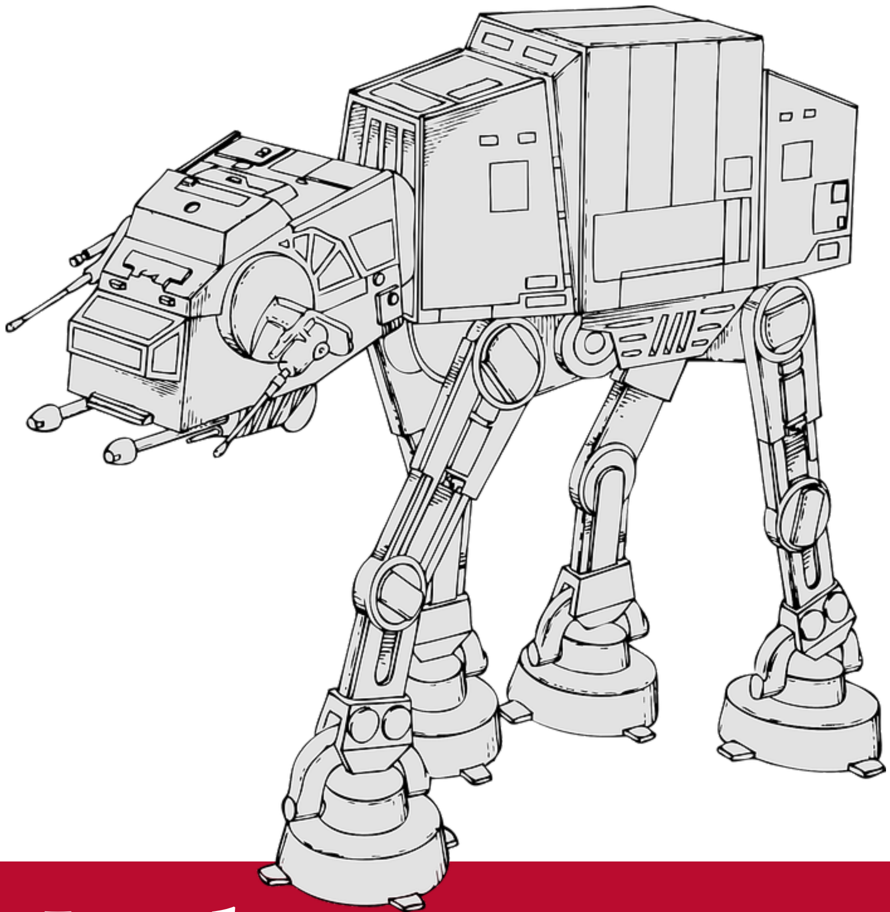
A Machine Learning Study Guide



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 ADD YOUR NAME TO THIS LIST

First printing, January 2019

Contents

Introduction

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

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

Activation Functions

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

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

Step Function

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

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

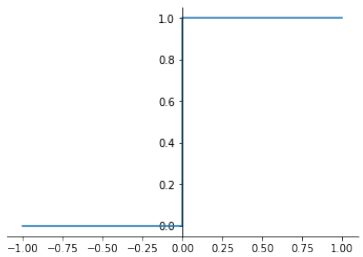


Figure 1: A graph of the step function.

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

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

Linear Functions

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

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

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

Sigmoid Function

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

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

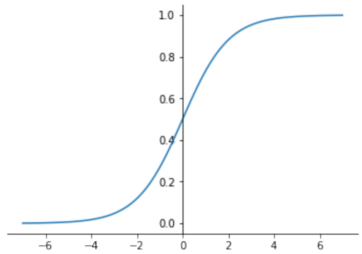


Figure 2: A graph of the sigmoid function.

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

On the other hand, this last property makes the sigmoid function very susceptible to the vanishing gradient problem (Bengio et al., 1994). When observing the shape of the sigmoid function we see that towards the ends of the curve, the function becomes very unresponsive to changes in the input. In other words, the gradient of the function for large inputs becomes very close to 0. This can become very problematic for neural networks that are very deep in design, such as recurrent neural networks (RNNs).To address this problems in RNNs Long Short-Term Memory (LSTM) units where introduced as a variant of the traditional RNN architecture (Hochreiter and Schmidhuber, 1997).

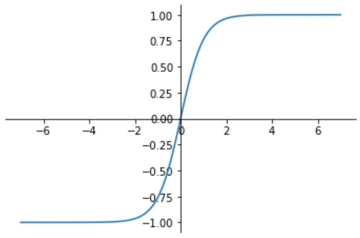


Figure 3: A graph of the hyperbolic tangent (tanh) function.

Hyperbolic Tangent

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

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

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

LeCun et al. (2012) analysed various factors that affect the performance of backpropagation, and suggested that tanh may be better suited than sigmoid as an activation function due to its symmetry about the origin, which is more likely to produce outputs that are on average close to zero, resulting in sparser activations. This means that not all nodes in the network need to be computed, leading to better performance. Glorot and Bengio (2010) studied in detail the effects of the sigmoid and tanh activation functions and noted how the sigmoid function in particular is not well suited for deep networks with random initialisation and go on to propose an alternative normalised initialisation scheme which produced better performance in their experiments.

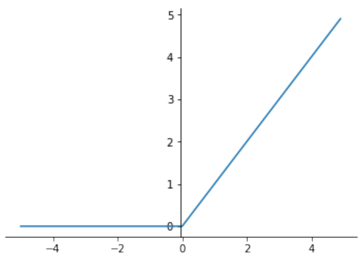


Figure 4: A graph of the ReLU function.

Rectified Linear Unit

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

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

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

The ReLU function has been used in many neural network models to improve their performance. Nair and Hinton (2010) use ReLU to improve the performance of Restricted Boltzmann Machines in object recognition. Krizhevsky et al. (2012) introduced a breakthrough Convolutional Neural Network (CNN) architecture called AlexNet, which pioneered the use of the ReLU activation function together with dropout layers to minimise over fitting in CNNs.

Unfortunately, because the gradient of the function for inputs that are negative is 0, as seen in equation 10, the ReLU function can still be susceptible to the vanishing gradient problem. To manage this problem a variant of the ReLU function, called Leaky ReLU is sometimes used. Rather than simply returning 0 for negative inputs, the leaky ReLU returns a very small value such as $0.01x$. Maas et al. (2013) compared the performance of Sigmoid, ReLU and Leaky ReLU functions and found that while the the performance of both the ReLU and Leaky ReLU functions was better than the performance achieved with the sigmoid function, the performance of the two ReLU functions was nearly identical.

Confusion Matrix

A *confusion matrix* (CM), is a contingency table showing how well a model classifies categorical data. By convention (Sammut and Webb, 2017), the CM of an N-class model is an N×N matrix indexed by the true class in the row dimension and the predicted class in the column dimension (Table 1).

		Predicted Class	
		spam	*¬spam*
True Class	*spam*	10	1
	¬spam	2	100

Table 1: CM of a hypothetical binary classifier which predicts whether out-of-sample text objects are spam or not. In this example, 10 spam and 100 non-spam objects are classified correctly, whilst 1 spam and 2 non-spam objects are misclassified.

Even though CMs are commonly used to evaluate binary classifiers, they are not restricted to 2-class models (Martin and Jurafsky, 2018). A CM of a multi-class model would show the number of times the classes were predicted correctly and which classes were confused with each other (Table 2).

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

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

The CM of the model $h : X \mapsto C$ over the concept $c : X \mapsto C$ using dataset $S \subset X$ is formally defined as a matrix Ξ such that $\Xi_{c,S}(h)[d_1, d_2] = |S_{h=d_1,c=d_2}|$ (Cichosz, 2014). The CM is constructed by incrementing the element corresponding to the true class *vis-a-vis* the predicted class for each object in the dataset (Algorithm 1).

```
Ξ ← 0
for x ∈ S do
    d₁ ← c(x)
    d₂ ← h(x)
    Ξ_{d₁,d₂} ← Ξ_{d₁,d₂} + 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 11-17) derived from the CM have been proposed in the literature (Deng et al., 2016) to gain a better understanding of what are the strengths and weaknesses of different classifiers. Caution should be exercised when interpreting metrics (Jeni et al., 2013), since the CM could be misleading if the data is imbalanced and an important subrange of the domain is underrepresented (Raeder et al., 2012). For instance, an albino zebra classifier which always returns negative will achieve high accuracy since albinism is a rare disorder.

These metrics are important in situations in which a particular type of misclassification, i.e. FP or FN, could have worse consequences than the other (Hassanien and Oliva, 2017). For example, FP are more tolerable than FN in classifiers which predict whether a patient has a disease. Both outcomes are undesirable, but in medical applications it is better to err on the side of caution since FN could be fatal.

Accuracy (ACC) is the proportion of correct predictions (Equation 11). It is a class-insensitive metric because it can give a high rating to a model which classifies majority class objects correctly but misclassifies interesting minority class objects (Branco et al., 2016). The other metrics should be preferred since they are more class-sensitive and give better indicators when the dataset is imbalanced.

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

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

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

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

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

True positive rate (TPR), also called *sensitivity* or *recall*, is the ratio of the correct positive predictions from the total true positives (Equation 14).

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

Sensitivity and specificity can be combined into a single metric (Equation 15). These metrics are often used in domains in which minority classes are important (Kuhn and Johnson, 2013). For example, the sensitivity of a medical classifier (El-Dahshan et al., 2010) measures how many patients with the condition tested positive, and specificity measures how many did not have the condition and tested negative.

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

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.

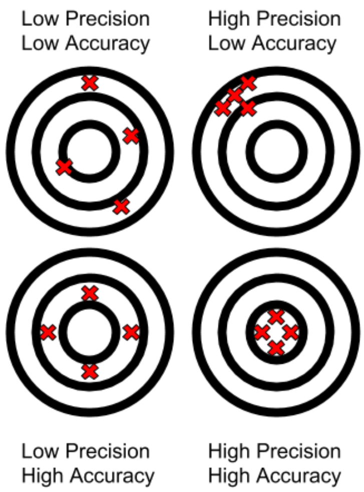


Figure 5: Accuracy vs Precision.

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

Precision and recall are borrowed from the discipline of *information extraction* (Sokolova and Lapalme, 2009). A composite metric called *F-score*, *F1-score*, or *F-measure* (Equation 17) can be derived by finding their harmonic mean (Kelleher et al., 2015).

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

The complements of ACC, NPV, TNR, TPR and PPV are called, respectively, *error rate*, *false omission rate*, *false positive rate*, *false negative rate* and *false discovery rate*.

The metrics can be adapted for evaluating multi-class models by decomposing an N-class CM into 2-class CMs, and evaluating them individually (Stager et al., 2006). The literature describes two methods for decomposing this kind of CM. In the *1-vs-1* approach, 2-class CMs are constructed for each pairwise class as shown in Table 4.

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

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

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

+ve	-ve
M&M's	Skittles $\cup$ Smarties
Skittles	M&M's $\cup$ Smarties
Smarties	Skittles $\cup$ M&M's

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

Using all metrics could be counterproductive due to information redundancy, but none of the metrics is enough on its own (Ma and Cukic, 2007). For instance, recall is class-sensitive but it would give a perfect score to an inept model which simply returns the positive class. Thus, the best approach is to evaluate with complementary pairs (Gu et al., 2009) such as sensitivity *vs* specificity, or precision *vs* recall; or a combined measure such as the F-score.

Taking into account the above, CMs are suitable for visualising, evaluating, and comparing the performance of binary or multi-class classifiers. They should be used in conjunction with metrics such as the F-measure to avoid bias, especially if the dataset is unbalanced. For further details on the theoretical aspects of CMs and for practical examples in R refer to (Cichosz, 2014); for examples in Python refer to (Müller et al., 2016).

The following example is motivated by the samples in the *Scikit-Learn* documentation and the work of (Géron, 2017). The models in Figure 6 were trained on the *wines* dataset included with Scikit-Learn.

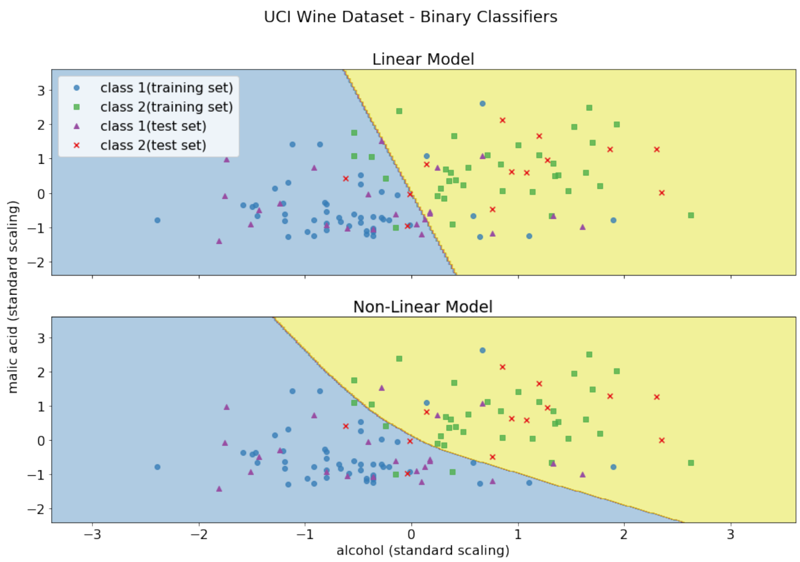


Figure 6: Decision boundary learned by a linear and non-linear binary classifier.

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

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

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

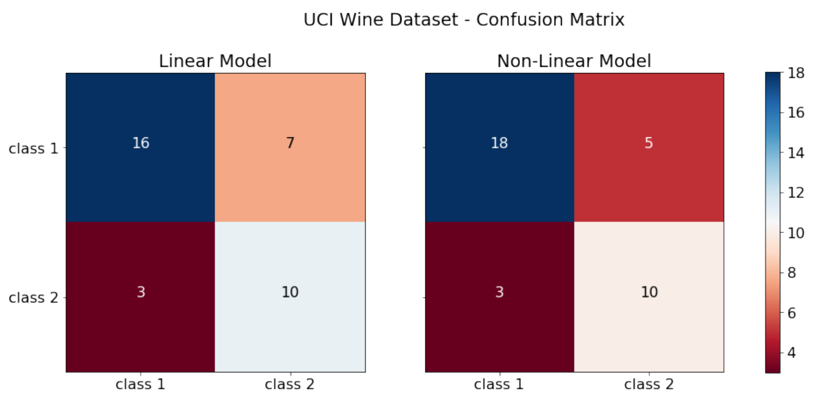


Figure 7: The linear classifier has 16 TP, 10 TN, 7 FN and 3 FP, whilst the non-linear classifier has 18 TP, 10 TN, 5 FN and 3 FP.

Cross-Validation

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

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

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

Exhaustive cross-validation

Leave-p-out (LpO)

This method takes p samples from the data set as the test set and keeps the remaining as the training set, as shown in Fig. 9a. This is repeated for every combination of test and training set formed from the original data set and the average error is obtained. Therefore, this method trains and tests the algorithm $\binom{n}{p}$ times when the number of samples in the original data set is n, becoming inapplicable when $p > 1$ (Arlot et al., 2010).

Leave-one-out (LOO)

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

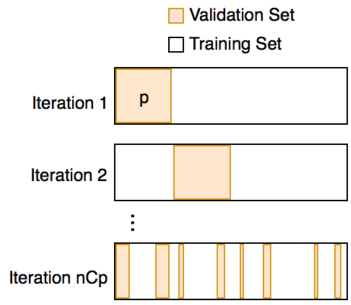


Figure 8: Leave-p-Out Exhaustive Cross Validation

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).

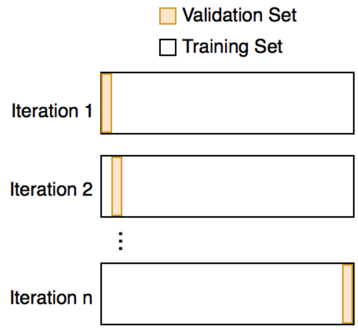


Figure 9: Leave-One-Out Exhaustive Cross Validation

k-fold

This method randomly splits the original data set into k equally sized subsets, as shown in Fig. 10. The function is then trained and validated k times, each time taking a different subset as the test data and the remaining $(k - 1)$ subsets as the training data, using each of the k subsets as the test set once. The k results are averaged to produce a single estimation. Stratified k-fold cross validation is a refinement of the k-fold method, which splits the original samples into equally sized and distributed subsets, having the same proportions of the different target labels (Kohavi, 1995).

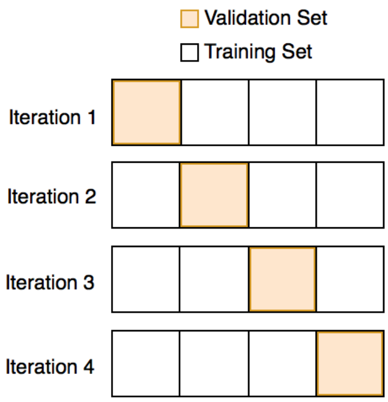


Figure 10: k-Fold Cross Validation where k=4

Repeated random sub-sampling

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

All of the above cross-validation methods are used to check whether the model has been overfitted or underfitted and hence estimating the model's ability of fitting to independent data . Such ability is measured using quantitative metrics appropriate for the model and data (Kohavi, 1995; Arlot et al., 2010). In the case of classification problems, the misclassification error rate is usually used whilst for regression problems, the mean squared error (MSE) is usually used. MSE is represented by Eq. 18, where n is the total number of test samples, Y_i is the true value of the i^{th} instance and $\hat{Y}_i$ is the predicted value of the i^{th} instance.

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

Underfitting is when the model has a low degree (e.g. $y = x$, where the degree is 1) and so is not flexible enough to fit the data making the model have a low variance and high bias (Baumann, 2003), as seen in Fig. 12a. Variance is the model's dependence on the training data and bias is model's assumption about the shape of the data (Arlot et al., 2010). On the other hand, as seen in Fig. 12b, overfitting is when the model has a too high degree (e.g. $y = x^{30}$, where the degree is 30) causing it to exactly fit the data as well as the noise and so lacks the ability to generalize (Baumann, 2003), making the model have a high variance. Cross-validation helps reduce this bias and variance since it uses most of the data for both fitting and testing and so helps the model learn the actual relationship within the data. This makes cross-validation a good technique for models to acquire a good bias-variance tradeoff (Arlot et al., 2010).

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

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

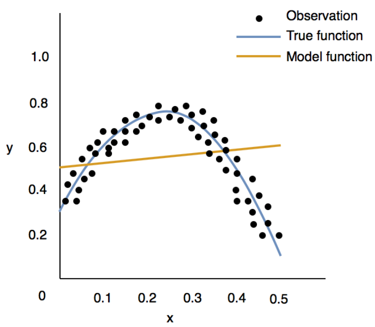


Figure 11: Model Underfitting

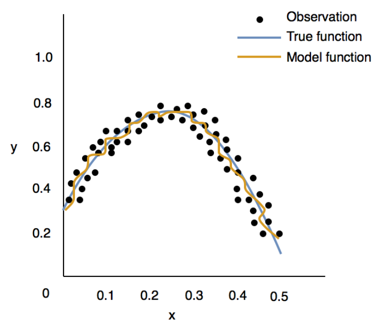


Figure 12: Model Overfitting

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

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

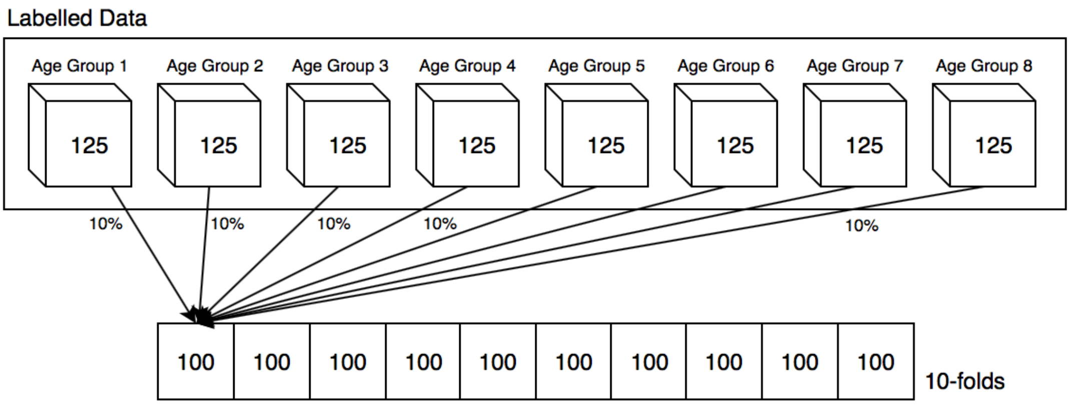


Figure 13: Stratified 10-fold cross-validation on 1000 labelled images of 8 different classes

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

Dimensionality Reduction

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

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

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

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

Random Forests

Breiman (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 14: Covariance Matrix.

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

The resultant diagonalized covariance matrix is used to calculate the *eigenvalues* and *eigenvectors*. The eigenvectors refer to the direction in which the data varies, and these are ordered such that the ones with the highest eigenvalues are chosen as the principal components of the data. Eventually, the selected vectors are kept to form a *feature vector*, which will be used to construct the new dataset with lower dimensionality, as in Equation 19

$$NewDataset = FeatureVector^T x AdjustedData^T \tag{19}$$

Equation 19 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 15 depicts the projection of a two-dimensional feature vector to a one-dimensional feature space. C_1, C_2 and C_3 denote the three different classes being used within the feature vectors. By projecting the feature vectors onto line $\hat{\mathrm{W}}$, 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 $\hat{\mathrm{W}}$, resulting in projection W, where the feature vectors of the classes are highly separated. Algorithmically, Fisher's criterion is maximized to obtain the required one-dimensional feature vector W.

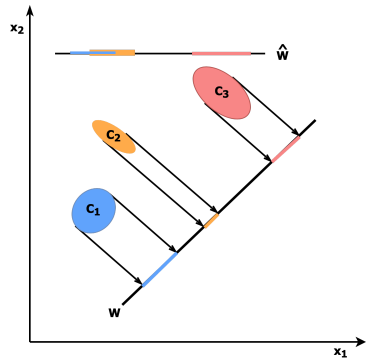


Figure 15: Projection of a two-dimensional feature vector to a one-dimensional feature space. Adapted from Sharma and Paliwal (2015).

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.

$$S_B = \sum_{i=1}^{c} n_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{20}$$

This is referred to as the *between-class matrix* and is represented by Equation 20, where c is the number of classes, n_i is the number of samples for the i^{th} class, μ_i is the mean for the i^{th} class and μ is the total mean of all samples.

$$S_W = \sum_{j=1}^{c} \sum_{i=1}^{n_j} (x_{ij} - \mu_j)(x_{ij} - \mu_j)^T \tag{21}$$

Following this, the *within-class matrix* is calculated using Equation 21, where c is the number of classes, n_j is the number of samples for the j^{th} class, x_{ij} is the i^{th} sample in the j^{th} class, and μ_j is the mean for the j^{th} class. This matrix indicates the distance between the mean and sample values for each class. Finally, a *transformation matrix* W which maximizes the between-class variance and minimizes the within-class variance should be constructed.

$$\arg\max_{w} \frac{W^T S_B W}{W^T S_W W} \tag{22}$$

$$S_W W = \lambda S_B W \tag{23}$$

Fisher's criterion, shown in Equation 22 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 23, where λ denotes the eigenvalues, eigenvectors and the corresponding eigenvalues of W may be calculated. A lower dimensional space is constructed by selecting the eigenvectors having the highest eigenvalues.

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 24. If it is a regression problem an average of the predicted values is taken as the final output as shown in Equation 25. This procedure can be better visualized in Figure 16. The idea behind this technique is to avoid overfitting by reducing variance.

$$f(x) = argmax \sum_{b=1}^{B} f_b(y|x) \tag{24}$$

$$f(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x) \tag{25}$$

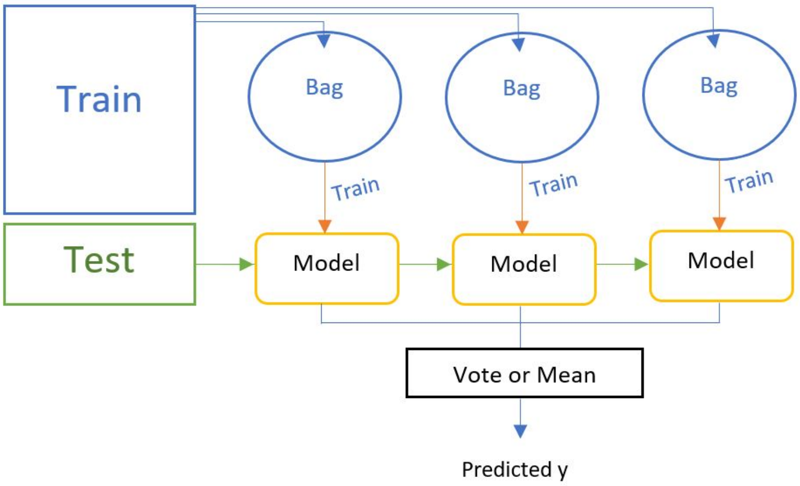


Figure 16: Dataset Bagging using 3 base learners. Each bag is a subset of the training set. Instances are selected randomly and with replacement. Each base learner is fitted with the respective bag. Predictions are then made by each base learner and the final output $\hat{y}$ is the maximum vote or an 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 17. 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 26. If it is a classification problem, it takes a weighted vote to come up with the predicted classification as shown in Equation 27.

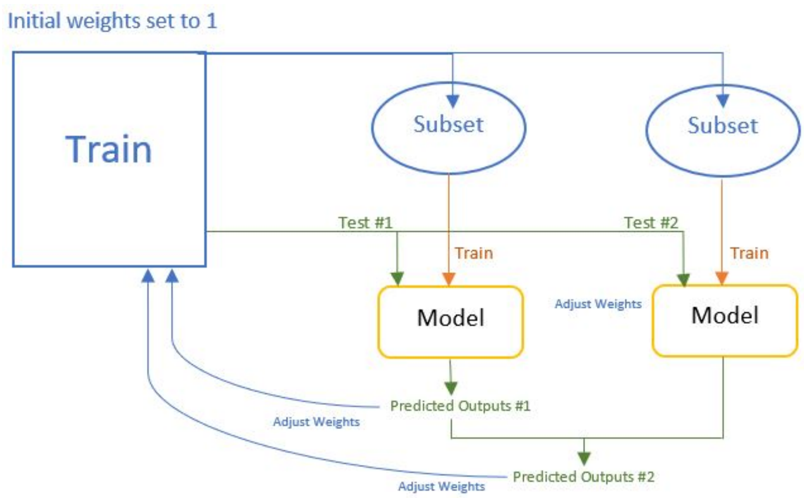


Figure 17: Boosting using 2 base learners. Initially weights are set to 1, so all instances have the same probability to be selected in the subset. The first learner is trained using the first subset and then validated using the whole training set. Weights are recalibrated depending on the errors of 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.

$$f(x) = \frac{1}{B}\sum_{b=1}^{B} \alpha_b f_b(x) \tag{26}$$

$$f(x) = sign(\sum_{b=1}^{B} \alpha_b f_b(y|x)) \tag{27}$$

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 18. 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).

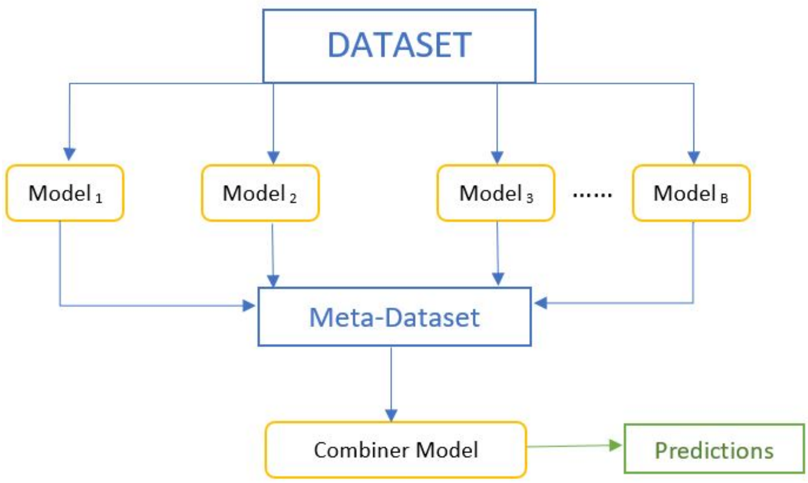


Figure 18: A visual representation of the steps required when Stacking multiple machine learning models.

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).

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 19.

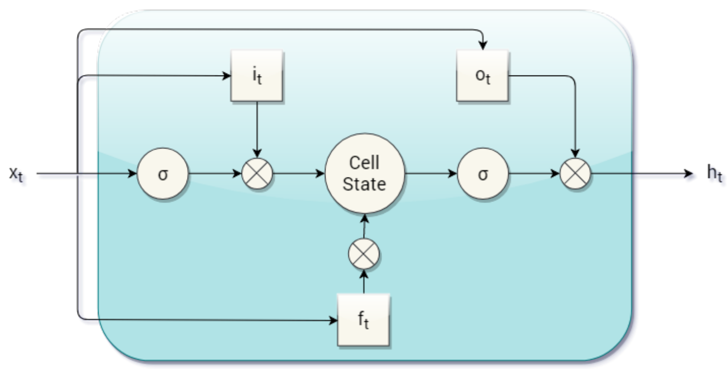


Figure 19: Architecture of standard LSTM module. Adapted from: Olah (2015)

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:

σ_g - denoting the Sigmoid Function (Rumelhart et al., 1986). σ_c - denoting the Hyperbolic Tangent Function. σ_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

$$h_t = o_t \circ tanh(c_t) \tag{28}$$

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

$$\sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \tag{29}$$

- c_t - the cell transfer state, denoted by

$$f_t \circ c_{t-1} + i_t \circ tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{30}$$

- f_t - the forget gate, denoted by

$$\sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \tag{31}$$

- i_t - the input gate, denoted by

$$\sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \tag{32}$$

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 20.

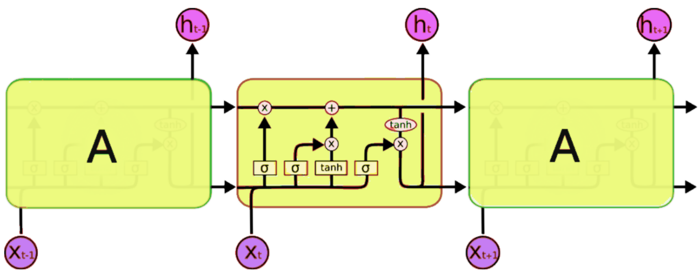


Figure 20: Typical LSTM Network. Retrieved from: Strivastava (2017)

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 21.

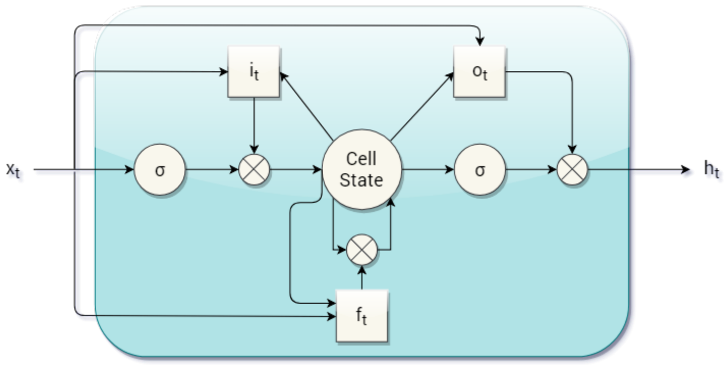


Figure 21: Peephole LSTM module variation, Adapted from Graves et al. (2013)

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 22 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 23. 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:

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ (W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \tag{33}$$

where

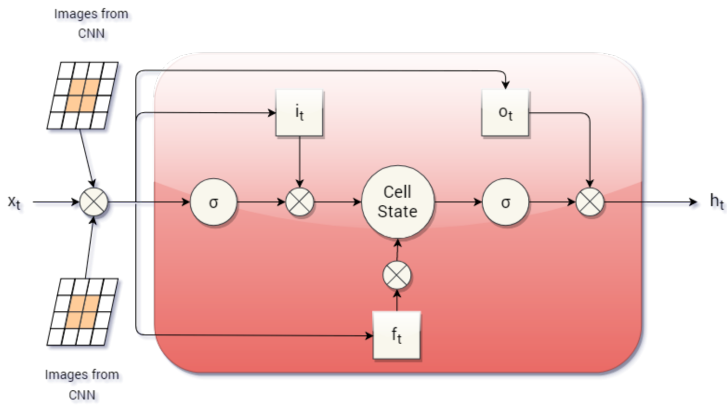


Figure 22: Peephole Convolutional LSTM module variation. Adapted from Gers et al. (2003)

- 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 $\sigma_g(W_r x_t + U_r h_{t-1} + b_r)$

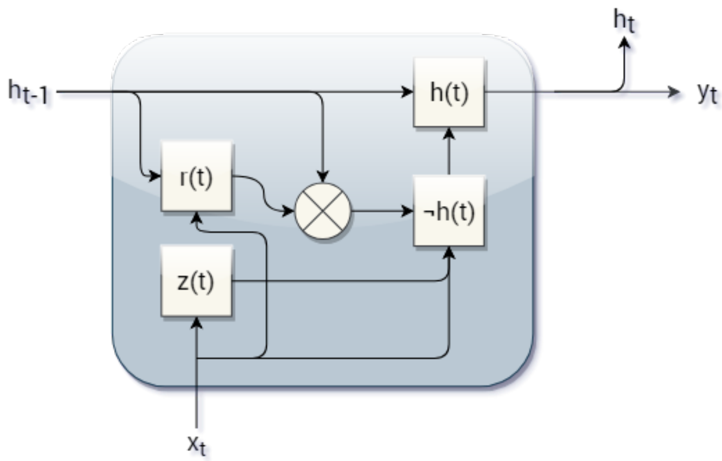


Figure 23: Structure of GRU, based on figure by Le et al. (2016).

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.

Noise in datasets

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

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

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

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

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

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

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

Effects of noise

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

This can be seen in Figure 24 that shows the effect of noise on the classification accuracy of various classifiers.

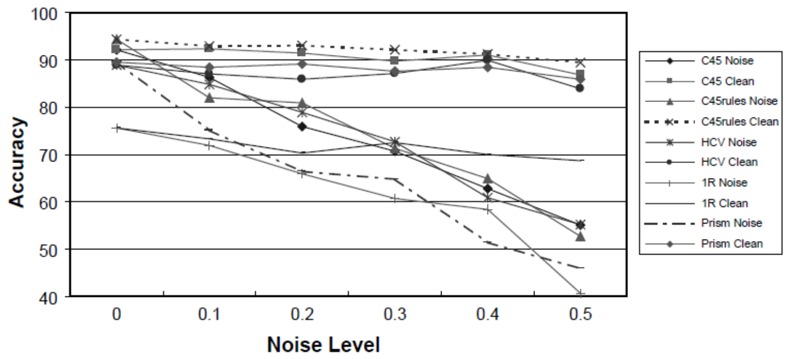


Figure 24: A graph showing the effect of noise on the classification accuracy of various classifiers. Each model was trained on a noisy dataset and on a manually cleaned dataset. These are represented as *XXX Noise* and *XXX Clean* respectively, where *XXX* denotes the classifier in question (Zhu and 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 25 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 34) and the number of incorrect instances that are not filtered (*Type II errors* -

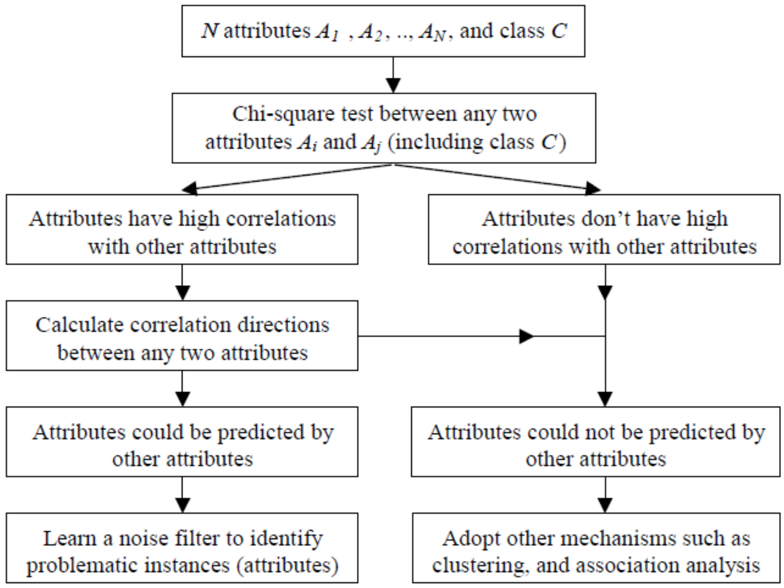


Figure 25: The approach adopted by Zhu and Wu (2004) to filter and correct attribute noise.

Equation 35). These are calculated as follows:

$$ER_1 = \frac{\text{\# of correctly labelled instances which are removed}}{\text{\# of correctly labelled instances}} \tag{34}$$

$$ER_2 = \frac{\text{\# of mislabelled instances which are not removed}}{\text{\# of mislabelled instances}} \tag{35}$$

Consequently, the *Noise Elimination Precision* (NER) can be calculated through Equation 36 (Frénay and Verleysen, 2014).

$$NER = \frac{\text{\# of mislabelled instances which are removed}}{\text{\# of removed instances}} \tag{36}$$

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

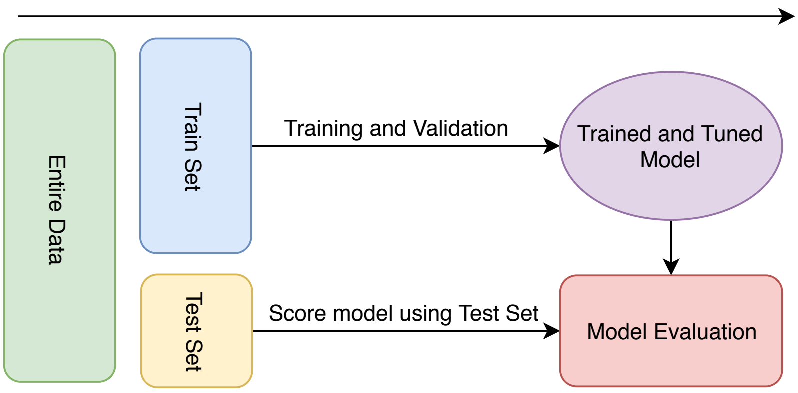


Figure 26: The traditional batch train-test machine learning approach workflow.

In the traditional machine learning approach depicted in Figure 26, 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

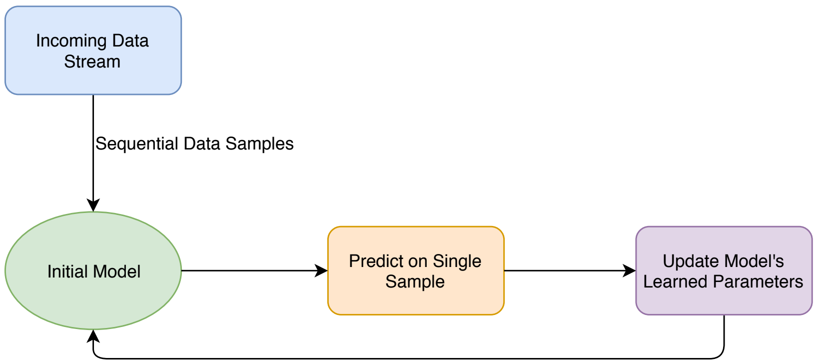


Figure 27: Basic workflow architecture of online learning.

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 27), 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 28:

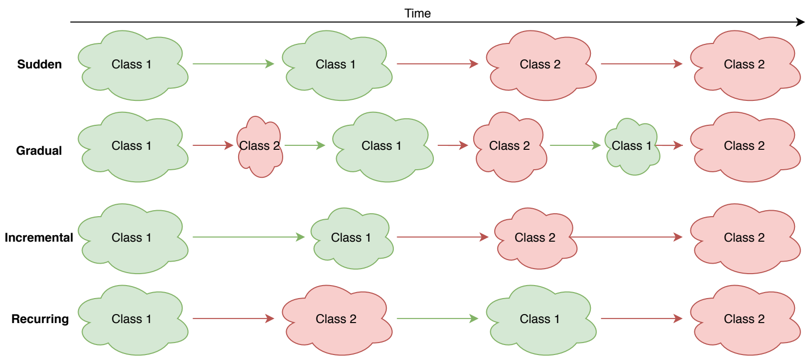


Figure 28: The four types of Concept Drift.

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 29 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.

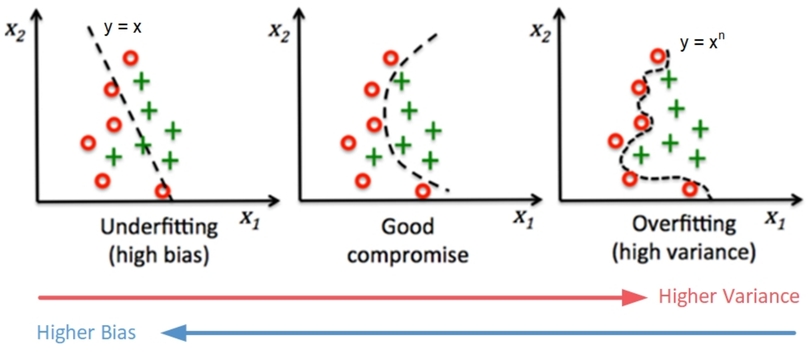


Figure 29: Overfitting and Underfitting: Challenge for a good bias-variance trade-off (Adapted fromTaschka (2015)).

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 29. The learning algorithms such as regression and deep learning typically learn using a cost or error function which is minimized such that the model with minimum error is determined. Regularisation uses a *regularisation* parameter λ to add the bias or penalty to the error function as the model complexity increases. Function 37 defines a simplified loss/cost function with regularisation.

$$minimize(costfunction, c(x) + \lambda(penaltyfunction)) \tag{37}$$

Function 37 includes two terms; i) the cost function, and ii) the penalty (regularization) function, where the penalty function is constrained to be less than or equal to a constant, t (Hastie et al., 2001).

To train the model, minimisation is done on both the cost and penalty functions. The cost function, c(x) depends on the training data whereas the regularisation function is independent of the data variable (x_n). Parameter λ is determined empirically or through cross validation, and it is used to control how to balance out the two terms in function 37 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 37 can be re-written as:

$$minimize(\sum_{i=1}^{n}(y_i - w_0 - \sum_{j=1}^{p}(x_i j.w_j))^2 + \lambda\sum_{j=1}^{p}(w_j^2)) \tag{38}$$

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

L1 Regularisation

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

$$minimize(\sum_{i=1}^{n}(y_i - w_0 - \sum_{j=1}^{p}(x_ij.w_j))^2 + \lambda \sum_{j=1}^{p}(w_j)) \tag{39}$$

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

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

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

Dropout

Dropout is another regularisation technique that is targeted for neural networks (NN) models and was proposed by Srivastava et al. (2014). The dropout techniques adds bias or noise to the NN model in order to prevent overfitting similar to the L1 and L2 regularisation techniques described earlier. In order to add this bias, the dropout technique removes nodes together with their input and output connections randomly during training. The Dropout technique is illustrated in Figure 30,31.

The random dropout action is performed during the training phase only, and it creates a different thinned NN (Figure 31) for each epoch. Therefore the minimisation, through back propagation, of the NN loss function is only applied to the thinned network, and the inactive

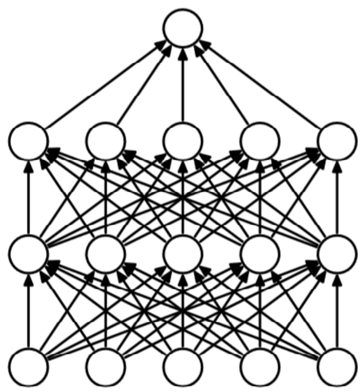


Figure 30: Standard Neural Net (Reproduced from Srivastava et al. (2014)).

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

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.

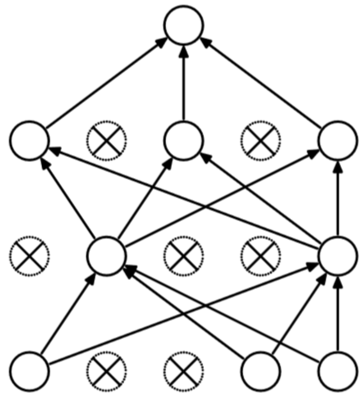


Figure 31: Thinned Neural Net after applying dropouteproduced from Srivastava et al. (2014)).

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

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

Early Stopping

Early Stopping is a method applied to NN where the training model is stopped before the training error is minimized (Sarle, 1995).Figure 32 illustrates a typical plot of the accuracy of model versus the epochs for the 'Training Data' and 'Test Data'. At each iteration the test data is used to evaluate out the model being trained. As can be noticed although the training accuracy continues to increase as cost function is minimized, the 'Test Data' accuracy start dropping after a certain epoch.

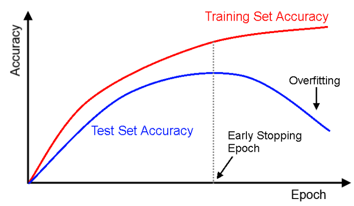


Figure 32: Training Data Accuary Vs Test Data Accuracy

Early Stopping determines the number of epochs a model is allowed to run by evaluating the *training* model after each epoch (or n epochs). The model is stopped if subsequent training results in lower performance. This marks the 'Early Stopping Epoch' as shown in Figure 32. Zur et al. (2009) showed that early stopping reduces the effect of overfitting but is it is not as effective as weight decay using L1 and L2 regularisation. Early Stopping is considered a form of regularisation method since it helps the model from overfitting.

Sample Selection Bias

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

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

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

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

False Information created by Selection Bias

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

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

How To Prevent Selection Bias

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

Stratified Sampling

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

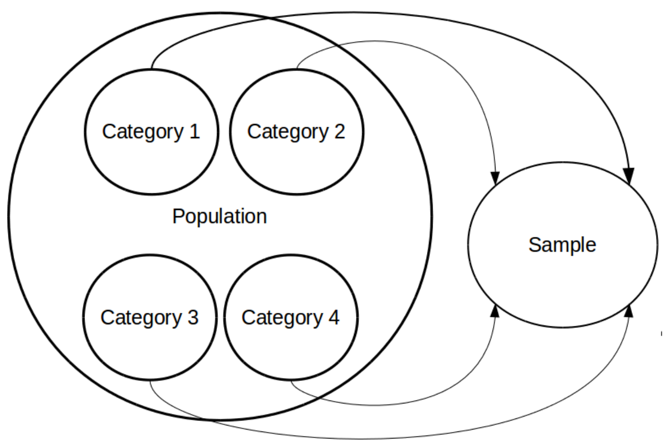


Figure 33: Stratified sampling: splitting the population into categories and selecting accordingly

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)

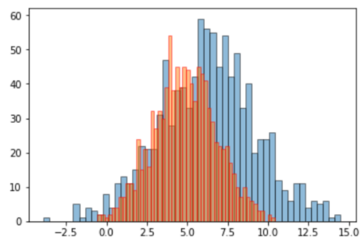


Figure 34: Comparing Distributions

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.

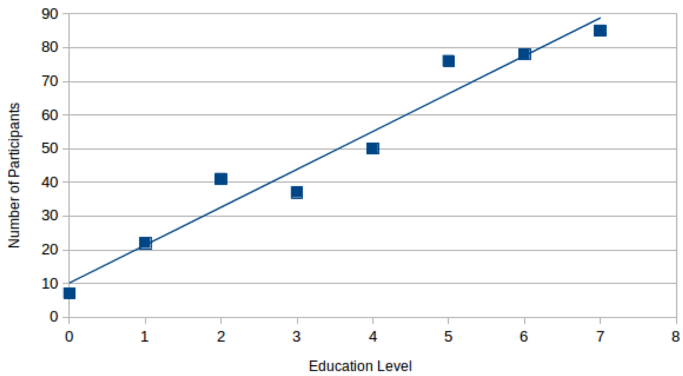


Figure 35: Two Step Estimator: Using regression models to determine correlations between participants and non-participants

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:

$$R(h) = \sum_{i=1}^{m} w_i c(h, z_i) \tag{40}$$

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.

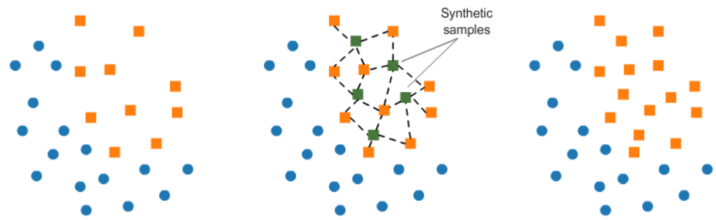


Figure 36: SMOTE: generating synthetic samples of the minority class

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).[3]

[3] 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 37), 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.

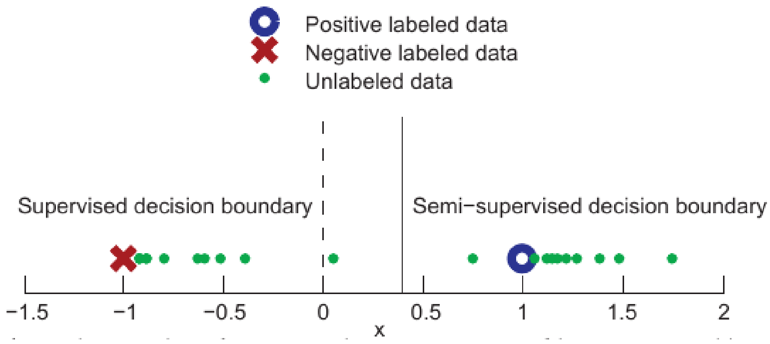


Figure 37: An example of the decision boundaries determined by a Supervised system and a Semi-Supervised system. Source: Zhu & Goldberg (2009)

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;

$$\{(x_i, y_i)\}_{i=1}^{l}, \{x_j\}_{j=l+1}^{l+u} \tag{41}$$

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

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

$$\{(x_i, y_i)\}_{i=1}^{l}, \{x_j\}_{j=l+1}^{l+u} \tag{42}$$

...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 $\{x_j\}_{j=l+1}^{l+u}$.

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.

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	*z13bgdvyluihfv11i22rgxwhuvabzz1os04*	*Zielimeek21*	*2013-11-28T21:49:00*	*I'm only checking the views*	*Ham*
Psy	*z13kxpqqssa0hlryd04cc1dxeyyngljjngk*	*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 8: Sample of four rows from the Psy dataset from the YouTube comment training dataset.

Table 9 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.* *If that is the case, two things should happen, none of which are directly related to the machine learning algorithm:* *- Maintain a blacklist of accounts that are probable spam (if a particular author often has flagged comments).* *- 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 9: 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 10: 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 11, 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 11: 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.

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).

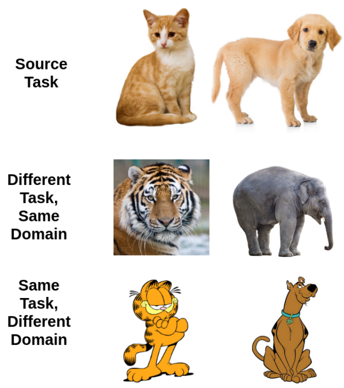


Figure 38: Transfer learning can occur across domains or across tasks. Images reproduced from commons.wikimedia.org (Tiger, Cartoon Cat), imdb.com (Cartoon Dog), pixabay.com (Cat), pnghunter.com (Elephant) and shutterstock.com (Dog).

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.

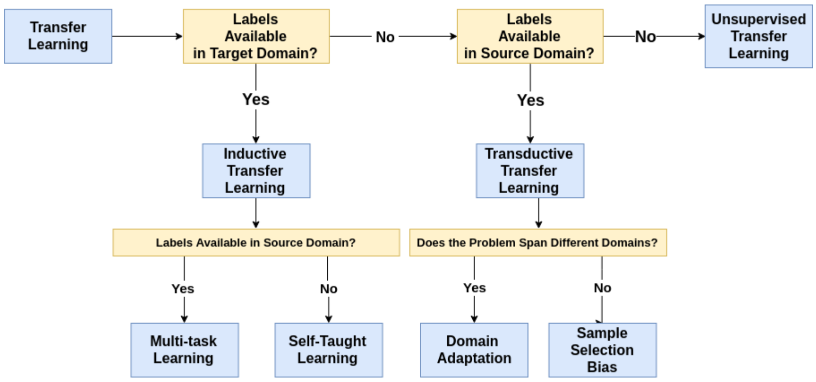


Figure 39: Transfer learning approaches vary depending on the availability of labeled data for the source and target tasks.

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 12: 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).

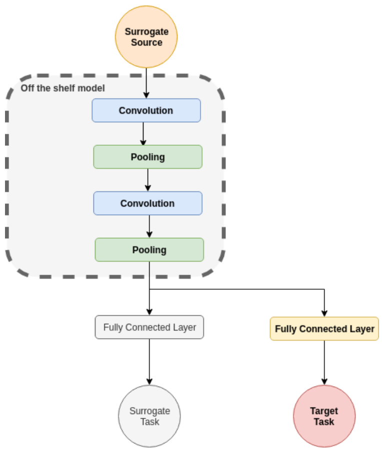


Figure 40: In off-the-shelf pre-trained models, we replace the final layer of the neural network.

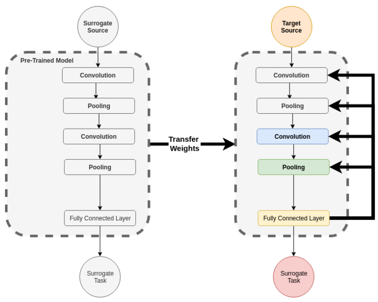


Figure 41: In Pre-trained model fine tuning, we use the source model to initialize the neural network, and fine-tune it using backpropagation.

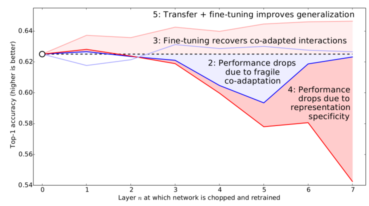


Figure 42: The performance gained form fine-tuning a neural network is only relative to the layer generalization and specificity of each distinct layer (Yosinski et al., 2014).

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).

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 13. 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 13: 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 13.

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 43 (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.

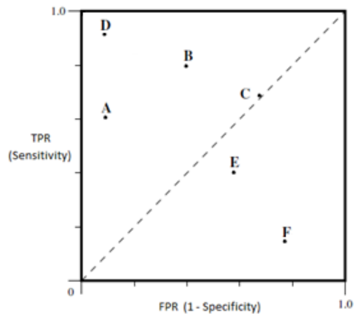


Figure 43: Basic ROC plot with five classifiers.

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 44, 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.

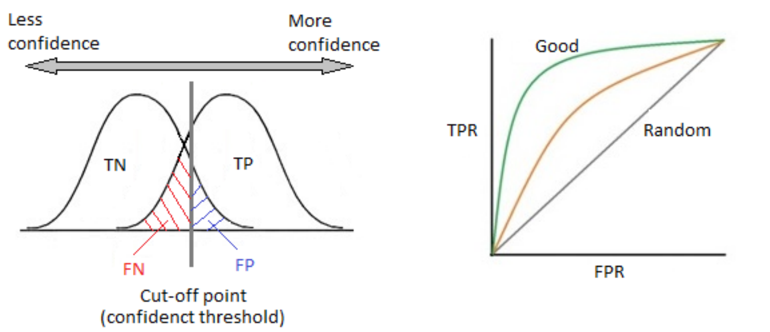


Figure 44: Left: Overlapping distributions with threshold. Right: ROC curves resulting from different thresholds.

The classifier that is able to increase the rate of detection while keeping the false alarm rate low is considered a good classifier. Figure 44 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 45.

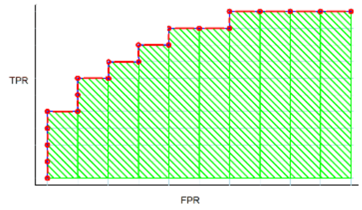


Figure 45: Area under the curve.

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 46 (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.

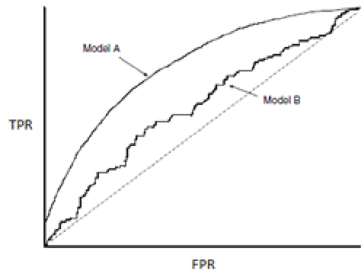


Figure 46: A comparison of two AUC curves.

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.

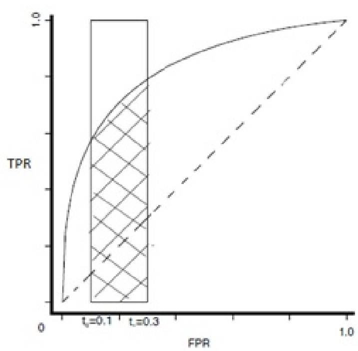


Figure 47: Illustration of an ROC curve and its partial AUC with t_0 = 0.1 and t_1 = 0.3.

Essentially the partial AUC is simply the area under the ROC curve between t_0 and t_1 (Figure 47) 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 43 (Dodd and Pepe, 2003).

$$AUC(t_0, t_1) = \int_{t_0}^{t_1} ROC(t)dt \tag{43}$$

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.

Bibliography

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

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

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

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

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

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

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.

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.

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.

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. 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.

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.

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.

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.

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.

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

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 Mar\'\ia 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.

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

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

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.

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

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

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

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.

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.

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

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

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.

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

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.

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.

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

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

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

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.

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

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 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.

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

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

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.

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

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.

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.

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.

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.

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

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

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.

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.

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

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

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.

Christopher Olah. Understanding lstm networks, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

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

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

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

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

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

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

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

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

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

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.

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. URL https://doi.org/10.1007/BF00116895.

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.

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.

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.

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.

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

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

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

Pranjal Strivastava. Essentials of deep learning : Introduction to long short term memory, 2017. URL https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/.

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.

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

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.

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

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.

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.

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

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

David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

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

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

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.

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

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

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