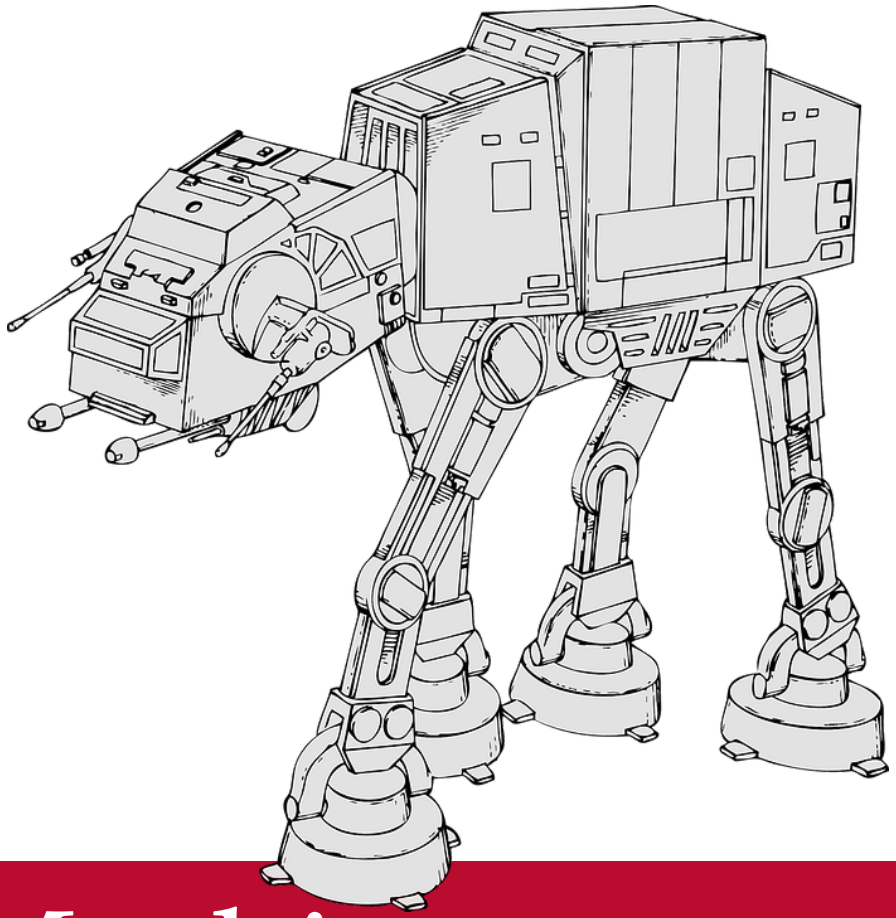
A Machine Learning Study Guide



Machine Learning Handbook

The Definitive Guide

ICS5110, class of 2018/9

L-Università ta' Malta

JEAN-PAUL EBEJER, DYLAN SEYCHELL, LARA MARIE DEMAJO, KEITH MINTOFFADD YOUR NAME TO THIS LIST

First printing, December 2018

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

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). Each of these approaches are further divided into different cross-validation methods, which are 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. 2a. 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).

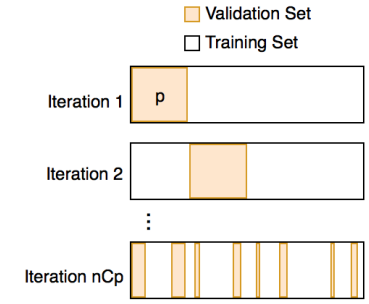


Figure 1: Exhaustive cross-validation methods: Leave-p-Out

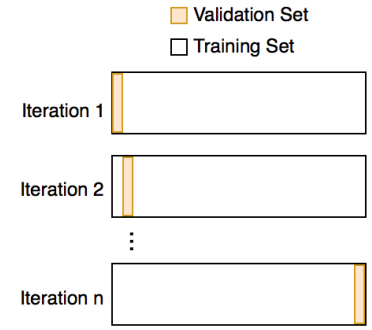


Figure 2: Exhaustive cross-validation methods: Leave-One-Out

Non-exhaustive cross-validation

- Holdout method
  This method randomly splits the original data set into two sets being the training set and the test set. Usually, the test set is smaller than the training set so that the algorithm has more data to train on. This method involves a single run and so must be used carefully to avoid misleading results. It is therefore sometimes not considered a CV method (Kohavi, 1995).

- k-fold
  This method randomly splits the original data set into k equally sized subsets, as shown in Fig. 3. 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).

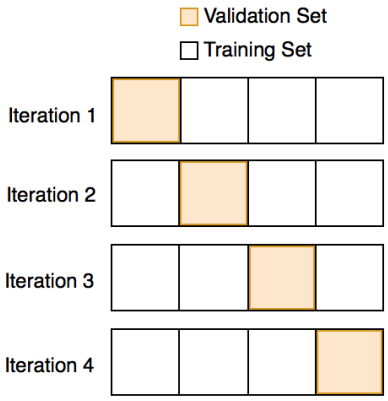


Figure 3: k-Fold Cross Validation where k=4

- 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. 1, where n is the total number of test samples, Y_i is the true value of the i^{th} instance and $\hat{Y}_i$ is the predicted value of the i^{th} instance.

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

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. 5a. 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. 5b, 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. 6, we can now start forming our 10 folds by taking 10% of each age-group bucket, randomly without replacement. Hence, we will end up with 10 subsets of 100 images that are equally distributed along all age-groups. With these subsets, we can estimate our model's accuracy with a lower bias-variance tradeoff. Since we are using 10-fold CV, we will train and test our model 10 times. For the first

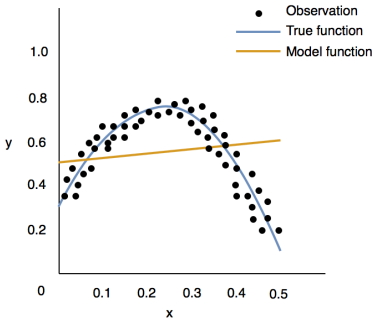


Figure 4: Underfitting

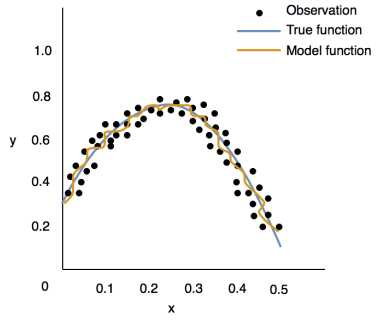


Figure 5: Overfitting

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

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. 3). 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.

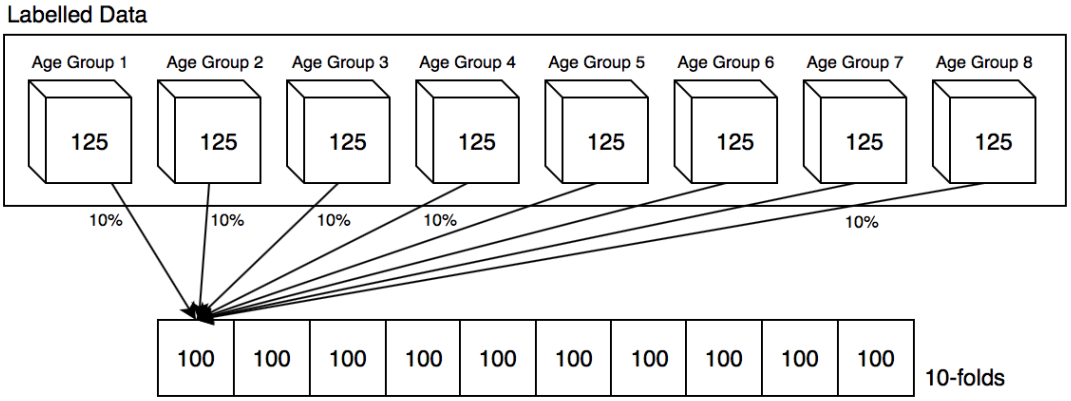


Figure 6: Stratified 10-fold cross-validation on 1000 labelled images of 8 different classes

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.

Activation Functions

"Neural networks were originally conceived 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." (Caterini, 2018)

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

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{2}$$

$$\frac{d}{d(x)} f(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases} \tag{3}$$

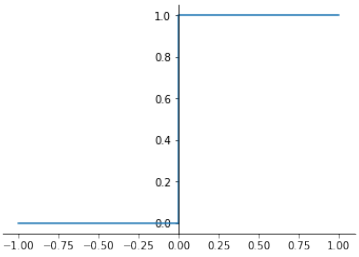


Figure 7: A graph of the step function.

The Heavside step function, represented by Eq. 2, is one of the simplest activation functions that can be used in a neural network. This activation 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 function was first used in a machine learning context in 1957 by Frank Rosenblatt in his seminal work describing the perceptron, the precursor to the modern day neural network (Rosenblatt, 1957).

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, represented by Eq. 3, is 0, 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

A linear activation 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. While a linear activation function could be used for multi-class problems, it can on be used on problems that are linearly separable. Linear functions can also run into problems with gradient descent algorithms, as the derivative of a linear function is a constant. 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

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{4}$$

$$\frac{d}{d(x)} f(x) = f(x)(1 - f(x)) \tag{5}$$

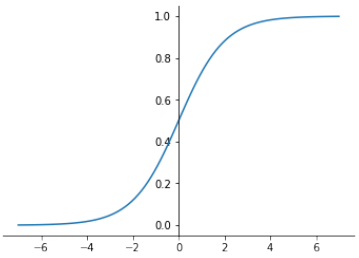


Figure 8: A graph of the sigmoid function.

The sigmoid function or logistic function, represented by Eq. 4, 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 Rummelhart, Hinton and Williams 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 (Rumelhart et al., 1986). The values produced by the step function are bound between 0 and 1, both not inclusive, which help manage the exploding gradient problem. The derivative of this function (Eq. 5) 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, 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.

Hyperbolic Tangent

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{6}$$

$$\frac{d}{d(x)} f(x) = 1 - f(x)^2. \tag{7}$$

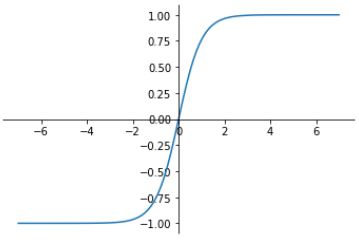


Figure 9: A graph of the hyperbolic tangent (*tanh*) function.

The hyperbolic tangent (*tanh*) function, represented by Eq. 6, is another activation function that is sometimes used instead of sigmoid.

The *tanh* function has the same characteristics of the sigmoid function mentioned above. In fact the *tanh* function is plotted, one can observe that it is simply a scaled version of the sigmoid function. As a result of this scaling, the tanh function has a steeper gradient in towards the origin, and it returns values between -1 and 1. The derivative of the hyperbolic tangent function is represented by Eq. 7.

Nowadays with the rise of deep learning, these functions are becoming less commonly used. Xavier Glorot and Yoshua Bengio studied in detail the effects of the sigmoid and tanh activation functions. They note how the sigmoid function in particular is not well suited for deep networks with random initialization (Glorot and Bengio, 2010).

Rectified Linear Unit

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{8}$$

$$\frac{d}{d(x)} f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{9}$$

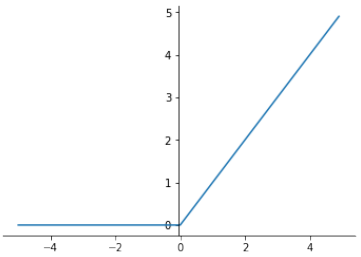


Figure 10: A graph of the ReLU function.

The ReLU function, represented by Eq. 8, 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 function due to its simplicity performance, and suitability to networks with many layers. Another benefit of the ReLU function is that it produces sparse activations (not all nodes in the network are activated) unlike the sigmoid or hyperbolic tangent functions.

The ReLU function has been used in many neural network models to improve their performance. Naid and Hinton use ReLU to improve the performance of Restricted Boltzmann Machines in object recognition (Nair and Hinton, 2010). In 2012, a breakthrough Convolutional Neural Network (CNN) architecture called AlexNet pioneered the use of the ReLU activation function together with dropout layers to minimise over fitting in CNNs. (Krizhevsky et al., 2017).

Unfortunately, because the gradient of the function for inputs that are negative is 0 (Eq. 9), 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 o for negative inputs, the leaky ReLU return a very small value such as $0.01x$. However, researchers at the University of Stanford 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 (Maas et al., 2013).

Bibliography

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

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

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

Yoshua Bengio, Patrice. Simard, and Paolo. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181.

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.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

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

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.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. ISSN 1557-7317.

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.

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.

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

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

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.

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

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

Index