is only applied to the thinned network, and the inactive

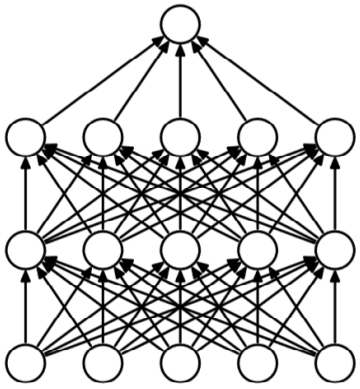


Figure 59: Standard Neural Net (Reproduced from Srivastava et al. (2014b)).

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. (2014b) 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

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. (2014b) 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.

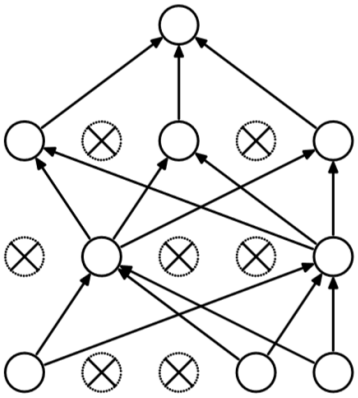


Figure 60: Thinned Neural Net after applying dropouteproduced from Srivastava et al. (2014b)).

## *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 61 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.

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 61. 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.

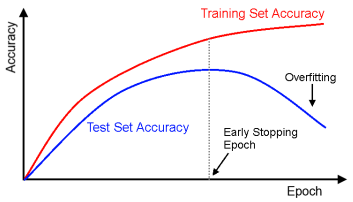



Figure 61: Training Data Accuary Vs Test Data Accuracy

# *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)

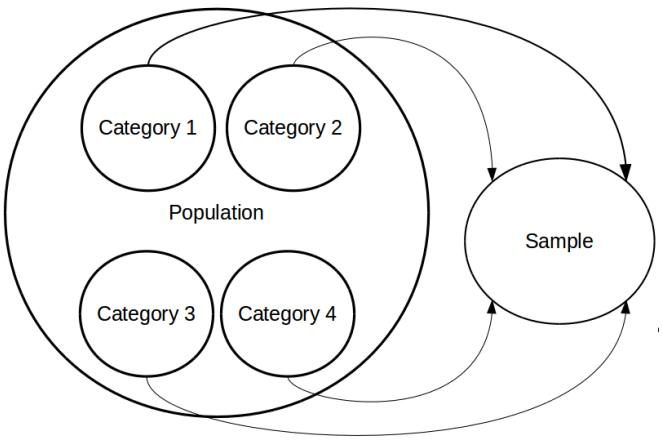



Figure 62: Stratified sampling: splitting the population into categories and selecting accordingly

### *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)

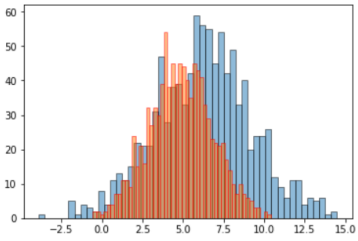



Figure 63: Comparing Distributions

### *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.

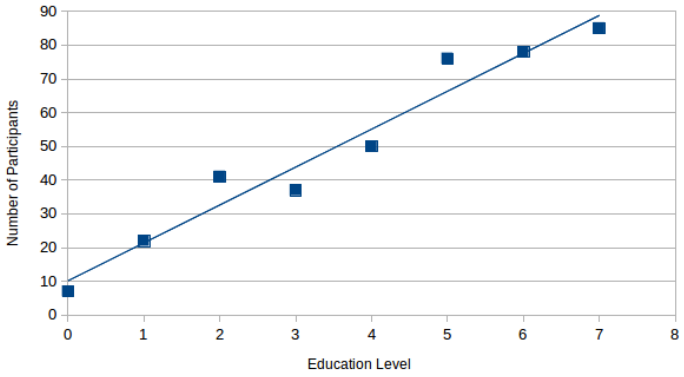



Figure 64: Two Step Estimator: Using regression models to determine correlations between participants and non-participants

## *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:

$$R(h) = \sum_{i=1}^{m} w_i c(h, z_i) \tag{87}$$

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.

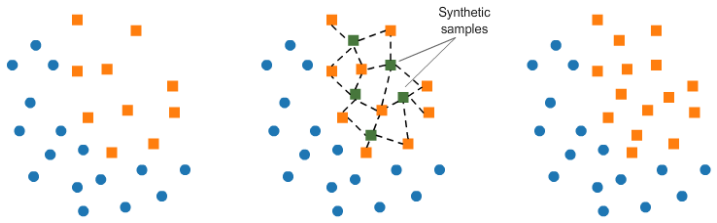



Figure 65: SMOTE: generating synthetic samples of the minority class

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.

# *Semi-Supervised Learning*

Supervised learning involves a labelled dataset on which a classifier is trained. Once trained, it is then used to predict the labels of similar unlabelled data. Unsupervised Learning deals with the identification of patterns in a given unlabelled dataset.

Semi-supervised learning lies somewhere in between these two techniques, blending them together. It makes use of both labelled and unlabelled data and is often used in applications where labelled data is difficult to come by. The goal of the semi-supervised classification method is to train a classifier on both labelled and unlabelled data, with the aim of getting a better result than that provided by a supervised classifier (Zhu and Goldberg, 2009).[9]

[9] It is also worth noting that semi-supervised methods do not exist only for classification, but also for regression problems.

## *Semi-Supervised Classification and Clustering*

Similar to Supervised Learning, Semi-Supervised Learning can be split into two categories; Semi-Supervised Classification and Semi-Supervised Clustering. We now take a closer look at each of these two areas.

### *Semi-Supervised Classification*

Semi-Supervised Classification is a classification problem which makes use of both labelled and unlabelled data in the same dataset. As is typical of such a problem, we assume that the volume of unlabelled data in the dataset is larger than that of labelled data.

In semi-supervised learning, we make use of Pseudo-Labelling (Lee, 2013) to increase the amount of labelled data upon which the classifier is being trained. The process is as follows:

1. We first use the smaller portion of labelled data to begin training our model.
2. We then use this model to predict the labels for the unlabelled data in the dataset.
3. The model is re-trained on all of the labelled data, including the original labelled entries as well as the new pseudo-labelled entries.

4. Steps 2 and 3 are then repeated for any unlabelled data left (if any).

Hence in this way, semi-supervised classification offers the same performance as a Supervised Classification, with the added benefit that the unlabelled data is automatically labelled by the classifier itself, reducing the effort needed for manual labelling in the dataset.

### *Semi-Supervised Clustering*

Semi-Supervised Clustering, also known as Constrained Clustering, can be considered as a supervised extension added to Unsupervised Clustering (Zhu and Goldberg, 2009; Bradley et al., 2000).

In such a case, the dataset in question consists of unlabelled data, the same as a typical Unsupervised Clustering problem. Distinctively however, in Constrained Clustering one also finds a degree of supervised information about the data clusters inside of the dataset. Such information may contain constraints such as *must-link* and *cannot-link*, where in the former, two data elements $x_i$ and $x_j$ must be in the same cluster, while in the latter they must not (Zhu and Goldberg, 2009). Using Constrained Clustering we aim at clustering better than a typical unsupervised clustering technique.

### *Method*

Initially, it may seem illogical that a semi-supervised process making use of unabelled data can perform as good as or better than a supervised labelled solution. Unlabelled data is incapable of providing a relationship between an element $x$ and a label $y$, which is what a Supervised Model is trained upon. What gives Semi-Supervised Learning its strength is the assumptions made between the unlabelled data and the target labels.

To examine the discussed process, let us take an example proposed by Zhu and Goldberg (2009). First we represent each data instance by a one-dimensional feature $x \in \mathbb{R}$. $x$ can be in one of two classes; positive or negative.

- In a case of Supervised Learning, we are given the labelled training instances $(x_1, y_1) = (-1, -)$ and $(x_2, y_2) = (1, +)$. In such a case the best Decision Boundary would be $x = 0$, where all instances having $x < 0$ are classified as $y = -$, and those having $x \geq 0$ as $y = +$.

- Now let us consider also a large number of unlabelled instances with unknown correct class labels. We observe however, that they form two groups. Taking the assumption that instances in each class form a coherent group, we know that instances tend to center around a central mean in a Gaussian Distribution.

- Through this assumption, the unlabelled data is capable of providing us with more information. We now notice that the two labelled instances detailed in the first point are not the best examples for each of the classes in our dataset.
- Hence we take a semi-supervised estimate, which shows us that the Decision Boundary should be between these two latter groups instead, at $x \approx 0.4$.

If the assumption holds, then using both the labelled and unlabelled data gives us a more reliable Decision Boundary than the one initially proposed. As we can see from this example (displayed also in Figure 66), the distribution of the unlabelled data helps us in identifying the regions having the same label, with the smaller amount of labelled instances providing us with the actual labels.

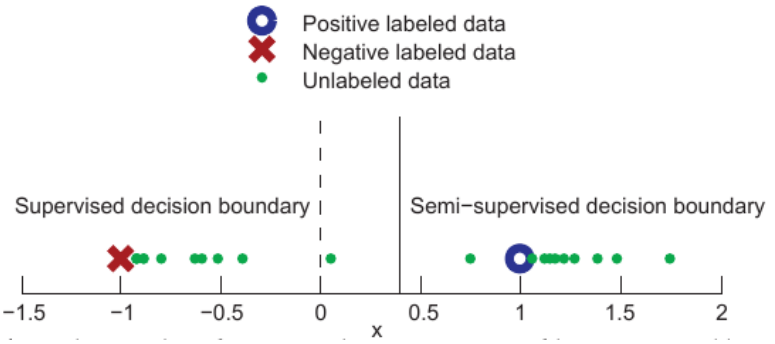



Figure 66: An example of the decision boundaries determined by a Supervised system and a Semi-Supervised system. Source: Zhu & Goldberg (2009)

*Inductive vs. Transductive Semi-Supervised Learning*

In Semi-Supervised Learning, on finds two modes. Since in our dataset we have both labelled and unlabelled data, we have two goals to achieve, namely:

1. To predict the labels for the upcoming data instances in the test set. This is known as **Inductive Semi-Supervised Learning**.
2. To predict the labels of the unlabelled data instances inside of the training set. This is known as **Transductive Semi-Supervised Learning**.

Formally, in Inductive Semi-Supervised Learning (Zhu and Goldberg, 2009), given a training sample;

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{l}, \{\boldsymbol{x}_j\}_{j=l+1}^{l+u} \tag{88}$$

...our model learns a function $f : X \mapsto Y$, such that $f$ is expected to be a good predictor on future data, beyond $\{\boldsymbol{x}_j\}_{j=l+1}^{l+u}$.

In Transductive Learning (Zhu and Goldberg, 2009), given a training sample;

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{l}, \{\boldsymbol{x}_j\}_{j=l+1}^{l+u} \tag{89}$$

...our model learns a function $f : X^{l+u} \mapsto Y^{l+u}$, such that $f$ is expected to be a good predictor on the unlabelled data $\{\boldsymbol{x}_j\}_{j=l+1}^{l+u}$.

### *Limitations of Semi-Supervised Learning*

Semi-Supervised Learning may seem as a revolutionary step on Supervised Learning; providing the same, if not better, performance, while using incomplete datasets with unlabelled data.

In reality however it is not that simple. Blindly opting for Semi-Supervised methods for any specific task can often lead to worse results than a Supervised solution (Zhu and Goldberg, 2009).

This is due to the assumption we make when handling the unlabelled data in our dataset. Since our Semi-Supervised model relies heavily on this assumption, a wrong one can lead to a significant decrease in performance and accuracy. Careful evaluation of the data is a must before committing to which type of learning algorithms to use.

### *Semi-Supervised Learning in Algorithms & Applications*

Semi-Supervised Learning can be found in various practical applications, including i) Image Searching, ii) Genomics, iii)Natural Language Processing and iv) Speech Analysis. It can be integrated inside of well-known algorithms, each of which having its own advantages and disadvantages, and must be used dependently on the application in question.

A simple yet effective algorithm making use of Semi-Supervised Learning is known as the Self-Training Model (McClosky et al., 2006; Zhu and Goldberg, 2009). It is a Wrapper Method, capable of wrapping itself around other algorithms without altering their inner workings. It also comes with a crucial limitation; small errors occurring in the initial training iterations can get reinforced throughout the rest of the training.

To combat this limitation, among other improvements, one also finds other, more complex algorithms. Transductive SVMs (Bennett and Demiriz, 1999) are Support Vector Machines embedded with Semi-Supervised Learning.These methods however have difficulty scaling to large amounts of data. Graph-Based Methods (Goldberg and Zhu, 2006; Zhu and Goldberg, 2009) are some of the most used techniques. Labelled information is spread through the graph from labelled to unlabelled nodes, connecting similar observations. One also finds Neural Network solutions such as Generative Models and Deep Generative Models (Zhu and Goldberg, 2009; Kingma et al., 2014), which are capable of allowing a more robust set of features to be used than Linear Embedding used by other solutions.

# *Structural Risk Minimization*

Structural Risk Minimization (SRM) is a technique first developed in 1974 by Vapnik and Chervonenkis (1974), used to identify the function $f(x)$ that solves a machine learning problem $y = f(x)$. SRM attempts to find a balance between training the model as accurately as possible, while not having an overly complex solution.
Consider the equation below (Guyon et al., 1992):

$$\min_{f} \left[\hat{R}(f) + \epsilon(f)\right] \tag{90}$$

The first part of this equation is the training loss, or what we can refer to as the expected risk $\hat{R}(f)$. The expected risk is found based on the training data, and represents the frequency of errors on our training set. Parameters are adjusted to minimize $\hat{R}(f)$ so as to obtain the best possible model for our problem. This technique is referred to as Empirical Risk Minimization (ERM). The second part of the equation $\epsilon(f)$ refers to the capacity measure of the function. This is used to quantify the complexity of our function $f$.

## *VC-Dimension*

A vital topic for one to be able to understand SRM is the VC-Dimension. The theory about uniform convergence of empirical risk to actual risk describes various conditions, as well as the bounds for the rate of convergence. These bounds are based upon a measure of the capacity of the set of functions implemented by the model, also known as the VC-Dimension (Vapnik, 1992). Before looking at the definition of the VC-Dimension, it is important to understand the following definition:

**Definition 1** *Given a collection $F$ of subsets of a set $S$, we say that the finite subset $A$ of $S$ is shattered provided that every subset $B$ contained in $A$ can be written as intersection of $A$ with an element of $F$.*

The VC-Dimension is defined(Vapnik, 1992) as:

**Definition 2** *The VC-dimension of a set of indicator functions is the maximum number $h$ of vectors that can be shattered in $2^h$ ways by using functions within that set.*

It should be noted that a VC-Dimension of $h$ does not mean that all sets of $h$ points can be shattered by a given set of functions, but there is at least one example of it occurring.

### *Structural Risk Minimization*

Suppose $m$ is the size of our sample, and $h$ is our VC dimension. When $\frac{m}{h}$ is large enough, our VC confidence is negligible and hence, it can be ignored and ERM can take place. However, when $\frac{m}{h}$ is small, the VC confidence can no longer be ignored. When this is the case, we need to be able adjust the VC-dimension $h$ of our model (Vapnik, 1992). This is where Structural Risk Minimization comes in. It is a technique that allows a trade off in accurate model fitting and avoiding overcomplexity. One may think that we would want our VC-dimension $h$ to be a high value, as our risk $R(f)$ drops. However, while this is true, this would increase the chances of overfitting. This is why we have our complexity term $\epsilon$, as it increases when this occurs. On the other hand, if $h$ is too low, then we would have an issue of our $R(f)$ being too high. So the user must have the right balance of accuracy and complexity.
In mathematical terms, SRM is carried out by first selecting a family of classifiers $\{F(x, w)\}$, and then defining a structure of nested subsets of the elements of the family: $S_1 \subset S_2 \subset \ldots \subset S_{n-1} \subset S_n$, where $S_i = \{f(x, w),\ w \in W_i\}$. Due to the structure of the subsets, their respective VC-dimension satisfies: $h_1 < h_2 < \ldots < h_{n-1} < h_n$. Our goal is to find the optimal structure for our problem, denoted $S^*$.

To find this optimal structure, we carry out the following steps, as described by Vapnik (1992) and Sewell (2008):

1. Find the ERM for each element of the structure $\hat{f}_i = \arg\min_{f \in S_i} \hat{R}(f)$
2. Next we calculate $i_* = \arg\min_{i \in \mathbb{N}} R(\hat{f}_i) + \epsilon_i$ and return $f_{i_*}$

A loss function $L(f(x), y)$ is used to represent our risk function $\hat{R}(f)$, of which there are many examples, such as the hinge loss function or the $\epsilon$-insensitive loss function, used in support vector regression. This function calculates the gap between the function $f(x)$ used as our solution, and the output $y$.
The generalization error of our function, $\epsilon(f)$, is a more technical proposition, explained below using the theorems 1 and 2 described by Shawe-Taylor (1997). These theorems are used to provide an upper bound for $\epsilon(f)$ with confidence $1 - \delta$. This is a very important result for us, as while we want our loss function $L(f(x), y)$ to be as small as possible, it is important that $\epsilon(f)$ is not too high.
$\mathrm{Er}_{\mathbf{z}}(s)$ is the number of errors that $s$ makes on $z$. $\mathrm{er}_{\mu}(s)$ represents the expected error, based on probability measure $\mu$, when $x_1, \ldots, x_m$ are drawn independently.

**Theorem 1** *Let $0 < \epsilon < 1$ and $0 < \gamma \leq 1$ Suppose $S$ is a hypothesis space of functions mapping the input space $X$ to $\{0,1\}$, and let $\mu$ be any probability measure on $G = X \times \{0,1\}$. The probability with respect to $\mu^m$ that for some $\mathbf{z} \in G^m$, $\exists s \in S$ s.t. $\mathrm{er}_\mu(s) > \epsilon$ and $\mathrm{Er}_\mathbf{z}(s) \leq m(1-\gamma)\mathrm{er}_\mu(s)$, that is at most:*

$$4\Pi_S(2m)\exp\left(-\frac{\gamma^2\epsilon m}{4}\right) \tag{91}$$

Theorem 1 is used to prove theorem 2, shown below. $p_d$ represents the probability that $S_d$ is the smallest class containing a consistent hypothesis. On the other hand, $q_{dk}$ represents the probability of errors in the training set. It influences the trade off between the selected function's complexity and accuracy.

**Theorem 2** *Let $S_1, S_2, \ldots$ be a sequence of hypothesis classes mapping $X$ to $(0,1)$ and having VC-dimension $i$. Let $\mu$ be any probability measure on $S = X \times (0,1)$, and let $p_d$, $q_dk$ be any sets of positive numbers satisfying*

$$\sum_{d=1}^{\infty} p_d = 1 \tag{92}$$

*and $\sum_{k=0}^{m} q_{dk} = 1$ for all $d$. Then with probability $1-\delta$ over $m$ independent identically distributed examples $x$, if the learner finds a hypothesis $f$ in $S_d$ with $Er_x(f) = k$, then the generalization error of $f$ is bounded from above by:*

$$\epsilon(m,d,k,\delta) = \frac{1}{m}\left(2k + 4\ln\left(\frac{4}{p_d q_d k \delta}\right) + 4d\ln\left(\frac{2em}{d}\right)\right) \tag{93}$$

*provided $d \leq m$.*

The function $\epsilon(m,d,k,\delta)$ provides an upper bound on the generalization error of $f$ with confidence $1-\delta$. The above theorem also takes into consideration the possibility of errors occurring within the training set.

An algorithm, such as the support vector machine (SVM), is used to find $d$ which contains a hypothesis $s \in S_d$ that is consistent with our dataset $\{\mathbf{x}, y\}$. The above theorems result in the inequality shown in equation 94 below.

$$R(f) \leq \hat{R}(f) + \epsilon(m,d,k,\delta) \tag{94}$$

## *Examples*

Consider a simple machine learning problem, where we need to find the ideal function to represent our model. As mentioned above, to carry out SRM we have two parts to our procedure, the first of which is finding the function $f_i$ that returns to us the minimum risk $R(f)$ for each structure $S_i$. The definition of our risk function depends on the type of problem we are solving. For a classification problem, one may want to use a zero-one loss function or a hinge-loss function. On the other hand, for a regression problem, a Huber loss function may

be favourable. Once we have selected a suitable loss function, our next step is to look at functions to represent our capacity measure. The $l_1$-regularizer and $l_2$-regularizer are two examples of functions that can be used to calculate the capacity measure.
Below is the formula that needs to be solved for a support vector machine, according to SRM:

$$\min_f \left[ C \cdot \sum_{i=1}^{n} \max\left[ |y_i - f(x_i)|, 0 \right] + \|\mathbf{w}\|_2^2 \right] \tag{95}$$

with the solution $\hat{f}(x_i) = C_k \cdot K(x, x_k)$, where $K$ represents the kernel function. Notice how SVMs use the hinge-loss function as their risk function and the $l_2$-regularizer to quantify complexity.
If we are to carry this out for linear regression, our formula would look something like this:

$$\min_f \left[ \sum_{i=1}^{n} (f(x_i) - y_i)^2 + \|\mathbf{w}\|_2^2 \right] \tag{96}$$

In reality, we can use whichever regularizer we prefer, however, the $l_2$-regularizer has an advantage over the $l_1$-regularizer as it is strictly convex and differentiable everywhere.

Figure 67 is a diagram illustrating the general idea of SRM:

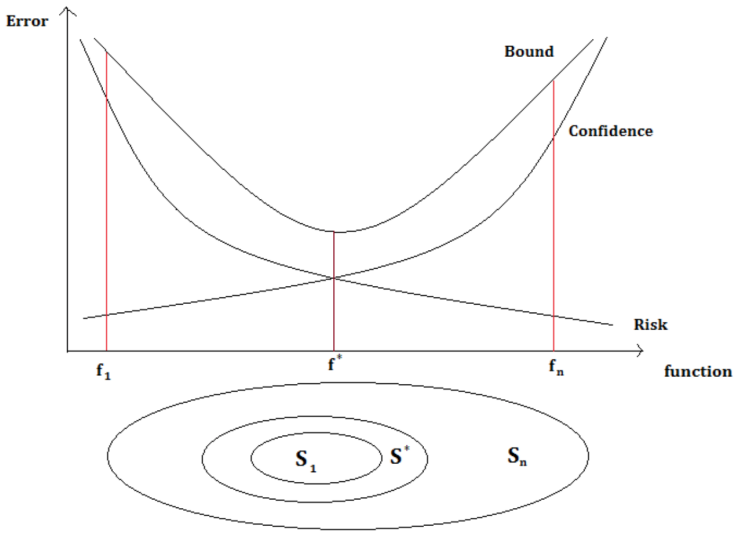



Figure 67: An illustration of the empirical risk and confidence for different functions. The optimal function is found at the minimum of the bound, listed as $f^*$

# *Synthetic Features*

## *The problem*

Training a machine learning algorithm requires inputting some form of training data. This training data comprises of all the features from which the algorithm learns from and builds a model. This input is often referred to as the training dataset.

Whilst in concept the above seems straightforward, it often transpires that the various data-points provided in the training dataset do not fit a structure that is easily understood by the algorithm. For this reason, an important pre-processing step is needed to:

1. Understand the original data well
2. Subsequently, if and where needed, generate synthetic features

If we look at the following example (Alberto et al., 2015): It contains a number of records used to train a spam / ham classifier for comments on a YouTube video.

| *Video* | *Comment ID* | *Author* | *Date* | *Content* | *Class* |
|---|---|---|---|---|---|
| *Psy* | *LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8* | *Evgeny Murashkin* | *2013-11-08T17:34:21* | *just for test I have to say murdev.com* | *Spam* |
| *Psy* | *z13bgdvyluihfo11i22rgxxvhuvabzz1os04* | *Zielimeek21* | *2013-11-28T21:49:00* | *I'm only checking the views* | *Ham* |
| *Psy* | *z13kzpqqssa0hlryd04cc1dxeyymgljjngk* | *Tasha Lucius* | *2014-01-19T13:25:56* | *2 billion....Coming soon* | *Ham* |
| *Psy* | *z12lg1vizrmsgxm3q23oij4aqrjxjdd1p* | *Holly* | *2014-11-06T13:41:30* | *Follow me on Twitter @mscalifornia95* | *Spam* |

Table 9: Sample of four rows from the Psy dataset from the YouTube comment training dataset.

Table 10 describes each feature in the original unmodified dataset.

As one can see, there is very little input the machine learning algorithm can reliably take just from using the four features described above. One could easily realise this by asking oneself the following question (in plain English):

*How can I describe the components of a comment well enough to decide whether it is probably spam or ham?*

One can therefore summarise this problem paradoxically as: *Having enough data to solve the problem, but very little meta-data to actually understand it and solve it.*

## *Ways of solving the problem*

A way of solving this problem is to apply a synthetic features approach, sometimes referred to as feature engineering. This is the generation of features derived from other existing features, in a way

| *Feature* | *Description* |
|---|---|
| *Video* | *The video this comment was written for. The relevance depends whether the classification model is being built generically for all videos, or a per-video specific model is also considered.* |
| *Comment ID* | *Random comment ID generated by the YouTube comment board system. This probably has no impact on the final class.* |
| *Author* | *The author / account that generated the comment. This has relevance only if this account has a lot of spam comments.*<br>*If that is the case, two things should happen, none of which are directly related to the machine learning algorithm:*<br>*- Maintain a blacklist of accounts that are probable spam (if a particular author often has flagged comments).*<br>*- Block such accounts.* |
| *Date* | *The date does not directly have a huge relevance on the classification of a comment.* |
| *Content* | *The comment body definitely has a big relevance in the classification result, however, can the whole sentence be easily understood by the algorithm as it is?* |

Table 10: Description of each feature in the original unmodified YouTube comment dataset.

that can be more easily captured or understood by a machine learning algorithm (Li et al., 2013). It is a way of generating meta-data for the existing features in the original dataset.

In essence, the idea is to look at every available feature and for each determine the following:

- Does the feature contain more than one feature within it? If so, try exploding it into sub-features and test.
- Does the feature contain too little information for any relevance, but could benefit from adding some context to it? If so, attempt at looking at other features that might be related, and produce new features as a result, and test.
- For each of the above, the original feature(s) might not be relevant anymore and be entirely replaced by the newly generated synthetic features instead.

What or how an explosion of features or a composite of features is generated depends on the very specific nature of the components involved and there is no generic formula behind it that works without some additional specificness. For example, two pairs of geo-coordinates probably qualify in giving a distance feature, however the formula applied here is specific to the geo-coordinates domain.

There is not a one-size-fits-all approach but rather it is more of an iterative approach with new synthetic features being outputted per iteration, following which one then assesses whether it is enough to generate a reliable machine learning model from the new features or not.

Following below is a practical example of this technique, using the dataset described at the introduction of this chapter.

### *Analysing each feature*

- Video

  - The video name / ID could be useful if a per-video classifier is also generated over and above the generic one. This together with other features could have some relevance.

- Comment ID
  - This feature does not have any relevance to the outcome whatsoever. It is a unique ID, built randomly, assigned to each comment. For this reason, it is out of scope for this discussion.

- Author and Date
  - As described earlier these two features independently do not have much of a direct impact on the outcome, however a synthetic feature could be generated which might have some form of effect on the outcome: A ratio of comment count over a time period for a particular author. The idea is to make it easier for the algorithm to detect a potential pattern related to volume over a typical short period, thus the definition of time period can be assigned via testing.

- Comment
  - The comment body is not an easy feature and it could grow into a number of features, however it is the most relevant input for this spam classifier. Quite a number of features could be exported from this comment, and most of them relate to natural language processing techniques (Cormack et al., 2007). For this reason, output quality could also vary based on the language in context. Some example features that could be extrapolated:

| *Synthetic Feature* | *Scope / Description* |
|---|---|
| *Language* | *This depends on the availabilities of various NLP implementations for different languages, however one could have an indication of spam / non-spam probabilities based on the comment languages for each particular video.* |
| *Readability score* | *A readability score could be calculated per comment which gives an indication on the quality of such text. An example of such a score could be the* Flesch Reading Ease *score.* |
| *Length (excl. stop words)* | *Very short or very long comments might have a probabilistic impact on the outcome.* |
| *Presence of account tags / URLs / emojis* | *The presence of account tags (ex. a Twitter username), URLs or emojis could increase probability of the comment being spam.* |

Table 11: Example of possible features that can be extracted from textual comments.

### *Updated feature / data set*

Following the synthetic feature generation described above, the updated data set used as an example here would look as follows:

Looking at the output in table 12, the effect of synthetic features can immediately be appreciated, as with such new features more meaning is given to the original dataset.

| *Video* | *Author Comments in last minute* | *Language* | *Readability* | *Length excl. stop words* | *Presence of account tags* | *Presence of URLs* | *Presence of emojis* | *Class* |
|---|---|---|---|---|---|---|---|---|
| *Psy* | *1* | *EN* | *94.3* | *3* | *No* | *Yes* | *No* | *Spam* |
| *Psy* | *1* | *EN* | *103* | *3* | *No* | *No* | *No* | *Ham* |
| *Psy* | *1* | *EN* | *83.3* | *3* | *No* | *No* | *No* | *Ham* |
| *Psy* | *1* | *EN* | *32.6* | *3* | *Yes* | *No* | *No* | *Spam* |

Table 12: Updated sample of the four rows from the Psy dataset from the YouTube comment training dataset now containg the synthetic features.

Naturally the above contains just a sample, and one must experiment with:

- more or less synthetic features
- a further iteration of synthetic features from the generated features
- a much bigger data-set (the example above is too small to build a reliable classifier)
- perform feature selection (such as Principle Component Analysis) to identify the features that actually matter and remove extra noise

Therefore, employing a synthetic feature approach on your dataset as a pre-processing step, should in general give you positive results.

# *Tensor Processing Unit*

A Tensor Processing Unit or TPU, is a proprietary Application-Specific Integrated Circuit (ASIC) used as an Artificial Intelligence accelerator (Jouppi et al., 2017). It is designed and built by Google for use in their datacentre to power their products, including; Google Search, Google Translate, Google Photos and Gmail. Through Google's proprietary Accelerated Linear Algebra (XLA) compiler, binaries to run on TPUs are compiled from an XLA graph format produced by the TensorFlow libraries (Google, 2017).

Note that for the scope of this documentation, the architecture of the first generation TPUs is going to be explained, since it's the most well documented one. Further updates in the architecture are fairly similar to what will be discussed here and follow the same concepts, with the main difference being that newer generations are also used for training, instead of just inference. Therefore they support a truncated variant of the IEEE 754 single-precision floating-point format called bfloat16 instead signed 8-bit integers together with various other training optimizations.

## *Background*

The motivation for Google to design and build a custom architecture on an ASIC, is due the shortcomings that come with Central Processing Units (CPUs) and Graphics Processing Units (GPUs) when used to run Artificial Neural Network models. This is because 95% of Google's applications in their datacentre run Neural Networks for inference, specifically; Convolutional Neural Networks (CNNs), Multi-layer Perceptron Neural Networks (MLPs) and Long short-term memory Neural Networks (LSTMs) (Jouppi et al., 2017).

CPUs are sequential by nature, where each instruction is executed one at a time, which is really undesirable, and would require high clock speeds to get very high performance but with the expense of higher power consumption. In contrast GPUs are concurrent by nature, meaning that they can do several concurrent operations at once, therefore eliminating the need to have very high clock speeds, but it is only well suited for vector processing, hence their main use is Graphics Processing. In Machine Learning and specifically in Neural Networks, matrix operations are preferred as they speed up the computation by quite a big margin(Sato et al., 2017).

Additionally, modern CPU and GPU architectures have perfor-

mance issues due to their complex and sometimes non-deterministic ways of executing programs. This includes complex optimizations, like branch-prediction, out-of-order execution and multiprocessing (Paun et al., 2013). These algorithms are more tailored for throughput efficiency rather than having fixed latency (Jouppi et al., 2017). Additionally, complex algorithms implemented in modern CPUs and GPUs, together with their high clock speeds, contribute to both a large area and higher power consumption. Google therefore designed their own custom AI accelerator chip around the necessary building blocks to run their Neural Network applications on, with the maximum possible efficiency.

In order to understand why Google TPUs are designed as they are, an insight into the mathematics of involved in Neural Networks is needed. The only computations done in Neural Networks involve only multiplications and additions, when excluding the activation function. While CNNs use convolution instead of multiplication, this report will explain the multiplication operations. This is because the input of a neuron is the sum of weighted (synapse weight) outputs of all the neurons in the previous layer, as represented in Equation 97.

$$Z = \sum_{i=1}^{n} x_i w_i \tag{97}$$

Where $i$ is the number of neurons in the previous layer connected to the current neuron, $x$ is the output of each individual neuron in the previous layer, $w$ is the synapse weight present in the connection between the previous neurons to the current one.

## *Architecture*

Therefore as described in the previous section, a Multiply-Accumulate (MAC) block is the only computational block that is needed to perform the needed operations. Figure 68 illustrates a typical MAC block, where binary values A and B of length $[n:0]$ are multiplied, therefore producing a binary value C of length $[2n+1:0]$. C is then added to the accumulated binary value Y, and the Accumulator register is updated with the new accumulated binary value. The Accumulator Register's value is present on binary value Y.

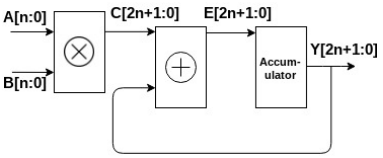



Figure 68: Multiply Accumulate Logic Block (Adapted from (Paquin, 2018)).

The MAC block alone does not contribute to good performance when a large data-set and complex Neural Networks are present, therefore TPUs implement it in a so called Matrix Multiplication Unit (MXU), which is basically a 2-dimensional array of MACs that operate in systolic form (Paquin, 2018). The best intuitive way to understand how an MXU works is to visualize matrix multiplication as done in a layer of a Neural Network prior to activation.

Consider the simple example shown in Figure 69, which shows a Neural Network layer with 3 inputs and 2 neurons. Prior to applying the activation function $f(x)$, the computation done is the same as described in Equation 97, for each neuron, y1 and y2.

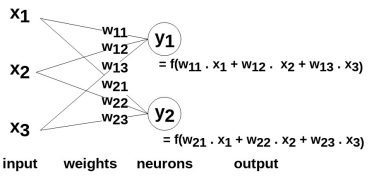



Figure 69: Example of a neural network layer with 2 neurons and 3 inputs (Reproduced from (Sato et al., 2017)).

For this example an MXU needs to contain at least 6 MACs ar-

ranged in array of 2×3, where 2 denotes the number of neurons and 3 denoting the number of inputs. Figure 70 shows how such computation is done inside the MXU with just 1 example/instance X being inputted for inference, and since we are computing just 1 layer, the weights $w$ only need to be loaded once.

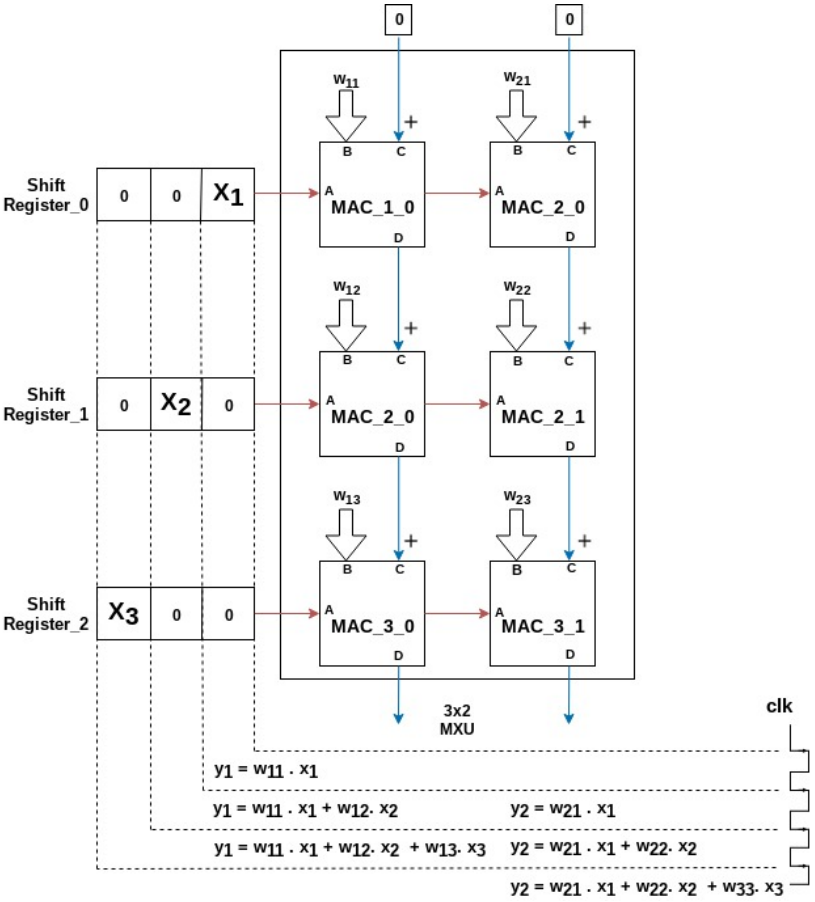



Figure 70: Operation of a Simplified Matrix Multiplication block based on a 2 neuron layer example having 3 inputs (Adapted from (Sato et al., 2017)).

Assuming input data X1, X2 and X3 are already loaded into their corresponding shift register and the weights $w$ are already present on one of their corresponding MAC inputs, the data flow with each clock cycle denoted by clk in Figure 70, which follows a systolic fashion.

Prior to the computation, inputs A of all the MACs are reset to 0, while inputs B will have already been loaded with the weight values of the current layer. With each clock cycle the shift registers shift the data to the right into the MXU, where the A and B inputs are multiplied together and added with the value present at input C. In this manner the values of outputs D of the bottom most MACs will output the total sum of products on output D, with the bottom right most MAC being the last one to be updated after $mn$-$m$ clock cycles, where m is the number of inputs and n is the number of neurons. In this example the total amount of clock cycles needed is $2 \times 3 - 2 = 4$. In a Google's first generation TPU, there are 65,536 signed 8-bit MAC blocks in a 256×256 array fashion that make up the MXU. The MXU in a TPU can compute both matrix multiplication

and convolution (Sato et al., 2017). After finally computing the sum of products, activation functions are applied on each neuron's result.

Figure 71 shows a high level representation of how all the components of the first generation TPU are connected internally, including the MXU. TPUs where designed to act as coprocessor with a host CPU, where they communicate together via a 3rd generation PCIe x16 bus. First generation TPUs were only designed for inference only, therefore the TPUs represent data only in a signed 8-bit integer, which has been proved that it is of sufficient precision for inference purposes (Jouppi et al., 2017).

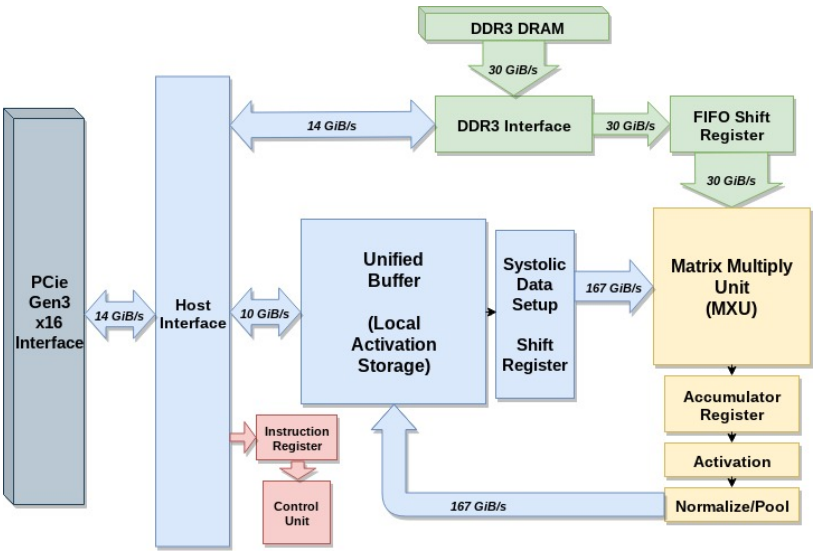



Figure 71: Internal Layout of a first generation TPU, showing the Control path (red), the Data paths for the weights (green) and for the activations/outputs (blue), together with the computational logic blocks (yellow) (Adapted from (Sato et al., 2017)).

As with any architecture, there is a Control path and a Data path. The Control path, which is denoted in red, serves the purpose of giving instructions to the TPU via the host CPU, like loading the data in registers, do multiplication and more. As for the Data path, two types of data are present. One being the weights, which are stored prior of inference and are denoted in green, while the other is the neurons activations/outputs, where its path is denoted in blue. As the weights are loaded only once before computing each layer, they do not require a high speed memory, so they are stored in a DDR3 DRAM, where later they are loaded into the MXU via a FIFO shift register. On the other hand, the neuron's activations/outputs, are stored by the host CPU in a high speed Unified Buffer through the PCI Express Interface. The Unified Buffer is where the activations of each neuron that are computed by the computational logic block are read and written, therefore it requires a high speed memory and interconnect bus. The Normalize/Pool block, processes the data to reduce its size or downsample it. As one can see the architecture is quite simple compared to other general purpose architectures like CPUs and GPUs. This allows Google to use a Complex Instruction Set Computer (CISC) instruction set with just a few instructions, since not many different operations are done (Sato et al., 2017).

# *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).

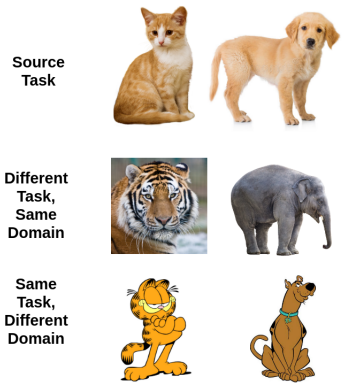



Figure 72: Transfer learning can occur across domains or across tasks. Images reproduced from commons.wikimedia.org (Tiger, Cartoon Cat), imdb.com (Cartoon Dog), pixabay.com (Cat), pnghunter.com (Elephant) and shutterstock.com (Dog).

## *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.

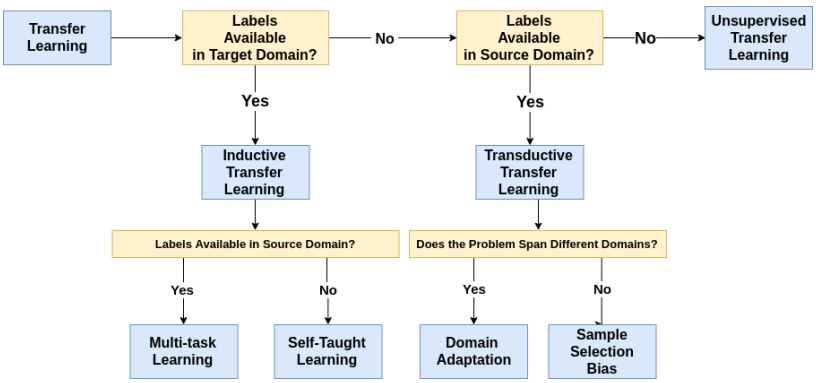



Figure 73: Transfer learning approaches vary depending on the availability of labeled data for the source and target tasks.

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 13: 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).

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).

### *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.

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).

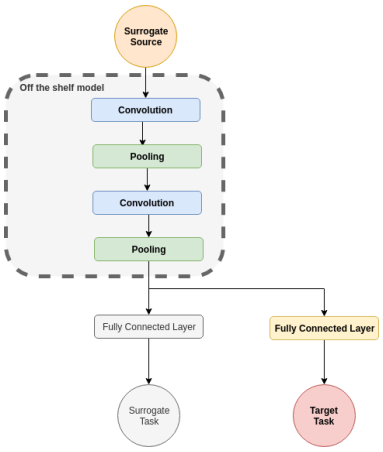



Figure 74: In off-the-shelf pre-trained models, we replace the final layer of the neural network.

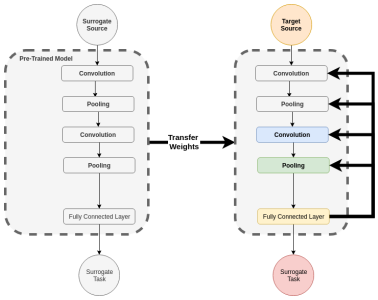



Figure 75: In Pre-trained model fine tuning, we use the source model to initialize the neural network, and fine-tune it using backpropogation.

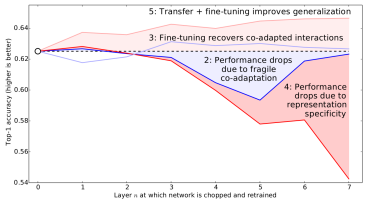



Figure 76: The performance gained form fine-tuning a neural network is only relative to the layer generalization and specificity of each distinct layer (Yosinski et al., 2014).

# *Receiver Operating Characteristic Curves*

## *Introduction*

A Receiver Operating Characteristic (ROC) curve is a graphical plot that presents the performance of a binary classifier when the discrimination cut-off, or threshold, is varied. ROC curves and the associated analysis area under the curve and finding optimal threshold, have proved to be very useful analysis tools in many fields of research. Numerous studies evidence that they have been found to be very effective techniques for evaluating diagnostic and predictive accuracy.

Essentially, for creating and plotting the ROC curve one needs to compute the true positive rate (TPR) and the false positive rate (FPR) at various threshold settings and plot them in a two-dimensional space. In machine learning the TPR is referred to as the sensitivity or recall which measures the proportion of actual positives that are correctly classified. The true negative rate (TNR), or specificity, measures the proportion of actual negatives that are correctly identified as such. These measures can be easily summarised in Table 14. ROC is an evaluation procedure based on the sensitivity-specificity pair of indices.

| | **True Class** | | | |
|---|---|---|---|---|
| **Predicted Class** | Positive | Negative | | |
| Yes / 1 / Positive | True Positives | False Positives | TPR=TP/P | *Sensitivity* |
| No / 0 / Negative | False Negatives | True Negatives | FPR=FP/N | *1-Specificity* |
| **Totals** | **P** | **N** | | |

Table 14: Sensitivity and specificity

Consider a diagnostic test that seeks to determine whether a person has a certain disease or not. The two-class predictor yields an outcome that is labelled either positive or negative. From table 1 there are four possible outcomes from the discrete classifier: true positive or true negative if the outcome matches the actual class, false positive if the person is incorrectly classified as sick, and false negative if the person is incorrectly classified as healthy. If the test is extended to a larger sample then given a classifier and a set of instances, a 2x2 contingency table, or confusion matrix, can be constructed for the classifier with values for the four possible outcomes. Sensitivity and specificity are computed accordingly as shown in Table 14.

If one extends the test further to six classifiers, where the same metrics are extracted from the outcomes of these classifiers, one ends up with six sensitivity-specificity pairs. The latter are represented as points on an ROC plot as illustrated in figure 77 (adopted from Fawcett (2006)). A to F are sensitivity-specificity pairs representing the six classifiers used in the hypothetical test. TPR (sensitivity) is plotted on the y-axis and FPR (or 1 âĂŞ specificity) on x-axis. This plot depicts the relative trade-offs between true positives and false positives. The higher the sensitivity, the more beneficial and trustworthy the classifier is, so TPR is a measure of the classifier's benefits. A low specificity value represents a costly classifier, so FPR is synonymous with a classifier's cost. The best predictor is the one that maximises the benefits and minimises the costs.

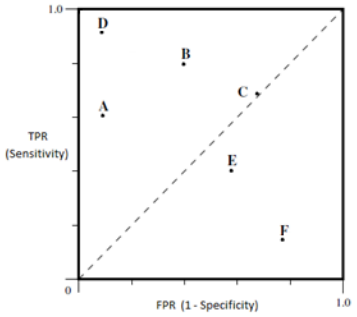



Figure 77: Basic ROC plot with five classifiers.

## *Evaluating Classifiers*

Classifier C lies on the diagonal line from (0,0) to (1,1) of the ROC plot. It represents a test that cannot distinguish between diseased and non-diseased states. Its TPR equals FPR. This means that classifier C is merely guessing the labels randomly. It is as accurate as predicting heads or tails in a coin toss. The diagonal line often serves as a benchmark for judging how good (or bad) a test performs. E and F lie on the right-hand side of the diagonal line with relatively low sensitivity and specificity metrics compared to the others. In general, points on this side of the ROC plot are considered worthless (or costly) classifiers.

Classifiers A, B and D are considered better predictors from the group since they lie on the left-hand side of the diagonal line with high TPR and low FPR. D is clearly the best predictor in this case because it is the closest to the top left-hand corner of the graph. The perfect predictor would yield 100% sensitivity and 100% specificity with zero false positives and zero false negatives, that is, zero cost. However this is realistically very difficult to achieve. In this way the ROC plot provides an easy comparison of classifiers and leads to choosing the one that is closest to (0,1) and furthest from the diagonal line.

Consider now a continuous random variable $X$, denoting the width in millimetres of metal pipes from an automated production line, where $X$ is normally distributed. An inspection test might measure the diameter of the pipes and classify a pipe as defective if the width is above a certain value. Thus given a threshold width $w$, the instance is labelled defective if the continuous random variable $X$ is greater than $w$ and non-defective otherwise. As shown in figure 78, one can idealise curves for both the number of positive and the number of negative results of the test. Since the test does not distinguish good pipes from defective pipes with 100% accuracy, the distributions overlap. The overlap area indicates where the test cannot distinguish between the two classes. In practice, one chooses a cut-off (or threshold) above which the test is considered normal. Different cut-off points are used

in different situations in order to minimise one of the erroneous types (false positives or false negatives) of the test result. The sensitivity and specificity of a classifier system depends on the particular confidence threshold that the system is using and changes inversely as the confidence threshold is changed. If the confidence threshold is made less strict, then sensitivity will increase due to more instances being classified as true positives. However, specificity will decrease because more false positives will appear. TPR and FPR increase or decrease simultaneously as the confidence interval threshold is changed. By repeating the experiment with adjusted thresholds, different value pairs are obtained leading to different positioning in the ROC space. ROC curves allow the visualisation of the trade-off between sensitivity and specificity for all possible thresholds rather than just the one that was chosen by the modelling technique.

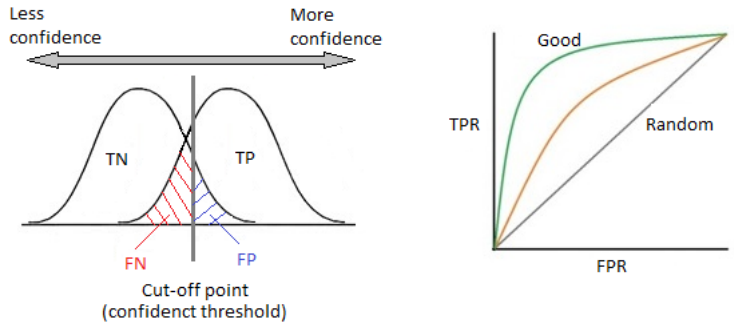



Figure 78: Left: Overlapping distributions with threshold. Right: ROC curves resulting from different thresholds.

The classifier that is able to increase the rate of detection while keeping the false alarm rate low is considered a good classifier. Figure 78 shows the ROC curves with different thresholds. As the ROC curve gets closer to the top left hand of the graph the more optimal the threshold and hence the better is the classifier's ability to discriminate between the classes.

## *Area Under Curve*

The most common condition for choosing the optimal parameterisation is to maximise the area under the curve, AUC for short. The AUC is a widely used measure of performance of supervised classification rules. It is a measure of the usefulness, or rather the discriminatory ability, of a test in general where the greater the area the more discriminative the test is in a given situation. A model or test with perfect discriminatory ability will have an AUC of, or very close to, 1.0, while a model unable to distinguish between classes will have an AUC value of 0.5 or less.

A precise mathematical computation of the AUC is $\int_0^1 ROC(t)dt$, however since the ROC curve is normally plotted by joining small points close to each other, it can be very closely approximated by adding the area of many small rectangles next to each other covering

the whole section under the curve as shown in figure 79.

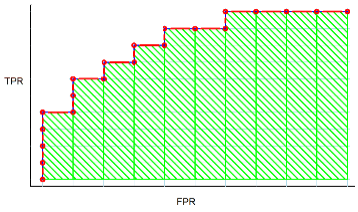



Figure 79: Area under the curve.

However, the true value of AUC comes when comparing two or more models or tests by reducing the ROC to a single scalar value representing the expected performance of the models. Figure 80 (adopted from Linden (2006)) provides a comparison between two predictive models. Clearly Model A yields a better predictive ability than Model B since its AUC is bigger. Therefore, if a choice was to be made between selecting one of the models, Model A would be the choice.

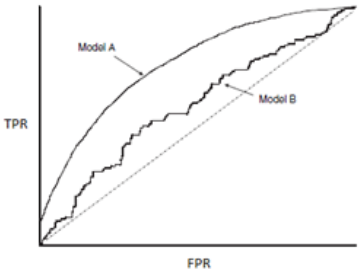



Figure 80: A comparison of two AUC curves.

AUC is a desirable measure because (1) it is scale-invariant: it measures how well predictions are ranked rather than their absolute values; (2) it is classification-threshold-invariant: it measures the quality of the modelâĂŹs predictions irrespective of the classification threshold chosen.

The majority of studies on binary classifiers in conjunction with imbalanced datasets still use the ROC plot as their main performance evaluation method and this make it easier to compare results. However a recent study (Saito and Rehmsmeier, 2015a) shows that Precision-Recall plots provide a more accurate prediction of future classification performance although they are less frequently used. It is therefore appropriate to use the AUC metric together with other measures of evaluation depending on the nature of the dataset.

## *Partial AUC*

AUC is closely related to another widespread statistic, the Wilcoxon statistic (Hanley and McNeil, 1982). Thanks to this relationship, AUC methods for AUC-based analyses are well developed and widely used. However one of the major practical drawbacks of the AUC as an index of diagnostic performance is that it summarises the entire ROC curve including the regions that are frequently not relevant to practical applications, such as regions with low level of specificity (Ma et al., 2013). To alleviate this deficiency while benefiting from some of the advantageous properties of the area under the ROC curve, one can use a partial area under the curve (pAUC for short), which summarizes a portion of the curve over the pre-specified range of interest. A number of approaches have been developed for pAUC-based analysis (Dodd and Pepe, 2003; He and Escobar, 2008). However, Ma et al. (2013) argue that the same features that increase the practical relevance of the pAUC introduce some difficulty to resolve issues related to arbitrariness of specifying the range of interest.

In the case of population screening, for instance, Dodd and Pepe (2003) claim that in order to avoid high monetary costs, the region of the curve corresponding to low false-positive rates is of primary interest whilst in diagnostic testing it is critical to maintain a high TPR in order not to miss detecting diseased subjects. They go on to consider a summary index for the ROC curve restricted to a clinically relevant range of false-positive rates.

Essentially the partial AUC is simply the area under the ROC curve

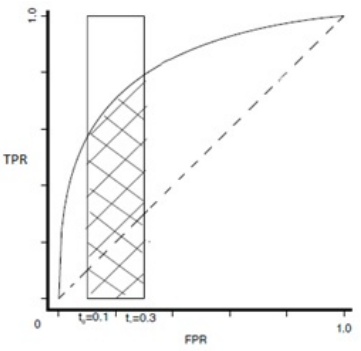



Figure 81: Illustration of an ROC curve and its partial AUC with $t_0$ = 0.1 and $t_1$ = 0.3.

between $t_0$ and $t_1$ (Figure 81) where the interval $(t_0, t_1)$ denotes the false-positive rates of interest. The partial AUC is thus the integral between $t_0$ and $t_1$ of the ROC curve function as shown in equation 98 (Dodd and Pepe, 2003).

$$AUC(t_0, t_1) = \int_{t_0}^{t_1} ROC(t)dt \tag{98}$$

### *Two Specific Applications of ROC*

The use of ROC curves and Area Under ROC curves is ubiquitous and the list of scientific research papers citing the use of AUC as an essential and effective evaluation technique is endless. They have been used by numerous authors to evaluate models, scoring systems, diagnostic tests, manufacturing defects, spatial images processing, psychiatric evaluations, behaviour analysis, and so on.

An interesting recent paper by Fung et al. (2018) evidences how ROC curve statistics are used to evaluate Convolutional Neural Network for breast cancer classification. It compares the ROC curves generated by the classifier with other reviewed studies. The analysis of AUC helped to evaluate the superiority of the neural network as compared to methods based on experimental results conducted on real patient subjects. Another recent paper (Ding et al., 2017) analyses a deep learning method that accurately recognises an identity from heavily noisy face images. The paper discusses ROC curves with optimal thresholds to show that noise-resistant Deep Neural Network is visibly superior to some state-of-the-art feature extraction algorithms.

Another field where ROC proves to be contributory is psychology. Correct identification of criminals by eyewitness can help to remove a dangerous criminal from society but a false identification can lead to the erroneous conviction of an innocent suspect. A paper by Gronlund et al. (2014) describes how by constructing ROC curves, researchers can trace out discriminability across levels of response bias for each lineup procedure. They illustrate the shortcomings of ratio-based measures and demonstrate why ROC analysis is required for evaluating the performance of a lineup procedure. Luby (2017) goes a step further and examines the application of ROC methodology to lineup data. He shows that by using a log-linear analysis in conjunction with an ROC approach to eyewitness identification, it is possible not only to visualise the trade-off between true positives and false positives, but also to identify which variables are interacting with one another and explain the trade-off.

### *History*

A final note on the history of ROC. The first use of ROC curves is traced back to World War II where in conjunction with signal detection theory, it was developed for the analysis of radar images. Radar operators had to decide whether a blip on the radar screen

represented an enemy target, a friendly ship or just noise. ROC curves helped signal detection theory to measure the ability of radar receiver operators to make these important distinctions. Their ability to do so was called the Receiver Operating Characteristics.

Decades later in the 1970s, ROC analysis was introduced into the biomedical field via the radiological sciences. It has been used extensively to test the ability of an observer to discriminate between healthy and diseased subjects, using a given diagnostic test. As well as to compare the efficacy among the various tests available at that time. Since then, ROC analysis has been extended for use in (but not limited to) visualising and analysing the behaviour of a broad range of diagnostic systems across many fields of science.

# *Standardization and Normalisation*

Data *standardization* and *normalisation* are crucial data preprocessing stages in order to prepare the data for machine learning or data mining purposes[10]. Standardization is related to the act of centre-reducing the data and a common measure used is the *standard deviation*. This is used in machine learning algorithms such as SVM or regression problems. On the other hand, normalisation is a process concerned with scaling or mapping the data in order to make it easier to work with (Patro and Sahu, 2015). Normalisation is useful in almost every machine learning algorithm (such as regression, kNN, neural networks) since it essentially cleans the data. In this section we shall see a few standardization and normalisation techniques that are related to machine learning.

[10] http://www.dataminingblog.com/standardization-vs-normalization/

## *Min-Max Normalisation*

Min-Max is a linear normalisation technique and hence it does not alter the distribution of a data set. Its goal is to transform the features of a data set to fit within a smaller and more specific range. Normally this new range is set to $[0, 1]$ (see Equation 99) or $[-1, 1]$. This technique is sometimes called *feature scaling*.

$$x'_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{99}$$

In equation 99 above, $x'_i$ is the normalised value, $x_i$ is the unnormalised value and $x_{max}$ and $x_{min}$ are the maximum and minimum values found in dataset $X$ respectively.

Equation 100 is a generic formula for this technique, where the range is specified at two arbitrary points $a$ and $b$:

$$x'_i = (x_{max} - x_{min}) \times \frac{x_i - x_{min}}{x_{max} - x_{min}} + x_{min} \tag{100}$$

When compared to other normalisation techniques, the Min-Max is very effective whilst being a simple technique to implement (Al Shalabi et al., 2006). However, Min-Max is prone*noise* and therefore produce unstable results as it is highly sensitive to *outliers* (Jain et al., 2005).

### *Z-score/Zero-Mean Normalisation*

When applying *z-score* to a data set is the same as standardizing it. Like Min-Max, this is a commonly used technique and it is easy to implement (Jain et al., 2005). However, the Z-score or Zero-Mean normalisation affects the distribution of a data set such that it becomes *normally distributed*. Figure 82 [11] shows a plot of a normal distribution, also known as the *Gaussian distribution*, compared with the original data (before transformation).

[11] https://www.kaggle.com/rtatman/data-cleaning-challenge-scale-and-normalize-data

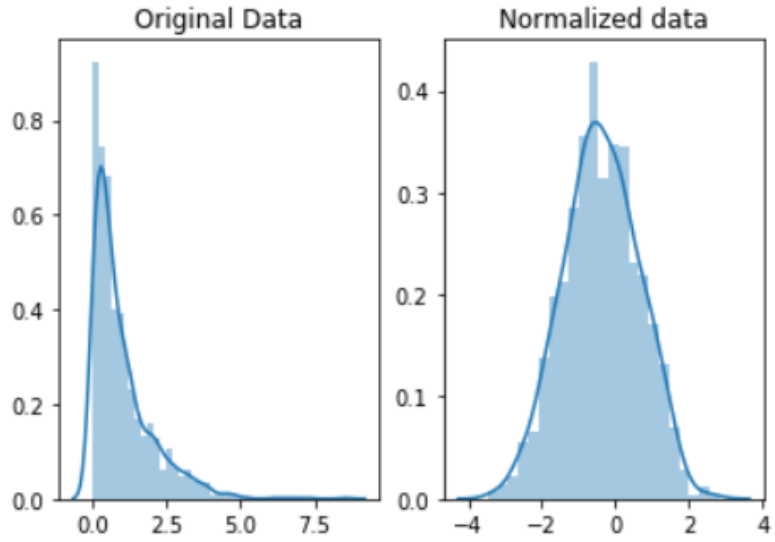



Figure 82: Normal Distribution example - notice the "bell curve" of the normalised data

Equation 101 below shows how z-score is computed by normalising value $v_i$ of attribute $A$ to $v_i'$:

$$v_i' = \frac{v_i - \bar{A}}{\sigma_A} \tag{101}$$

where $\bar{A}$ is the arithmetic mean and $\sigma_A$ is the standard deviation of attribute $A$. When the minimum and maximum of attribute $A$ are unknown, this method is very useful and therefore preferred over Min-Max (Han et al., 2011). This method is also highly applicable and effective for clustering algorithms such as *K-means clustering* (Mohamad and Usman, 2013) (Cheadle et al., 2003).

### *Median Normalisation*

The Median normalisation technique works by calculating the *median* of a sample from a data set, then each element in the sample is divided by the median (see equation 102). This procedure repeated for every sample $s$ in a data set (Jayalakshmi and Santhakumaran, 2011).

$$x_i' = \frac{x_i}{median(s)} \tag{102}$$

Equation 103 is another way how Median normalisation is formulated, by subtracting the median from every element in sample $s$:

$$x'_i = x_i - median(s) \tag{103}$$

For 102 and 103 $x_i$ is the unnormalised element in sample $s$. Note that this technique known as an intra-slide normalisation because it works with whole arrays/samples of data. Although this technique does not guarantee high accuracy, it is quite robust since it is much less sensitive to noise than Min-Max or Z-score (Jain et al., 2005).

### *Sigmoid Normalisation*

The *Sigmoid function* can be used as a normalisation technique to scale values, usually between -1 and 1. There are various forms of non-linear Sigmoid functions. One example is the *tan* Sigmoid function (refer to figure 83)which tends to speed up the normalisation process (Jayalakshmi and Santhakumaran, 2011).

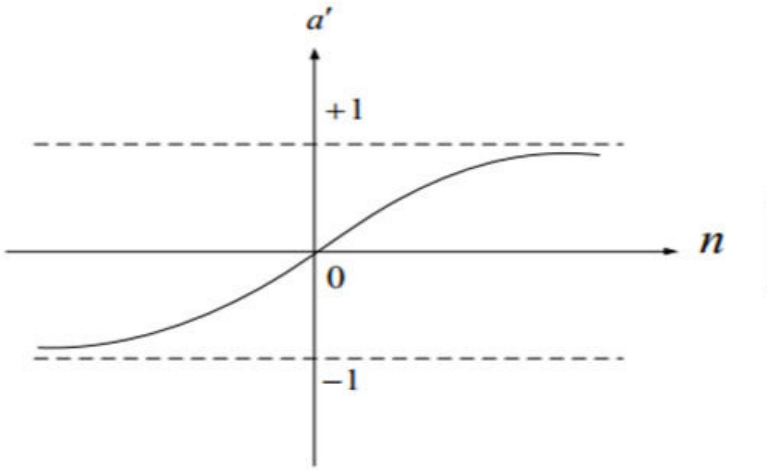



Figure 83: *Tan* Sigmoid Function which can be used as a normalisation technique

104is the equation of the *tan* Sigmoid function:

$$x'_i = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{104}$$

This normalisation technique is somewhat similar to the Min-Max method since it scales values to a specific range. However, Sigmoid normalisation is more robust and shows greater efficiency (Jain et al., 2005). Moreover, there is a more complex form of this technique which is the Double Sigmoid function (Cappelli et al., 2000).

### *Batch Normalisation*

*Batch-norm* is a normalisation technique specifically developed for enhancing *neural network* training. Before this technique was developed, neural networks were only normalised at the input layer only.

The problem with this was that if instability arises in one of the intermediate layers, then the layer would need to readapt to a new distribution thus slowing down the whole process. This problem is known as *internal covariate shift* (Ioffe and Szegedy, 2015). Batch normalisation solved the problem by adding normalisation at each layer in the network. Normalisation is done for every input neuron (one batch per neuron) found in the layer (see also figure 84).

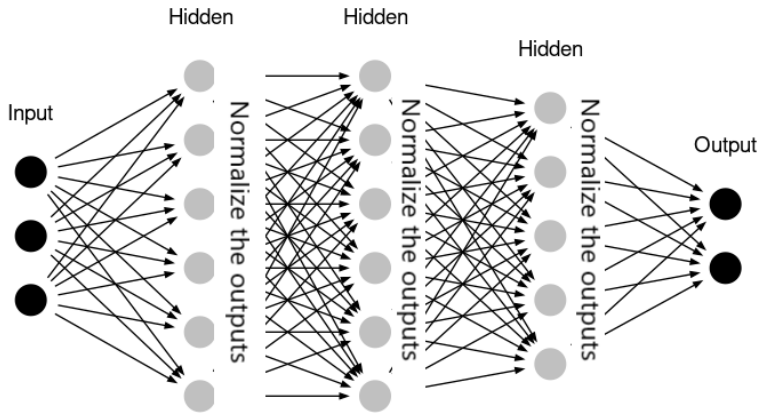



Figure 84: Neural network with applied batch normalisation https://mc.ai/batch-normalization-speed-up-neural-network-training/

## *Conclusion*

From these few standardization and normalisation techniques, we have seen the importance to choose the right technique for the right data set (it is important to find out what kind of problem the data set is presenting - for example a regression problem, classification problem, a time-series, text mining and so on). Some important features that a technique should have are: that it improves accuracy, reduces the training time, simplify the data and robustness to noise.

# *Bibliography*


Ahmed M. Abdelsalam, Pierre Langlois, and Farida Cheriet. Accurate and efficient hyperbolic tangent activation function on fpga using the dct interpolation filter (abstract only). In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 287–287, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4354-1.

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

Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9):735–739, 2006.

T C Alberto, J V Lochter, and T A Almeida. TubeSpam: Comment Spam Filtering on YouTube. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 138–143, dec 2015.

A. Alonso, M. Seguí-Gómez, 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.

Omar Alonso, Daniel E. Rose, and Benjamin Stewart. Crowdsourcing for relevance evaluation. *SIGIR | Special Interest Group on Information Retrieval*, 42(2):9–15, nov 2008.

Alexandre Alves. Stacking machine learning classifiers to identify higgs bosons at the lhc. *Journal of Instrumentation*, 12(05):T05005, 2017.

Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-49, pages 22:1–22:12, Piscataway, NJ, USA, 2016. IEEE Press.

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.

Itamar Arel, Derek Rose, and Thomas Karnowski. Deep machine learning - a new frontier in artificial intelligence research. *Computational Intelligence Magazine, IEEE*, 5(4):13–18, December 2010.

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.

Inci M. Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K. Jain, and Jiayu Zhou. Patient subtyping via time-aware lstm networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 65–74, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4.

Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.

Anja Belz and Albert Gatt. Intrinsic vs. extrinsic evaluation measures for referring expression generation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 197–200. Association for Computational Linguistics, 2008.

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.

Kristin P Bennett and Ayhan Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information processing systems*, pages 368–374, 1999.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13 (Feb):281–305, 2012.

Christopher M Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

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

Ronald N. Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill International Editions, third edition, 2000. ISBN 0-07-116043-4.

PS Bradley, KP Bennett, and Ayhan Demiriz. Constrained k-means clustering. *Microsoft Research, Redmond*, pages 1–8, 2000.

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

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996a.

Leo Breiman. Bias, variance, and arcing classifiers. Technical report, Statistics Department, University of California, 1996b.

Leo Breiman. Arcing the edge. Technical report, Technical Report 486, Statistics Department, University of California at âĂę, 1997.

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

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

Leo Breiman, JH Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. wadsworth, 1984. *Google Scholar*, 1993.

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

Raffaele Cappelli, Dario Maio, and Davide Maltoni. Combining fingerprint classifiers. In *International Workshop on Multiple Classifier Systems*, pages 351–361. Springer, 2000.

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.

Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7:1247–1250, 2014.

N. Chaikla and Yulu Qi. Genetic algorithms in feature selection. *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, 5:538–540, 1999.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011. ISSN 2157-6904.

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.

Chris Cheadle, Marquis P Vawter, William J Freed, and Kevin G Becker. Analysis of microarray data using z score transformation. *The Journal of molecular diagnostics*, 5(2):73–81, 2003.

Lei Chen. *Curse of Dimensionality*, pages 545–546. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9.

Kyunghyun Cho, Bart van Merriënboer, ÇaÄ§lar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, 10 2014. Association for Computational Linguistics.

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.

Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011. ISBN 978-1-57735-514-4.

Dan Ciresan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jurgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, 2:1237âĂŞ1242, November 2013.

Robert T Clemen. Combining forecasts: A review and annotated bibliography. *International journal of forecasting*, 5(4):559–583, 1989.

Stephan Clémençon, Patrice Bertail, and Emilie Chautru. Statistics A Journal of Theoretical and Applied Statistics Sampling and empirical risk minimization Sampling and empirical risk minimization. *View Crossmark data STATISTICS*, 51(1):30–42, 2017. ISSN 1029-4910.

David Colquhoun. *Lectures on biostatistics: an introduction to statistics with applications in biology and medicine*. David Colquhoun, 1971.

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.

Gordon V Cormack, José María Gómez Hidalgo, and Enrique Puertas Sánz. Feature Engineering for Mobile (SMS) Spam Filtering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 871–872. ACM, 2007. ISBN 978-1-59593-597-7.

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.

T.F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. CRC Press, 2000. ISBN 9781420036121.

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.

James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, pages 293–296. ACM, 2010.

Cameron Davidson-Pilon. *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference*. Addison-Wesley Professional, 1st edition, 2015. ISBN 0133902838, 9780133902839.

Christian W. Dawson and Robert Wilby. An artificial neural network approach to rainfall-runoff modelling. *Hydrological Sciences Journal*, 43(1):47–66, 1998.

Anthony Mihirana De Silva and Philip HW Leong. *Grammar-based feature generation for time-series prediction*. Springer, 2015.

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.

Krishna Devulapalli. Neural networks for classification and regression. *Biom Biostat Int J*, 2(6), September 2015.

Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

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

Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. *Machine learning*, 32:1–22, 1998.

Y. Ding, Y. Cheng, X. Cheng, B. Li, X. You, and X. Yuan. Noise-resistant network: a deep-learning method for face-recognition under noise. *EURASIP Journal on Image and Video Processing*, 2017 (43), 2017.

L.E. Dodd and M.S. Pepe. Partial auc estimation and regression. *Biometrics*, 59(3):614–623, 2003.

Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv e-prints*, February 2017.

Chris Drummond and Nathalie Japkowicz. Warning: statistical benchmarking is addictive. Kicking the habit in machine learning. *Journal of Experimental & Theoretical Artificial Intelligence*, 22(1):67–80, 2010.

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

Kai-Bo Duan and S Sathiya Keerthi. Which is the best multiclass svm method? an empirical study. In *International workshop on multiple classifier systems*, pages 278–285. Springer, 2005.

Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3, 2013.

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.

T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters - Special issue: ROC analysis in pattern recognition*, 27(8):861–874, 2006.

Jianchao Fei, Husheng Fang, Qin Yin, Chengsong Yang, and Dong Wang. Restricted stochastic pooling for convolutional neural network. In *Proceedings of the 10th International Conference on Internet Multimedia Computing and Service*, ICIMCS '18, pages 24:1–24:4, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6520-8.

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.

Yoav Freund. An adaptive version of the boost by majority algorithm. *Machine learning*, 43(3):293–318, 2001.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14 (771-780):1612, 1999.

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Springer Series in Statistics, 2001.

T.T. Fung, Y.J. Tan, and K.S. Sim. Convolutional neural network improvement for breast cancer classification. *Expert Systems With Applications*, 120:103–115, 2018.

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.

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

Athinodoros S Georghiades, Peter N Belhumeur, and David J Kriegman. From few to many: Generative models for recognition under variable pose and illumination. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 277–284. IEEE, 2000.

Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016.

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

F. A. Gers and E. Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, November 2001. ISSN 1045-9227.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research (JMLR)*, 3:115–143, March 2003.

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.

Andrew B Goldberg and Xiaojin Zhu. Seeing stars when there aren't many stars: graph-based semi-supervised learning for sentiment categorization. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, pages 45–52. Association for Computational Linguistics, 2006.

A. Gómez Losada. Data science applications to connected vehicles. Technical report, European Commission, Joint Research Centre, 2017.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, pages 2672–2680. MIT Press, 2014a.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014b. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pd.

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

Google. Xla - tensorflow, compiled. Website, 3 2017. URL https://developers.googleblog.com/2017/03/xla-tensorflow-compiled.html.

A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2.

Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, May 2009.

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.

S.G. Gronlund, J.T. Wixted, and L. Mickes. Evaluating eyewitness identification procedures using receiver operating characteristc analysis. *Current Directions In Psychological Science*, 23(1):3–10, 2014.

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

Yue Guan and Jg Dy. Sparse probabilistic principal component analysis. In *AISTATS*, pages 185–192, 2009.

Erkam Guresen and Gulgun Kayakutlu. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3:426 – 433, 2011. ISSN 1877-0509. World Conference on Information Technology.

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar): 1157–1182, 2003.

Isabelle Guyon, Vladimir Vapnik, Bernhard E. Boser, Leon Bottou, and Sara A. Solla. Structural risk minimization for character recognition. Technical report, AT&T Bell Laboratories, Holmdel, NJ 07733, USA, 1992.

Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Jung-Soo Han, Geum-Bae Cho, and Keun-Chang Kwak. A design of convolutional neural network using relu-based elm classifier and its application. In *Proceedings of the 9th International Conference on Machine Learning and Computing*, ICMLC 2017, pages 179–183, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4817-1.

David J Hand. Classifier technology and the illusion of progress. *Statistical science*, pages 1–14, 2006.

J.A. Hanley and B.J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

Frank E Harrell. *Regression modeling strategies*. Springer, 2015.

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.

Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1994. ISBN 0023527617.

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

Y. He and M. Escobar. Nonparametric statistical inference method for paired areas under receiver operating characteristics curves, with application to genomic studies. *Statistics in Medicine*, 27(25): 5291–5308, 2008.

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

Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

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

g. e. hinton. boltzmann machine. *scholarpedia*, 2(5):1668, 2007. revision #91076.

Tin Kam Ho. Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, volume 1, pages 278–282. IEEE, 1995.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998. ISSN 0218-4885.

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

Sepp Hochreiter, Yoshua Bengio, and Paolo Frasconi. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In J. Kolen and S. Kremer, editors, *A Field Guide to Dynamical Recurrent Networks*. IEEE Press, 2001.

Katja Hofmann, Lihong Li, and Filip Radlinski. Online evaluation for information retrieval. *Foundations and Trends in Information Retrieval*, 10(1):1–117, 2016.

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.

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

John H. Holland. Outline for a Logical Theory of Adaptive Systems. *Journal of the ACM*, 1962.

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.

Cheng-Lung Huang and Chieh-Jen Wang. A ga-based feature selection and parameters optimizationfor support vector machines. *Expert Systems with applications*, 31(2):231–240, 2006.

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

David H Hubel and Torsten Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195 (1):215âĂŞ243, March 1968.

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Trans. Graph.*, 35(4):110:1–110:11, July 2016. ISSN 0730-0301.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

Anil Jain, Karthik Nandakumar, and Arun Ross. Score normalization in multimodal biometric systems. *Pattern recognition*, 38(12):2270–2285, 2005.

Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

Unnat Jain, Ziyu Zhang, and Alexander G Schwing. Creativity: Generating diverse questions using variational autoencoders. In *CVPR*, pages 5415–5424, 2017.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York, 2013. ISBN 9781461471387.

Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *CoRR*, abs/1702.05659, 2017.

Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

T Jayalakshmi and A Santhakumaran. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, 3(1):1793–8201, 2011.

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*. Elsevier Science, 2011. ISBN 9780123814807.

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

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.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 1–12. IEEE, 2017.

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

S. Sathiya Keerthi, Vikas Sindhwani, and Olivier Chapelle. An efficient method for gradient-based adaptation of hyperparameters in svm models. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 673–680, Cambridge, MA, USA, 2006. MIT Press.

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

Dennis F. Kibler and Pat Langley. Machine learning as an experimental science. In *Proceedings of the 3rd European Conference on European Working Session on Learning*, pages 81–92. Pitman Publishing, Inc., 1988.

Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

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.

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, 2012a.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012b.

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. *Pattern Analysis & Applications*, 5(2):210–220, 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.

S. Lawrence, C.L. Giles, Ah Chung Tsoi, and A.D. Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98 – 113, January 1997.

Thi-Thu-Huong Le, Jihyun Kim, and Howon Kim. Classification performance using gated recurrent unit recurrent neural network on energy disaggregation. In *2016 International Conference on Machine Learning and Cybernetics (ICMLC)*, pages 105–110. IEEE, 07 2016.

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.

Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013.

Jiefei Li, Xiaocong Liang, Weijie Ding, Weidong Yang, and Rong Pan. Feature Engineering and Tree Modeling for Author-paper Identification Challenge. In *Proceedings of the 2013 KDD Cup 2013 Workshop*, KDD Cup '13, pages 5:1—-5:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2495-3.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

A. Linden. Measuring diagnostic and predictive accuracy in disease management: an introduction to receiver operating characteristic (roc) analysis. *Journal of Evaluation in Clinical Practice*, 12(2):132–129, 2006.

Huan Liu and Hiroshi Motoda. *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media, 2012.

Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering*, 17(4):491–502, 2005.

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.

A. Luby. Strengthening analysis of line-up procedures: a log-linear model framework. *Law, Probability and Risk*, 16(4):241–257, 2017.

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.

H. Ma, A.I. Bandos, H.E. Rockette, and D. Gur. On use of partial area under the roc curve for evaluation of diagnostic performance. *Statistics in Medicine*, 32(20):3449–3458, 2013.

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.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2009. ISBN 0521865719, 9780521865715.

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

Mahdokht Masaeli, Jennifer G Dy, and Glenn M Fung. From transformation-based dimensionality reduction to feature selection. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 751–758, 2010.

David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics, 2006.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2): 153–157, 1947.

Ismail Bin Mohamad and Dauda Usman. Standardization and its effects on k-means clustering algorithm. *Research Journal of Applied Sciences, Engineering and Technology*, 6(17):3299–3303, 2013.

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.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Massachusetts Institute of Technology, 2012.

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.

Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

Siqi Nie, Ziheng Wang, and Qiang Ji. A generative restricted boltzmann machine based method for high-dimensional motion data modeling. *Computer Vision and Image Understanding*, 136:14–22, 2015.

Christopher Olah. Understanding lstm networks, 2015.

Zhijian Ou and Yang Zhang. Probabilistic acoustic tube: a probabilistic generative model of speech for speech analysis/synthesis. In *Artificial Intelligence and Statistics*, pages 841–849, 2012.

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.

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.

Antom Paquin. What's inside a tpu? Website, 6 2018. URL https://medium.com/@antonpaquin/whats-inside-a-tpu-c013eb51973e.

S Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

Vladimir-Alexandru Paun, Bruno Monsuez, and Philippe Baufreton. On the determinism of multi-core processors. In *French Singaporean Workshop on Formal Methods and Applications*, pages 35,36, 2013.

Michael Peter Perrone. *Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization*. PhD thesis, Citeseer, 1993.

S James Press. *Applied multivariate analysis: using Bayesian and frequentist methods of inference*. Courier Corporation, 2005.

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.

Brandon Rohrer. How do convolutional neural networks work?, 2016.

Lior Rokach and Oded Maimon. *Data mining with decision trees: theory and applications*. World Scientific Publishing Company, 2014. ISBN 9789814590099.

Bruce E. Rosen and James M. Goodwin. Training hard to learn networks using advanced simulated annealing methods. In *Proceedings of the 1994 ACM Symposium on Applied Computing*, SAC '94, pages 256–260, New York, NY, USA, 1994. ACM. ISBN 0-89791-647-6.

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.

Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191–198, 2012. ISSN 0167-8655.

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. *International Journal of Computer Applications*, 163(7):975–8887, 2017.

T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3), 2015a.

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, 2015b.

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

G. Saon and M. Picheny. Recent advances in conversational speech recognition using convolutional and recurrent neural networks. *IBM Journal of Research and Development*, 61(4-5):1–10, July 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.

Kaz Sato, Cliff Young, and David Patterson. An in-depth look at google's first tensor processing unit (tpu). Website, 5 2017. URL https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu.

Anton Maximilian Schäfer, Steffen Udluft, and Hans Georg Zimmermann. Learning long term dependencies with recurrent neural networks. In *Proceedings of the 16th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'06, pages 71–80, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-38625-4, 978-3-540-38625-4.

Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, September 1986. ISSN 1573-0565.

Jurgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

Jürgen Schmidhuber, Daan Wierstra, and Faustino Gomez. Evolino: Hybrid neuroevolution / optimal linear search for sequence learning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 853–858, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language*

*Processing*, pages 298–307. Association for Computational Linguistics, 2015.

Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

Martin Sewell. Structural risk minimization, September 2008. URL http://www.svms.org/srm/srm.pdf.

S Shalev-Shwatz. *Understanding Machine Learning*. Cambridge University Press, 2014.

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

John Shawe-Taylor. Structural risk minimization over data-dependent hierarchies. Technical report, University of London, November 1997.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 802–810, Cambridge, MA, USA, 2015. MIT Press.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

W. Siedlecki and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 1989.

S.N. Sivanandam and S.N. Deepa. *Introduction to Genetic Algorithms*. Springer Berlin Heidelberg, 2007. ISBN 9783540731900.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.

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

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

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*, 2015.

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

Hari M. Srivastava and R.G. Buschman. *Theory and Applications of Convolution Integral Equations*. Springer Netherlands, April 2013.

Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014a. ISSN 1532-4435.

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, 2014b.

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.

Katarzyna Stapor. Evaluating and comparing classifiers: Review, some recommendations and limitations. In *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017*, pages 12–21, 2017.

Dave Steinkrau, Ian Buck, and Patrice Y Simard. Using gpus for machine learning algorithms. *Eighth International Conference on Document Analysis and Recognition (ICDAR'05) (2005)*, pages 1115–1119, September 2005.

Jack Stewart. TESLA'S AUTOPILOT WAS INVOLVED IN ANOTHER DEADLY CAR CRASH. Technical report, wired.com, 2018.

Pranjal Strivastava. Essentials of deep learning : Introduction to long short term memory, 2017.

Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformaticsoot*, 2007.

Y Sun, CF Babbs, and EJ Delp. A comparison of feature selection methods for the detection of breast cancers in mammograms: adaptive sequential floating search vs. genetic algorithm. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 6532–6535. IEEE, 2006.

Şakir Taşdemir, Süleyman Neşeli, Ismail Saritaş, and Süleyman Yaldiz. Prediction of surface roughness using artificial neural network in lathe. In *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, CompSysTech '08, pages 41:IIIB.6–41:1, New York, NY, USA, 2008. ACM. ISBN 978-954-9641-52-3.

Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. *Data classification: Algorithms and applications*, page 37, 2014.

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.

Kai Ming Ting and Ian H Witten. Issues in stacked generalization. *Journal of artificial intelligence research*, 10:271–289, 1999.

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.

David S Touretzky, Michael C Mozer, and Michael E Hasselmo. *Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference*, volume 8. Mit Press, 1996.

Alexey Tsymbal. The Problem of Concept Drift: Definitions and Related Work. Technical report, Trinity College Dublin, Ireland, 2004.

Zhuowen Tu. Learning generative models via discriminative approaches. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

Timothy C Urdan. *Statistics in plain English*. Routledge, 2011.

C. J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann Newton, MA, USA, 2 edition, 1979. ISBN 0408709294.

V Vapnik. Principles of Risk Minimization for Learning Theory. Technical report, Bell Laboratories, Holmdel, NJ, 1991.

V Vapnik. *Statistics for Engineering and Information Science*. Springer, New York, NY, 2000.

Vladimir Vapnik. Principles of risk minimization for learning theory. Technical report, AT&T Bell Laboratories, Holmdel, NJ 07733, USA, 1992.

Vladimir Vapnik and Alexey Chervonenkis. *Theory of Pattern Recognition: Statistical Problems of Learning*. Nauka, Moscow, 1974.

N Vapnik Vladimir. Statistical learning theory. *New York: W iley, l998*, 1998.

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.

Alex Waibel, Geoffrey Hinton, K.J. Lang, Kiyohiro Shikano, and Toyoyuki Hanazawa. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328 – 339, March 1989.

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

Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing Concept Drift. *Data Min. Knowl. Discov.*, 30(4):964–994, jul 2016. ISSN 1384-5810.

Louis Wehenkel. Statistical learning theory : a primer. Technical report, Institut Montefiore, 2018.

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.

Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30:79–82, 2005.

David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

Michal Wozniak. A hybrid decision tree training method using data streams. *Knowledge and Information Systems*, 29(2):335–347, November 2011. ISSN 0219-3116.

Xinfeng Xie, Dayou Du, Qian Li, Yun Liang, Wai Teng Tang, Zhong Liang Ong, Mian Lu, Huynh Phung Huynh, and Rick Siow Mong Goh. Exploiting sparsity to accelerate fully connected layers of cnn-based applications on mobile socs. *ACM Trans. Embed. Comput. Syst.*, 17(2):37:1–37:25, December 2017. ISSN 1539-9087.

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

Wenzhuo Yang, Melvyn Sim, and Huan Xu. The coherent loss function for classification. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 37–45. PMLR, 22–24 Jun 2014.

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.

Lean Yu, Shouyang Wang, and Kin Keung Lai. Forecasting crude oil price with an emd-based neural network ensemble learning paradigm. *Energy Economics*, 30(5):2623–2635, 2008.

Wei Zang, Kunio Doi, Maryellen L Giger, Yuzheng Wu, Robert M Nishikawa, and Robert A Schmidt. Computerized detection of clustered microcalcifications in digital mammograms using a shift-invariant artificial neural network. *Medical Physics*, 21(4):517–524, April 1994.

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.

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

Zheng Zhao, Fred Morstatter, Shashvata Sharma, Salem Alelyani, Aneeth Anand, and Huan Liu. Advancing feature selection research. *ASU feature selection repository*, pages 1–28, 2010.

Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

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.

Maciej Zięba, Sebastian K Tomczak, and Jakub M Tomczak. Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction. *Expert Systems with Applications*, 58:93–101, 2016.

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*