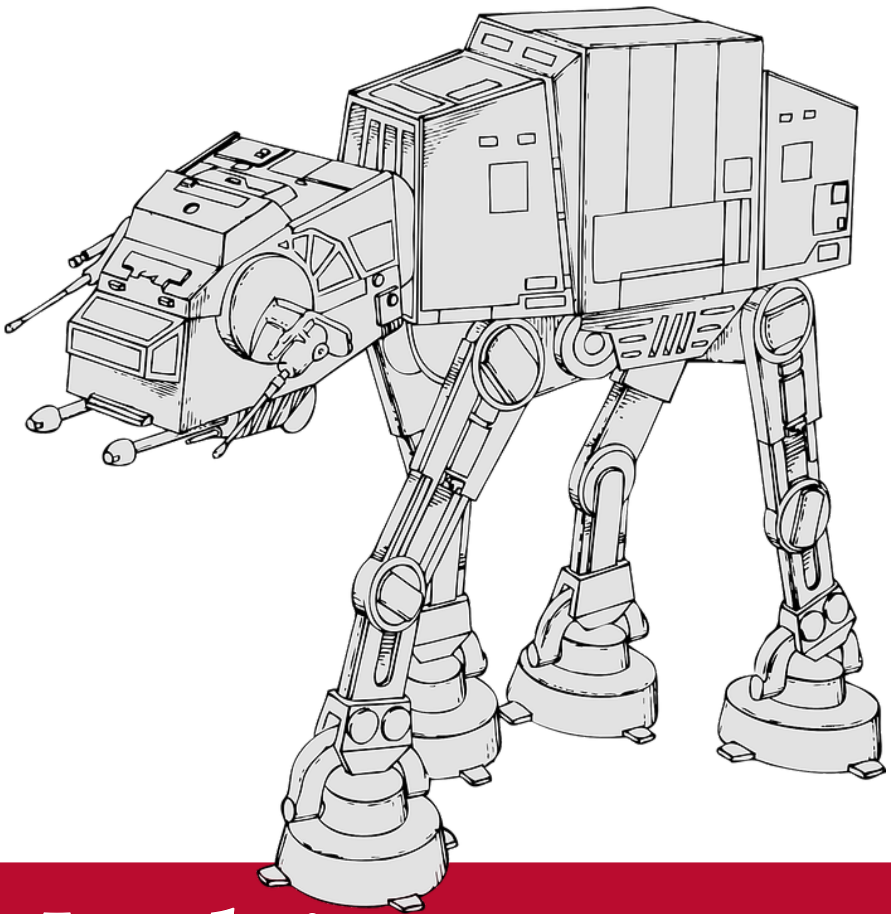
A Machine Learning Study Guide



Machine Learning Handbook

The Definitive Guide

L-Università ta' Malta

ICS5110, class of 2018/9

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

First printing, January 2019

Contents

Introduction

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

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

Activation Functions

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

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

Step Function

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

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

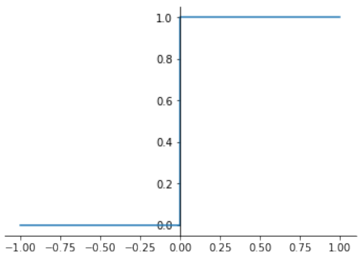


Figure 1: A graph of the step function.

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

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

Linear Functions

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

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

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

Sigmoid Function

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

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

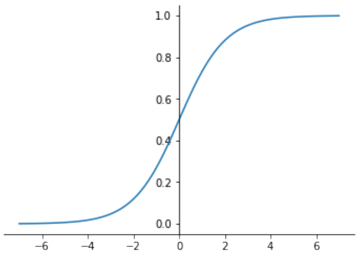


Figure 2: A graph of the sigmoid function.

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

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

Hyperbolic Tangent

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

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

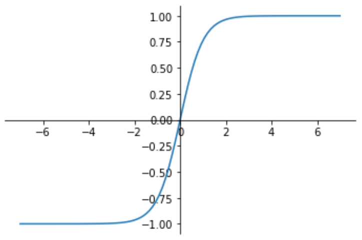


Figure 3: A graph of the hyperbolic tangent (tanh) function.

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

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

Rectified Linear Unit

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

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

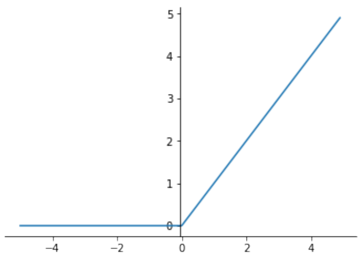


Figure 4: A graph of the ReLU function.

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

The ReLU function has been used in many neural network models to improve their performance. **?** use ReLU to improve the performance of Restricted Boltzmann Machines in object recognition. **?** introduced a breakthrough Convolutional Neural Network (CNN) architecture called AlexNet, which pioneered the use of the ReLU activation function together with dropout layers to minimise over fitting in CNNs.

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

Confusion Matrix

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

		Predicted Class	
		spam	*¬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 (**?**). A CM of a multi-class model would show the number of times the classes were predicted correctly and which classes were confused with each other (Table 2).

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

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

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

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

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

In binary classification, the CM consists of 2 specially designated classes called the *positive* class and the *negative* class (**?**). As indicated in Table 3, positive outcomes from the true class which are classified correctly are called *True Positives* (TP), whilst misclassifications are called *False Negatives* (FN). On the other hand, negative true class outcomes which are classified correctly are called *True Negatives* (TN), and misclassifications are called *False Positives* (FP). In natural sciences, FP are called *Type I Errors* and FN are known as *Type II Errors* (**?**).

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

Table 3: CMs of binary classifiers have positive (+ve) and negative (-ve) classes, and elements called *True Positives* (TP), *False Positives* (FP), *True Negatives* (TN) and *False Negatives* (FN).

The information presented in the CM can be used to evaluate the performance of different binary classifiers (**?**). A number of statistics (Equations 11-17) derived from the CM have been proposed in the literature (**?**) to gain a better understanding of what are the strengths and weaknesses of different classifiers. Caution should be exercised when interpreting metrics (**?**), since the CM could be misleading if the data is imbalanced and an important subrange of the domain is underrepresented (**?**). For instance, an albino zebra classifier which always returns negative will achieve high accuracy since albinism is a rare disorder.

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

Accuracy (ACC) is the proportion of correct predictions (Equation 11). It is a class-insensitive metric because it can give a high rating to a model which classifies majority class objects correctly but misclassifies interesting minority class objects (**?**). The other metrics should be preferred since they are more class-sensitive and give better indicators when the dataset is imbalanced.

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

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

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

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

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

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

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

Sensitivity and Specificity can be combined into a single metric (Equation 15). These metrics are often used in domains in which minority classes are important (?). For example, the Sensitivity of a medical classifier (?) measures how many patients with the condition tested positive, and Specificity measures how many did not have the condition and tested negative.

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

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

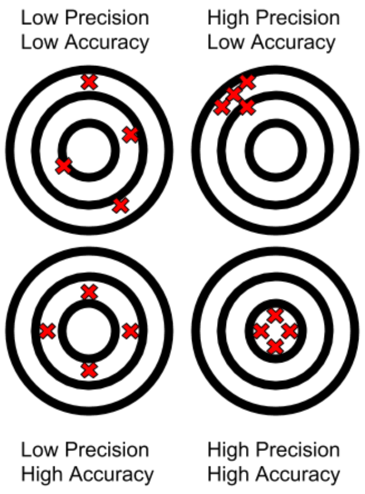


Figure 5: Accuracy vs Precision.

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

Precision and Recall are borrowed from the discipline of *Information Extraction* (?). A composite metric called *F-score*, *F1-score*, or *F-measure* (Equation 17) can be derived by finding their harmonic mean (?).

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

The complements of ACC, NPV, TNR, TPR and PPV are called, respectively, *Error Rate*, *False Omission Rate*, *false positive rate*, *false negative rate* and *False Discovery Rate*.

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

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

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

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

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

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

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

Taking into account the above, CMs are suitable for visualising, evaluating, and comparing the performance of binary or multi-class classifiers. They should be used in conjunction with metrics such as the F-measure to avoid bias, especially if the dataset is unbalanced. For further details on the theoretical aspects of CMs and for practical examples in R refer to (**?**); for examples in Python refer to (**?**).

The following example is motivated by the samples in the *Scikit-Learn* documentation and the work of (**?**). The models in Figure 6 were trained on the *wines* dataset included with Scikit-Learn.

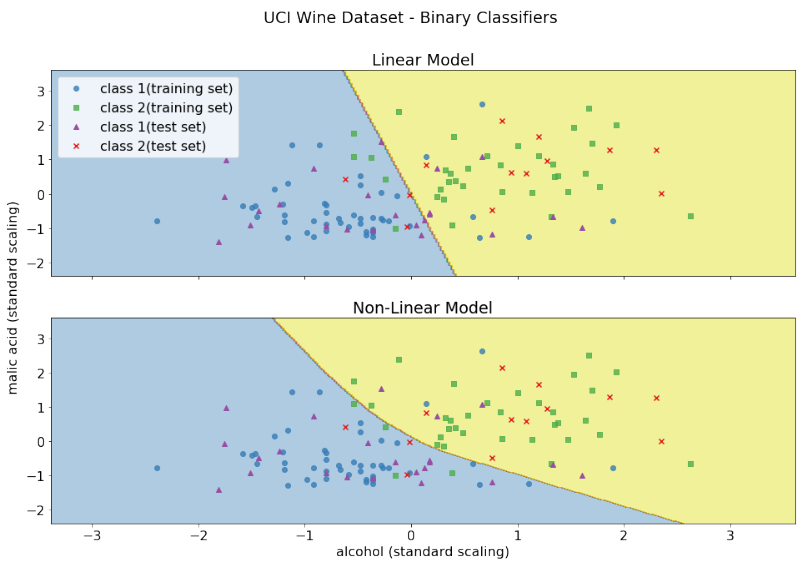


Figure 6: Decision boundary learned by a linear and non-linear binary classifier.

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

Table 6: Statistics derived from the CMs in Figure 7.

As it can be intuitively deduced from Figure 6, the decision boundary of the non-linear model is a better fit than the linear model. The CMs in Figure 7 also show that non-linear model performs better with a higher TP, and consequently lower TN. The biggest advantage of the non-linear model is the higher Sensitivity resulting in a better F-score.

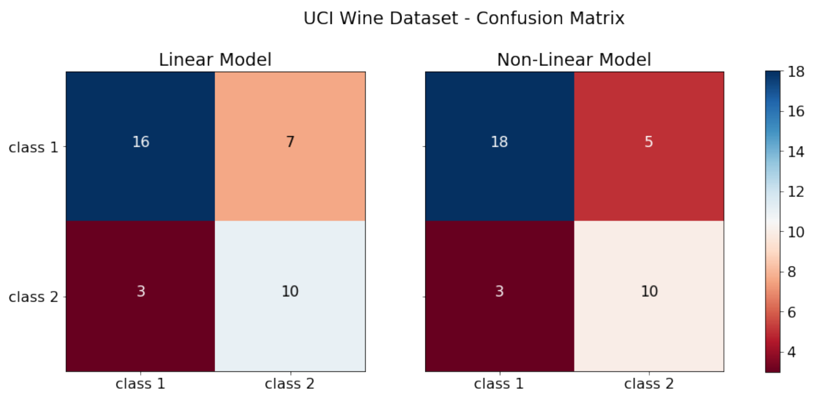


Figure 7: The linear classifier has 16 TP, 10 TN, 7 FN and 3 FP, whilst the non-linear classifier has 18 TP, 10 TN, 5 FN and 3 FP.

Cross-Validation

Cross-validation (CV) is an estimation method used on supervised learning algorithms to assess their ability to predict the output of unseen data (**??**). 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 (**?**). 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 (**??**). Therefore, cross-validation, also known as out-of-sample testing, tests the function's ability to generalize to unseen situations (**??**).

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 (**?**).

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

Exhaustive cross-validation

Leave-p-out (LpO)

This method takes p samples from the data set as the test set and keeps the remaining as the training set, as shown in Fig. 9a. This is repeated for every combination of test and training set formed from the original data set and the average error is obtained. Therefore, this method trains and tests the algorithm $\binom{n}{p}$ times when the number of samples in the original data set is n, becoming inapplicable when $p > 1$ (**?**).

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 (?).

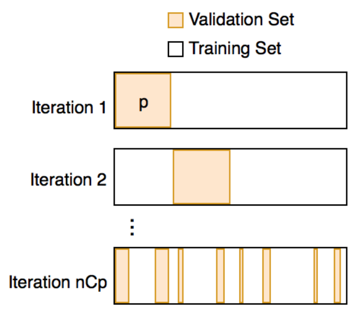


Figure 8: Leave-p-Out Exhaustive Cross Validation

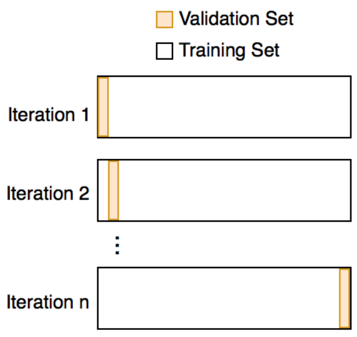


Figure 9: Leave-One-Out Exhaustive Cross Validation

Non-exhaustive cross-validation

Holdout method

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

k-fold

This method randomly splits the original data set into k equally sized subsets, as shown in Fig. 10. The function is then trained and validated k times, each time taking a different subset as the test data and the remaining $(k-1)$ subsets as the training data, using each of the k subsets as the test set once. The k results are averaged to produce a single estimation. Stratified k-fold cross validation is a refinement of the k-fold method, which splits the original samples into equally sized and distributed subsets, having the same proportions of the different target labels (?).

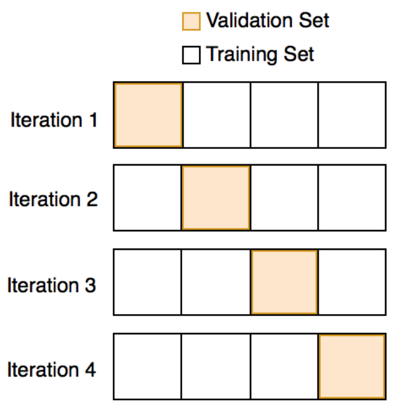


Figure 10: k-Fold Cross Validation where k=4

Repeated random sub-sampling

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

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 (??). In the case of classification problems, the misclassification error rate is usually used whilst for regression problems, the mean squared error (MSE) is usually used. MSE is represented by Eq. 18, where n is the total number of test samples, Y_i is the true value of the i^{th} instance and $\hat{Y}_i$ is the predicted value of the i^{th} instance.

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

Underfitting is when the model has a low degree (e.g. $y = x$, where the degree is 1) and so is not flexible enough to fit the data making the model have a low variance and high bias (?), as seen in Fig. 12a. Variance is the model's dependence on the training data and bias is model's assumption about the shape of the data (?). On the other hand, as seen in Fig. 12b, overfitting is when the model has a too high degree (e.g. $y = x^{30}$, where the degree is 30) causing it to exactly fit the data as well as the noise and so lacks the ability to generalize (?), 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 (?).

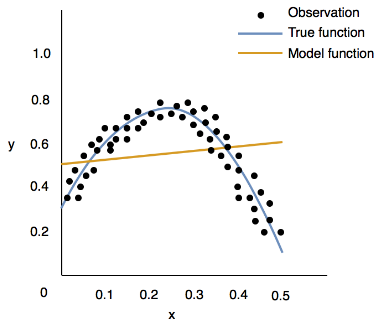


Figure 11: Model Underfitting

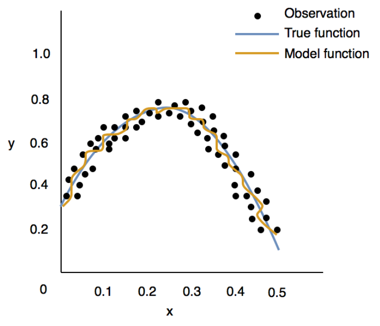


Figure 12: Model Overfitting

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

Therefore, lets take an example using the stratified k-fold cross-validation method with $k = 10$. Let's say that we are trying to solve age group classification, using eight non-overlapping age groups being 0-5, 6-10, 11-20, 21-30, 31-40, 41-50, 51-60, and 61+. We are using the FG-NET labelled data set, which contains around 1000 images of individuals aged between 0 and 69. Before we can start training our model (e.g. CNN), we must divide our data set into training and test subsets and this is where cross validation comes in. Therefore, we start by taking the 1000 images of our data set and splitting them according to their target class. Let us assume we have an equal amount of 125 $(1000/8)$ images per class[2]. As depicted in Fig. 13, we can now start forming our 10 folds by taking 10% of each age-group bucket, randomly without replacement. Hence, we will 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

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

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

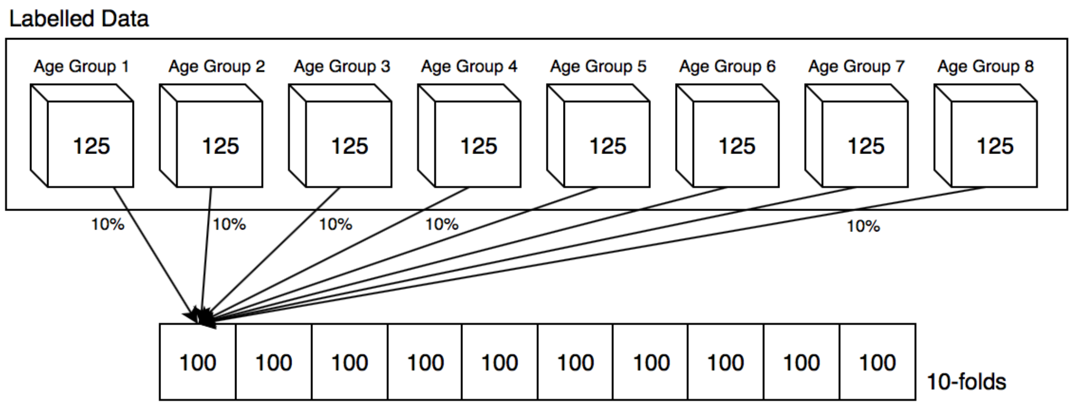


Figure 13: Stratified 10-fold cross-validation on 1000 labelled images of 8 different classes

As concluded by **?**, 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.

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 (**?**):

$$L(f(x), y) \tag{19}$$

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. In almost all cases, it is impossible to collect the entire data as it almost impossible. 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 all into one set of data. Since entire population is not available there needs to be a way of minimising the true risk, without the true distribution being known.

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 the risk is 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 should be minimised also.

Equation of Empirical Risk Minimisation

The following is the equation for the Empirical Risk Minimisation as proposed by **?**(**?**):

$$R_{emp}(f) = \frac{1}{m}\sum_{i=1}^{m} L(f(x_i), y_i). \tag{20}$$

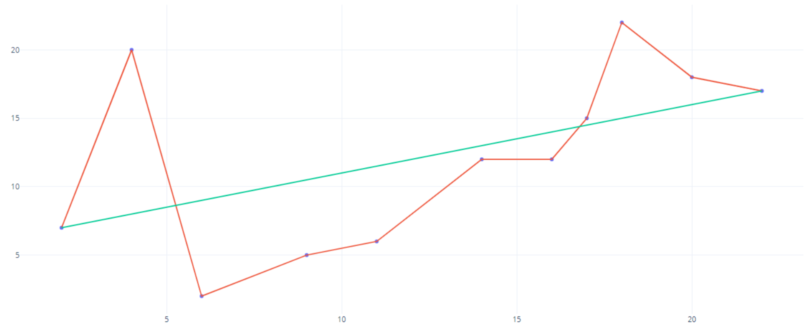
Here R_{emp} denotes the empirical risk or empirical error for a particular prediction function $f(x)$. The number of input and output pairs is denoted as m. As can be seen in the equation above, empirical risk equates to the sum average of the of the loss function on each pair of x and y points generated. 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 (**?**). 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 (**?**).



The figure 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 of an approximation of 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.