Applied statistical modelling to guide the control of introduced mammalian predators in the Murchison Mountains (Fiordland, New Zealand).

A thesis

submitted in partial fulfilment

of the requirements for the Degree of

Master of Science

at

Lincoln University

by

Benjamin Hayden Wiseman

Lincoln University

2016

Abstract of a thesis submitted in partial fulfilment of the requirements for the Degree of Master of Science.

Applied statistical modelling to guide the control of introduced mammalian predators in the Murchison Mountains (Fiordland, New Zealand).

by

Benjamin Hayden Wiseman

Since the rediscovery of *Porphyrio hochstetteri* (takahē), in the Murchison Mountains (Fiordland, New Zealand) in 1949, there have been continued efforts to protect the species. Takahē numbers have been increasing from a minimum of 120 individuals in 1981, peaking in 2006 when 297 takahē were recorded (168 in the Murchison Mountains). Between 2002 and 2008, an experimental trapping programme was implemented within the Murchison Mountains targeting *Mustela erminea* (Linnaeus, 1758) (stoats) and *Rattus rattus* (Linnaeus, 1758) (ship/black rats). Approximately 40% of the Murchison Mountains was assigned for intensified predator trapping while the remaining area was left as a control. However, during the period of intensified trapping a stoat plague of unexpected magnitude occurred within the Special Takahē Area (STA); nearly half of the remaining takahē in the Murchison Mountains were killed. Thus, after introducing predator control in the area, the highest levels of observed predation occurred. In the previous decades of takahē monitoring and management with no stoat control, such high predation (40%) was never recorded.

My MSc research primarily aims to elucidate the mechanisms underlying the 2007-2008 stoat plague and develop improved methods and tools for conservation managers. To facilitate the practical application of predictive models for management purposes, I devolop two fully featured software packages with a graphic user interfaces (GUIs) enabling quick, customisable approach to 1) generating artificial neural networks, and 2) a GIS platform to apply neural networks to an area. The predictive models I generate, combined with the software and improved CPUE metrics, can be used to optimise predator management in the Murchison Mountains.

Keywords: Catch per unit effort, neural networks, GIS, application development, conservation management, predator control, machine learning, stoat, rat, takahē,

Acknowledgements

My supervisors James Ross and Des Smith, my advisor Will Gittoes, Adrian Patterson, Andrey Digsby, Sanjay Thakur, Kerri-Anne Edge, Gerad Hill, Elaine Murphey, Department of Conservation TeAnau, the Takahe recovery team, Lincoln University Spatial Ecology Group, my wife Emily Fountain, my Friends Vikki Smith, Hamish Patrick, Arsalan Emami Khoyi, Andrew Pugh, Senait Senay, the lord Bob Ross, and my son Piglet.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and literature review

1.1 Conservation of the Takahe in the Murchison mountains

Once widely distributed across the South Island, takahē numbers have dramatically declined due to human actions until they were thought extinct by the end of the 19th century (Trewick & Worthy, 2001). After the rediscovery of the takahē in the Murchison mountains in 1948, a 518 km^2 'Special Takahē Area' was established for the species' protection (Ballance, 2001) (**Figure 1-1**). Takahē are listed as nationally critical within the Department of Conservation's (DoC) classification system (Hitchmough, Bull, & Cromarty, 2007) with approximately 275 birds surviving in total. Of the remaining takahē, approximately 110 individuals reside in the Murchison Mountains with the remainder on offshore islands and in the Burwood breeding facility (Hegg, Greaves, Maxwell, MacKenzie, & Jamieson, 2012).

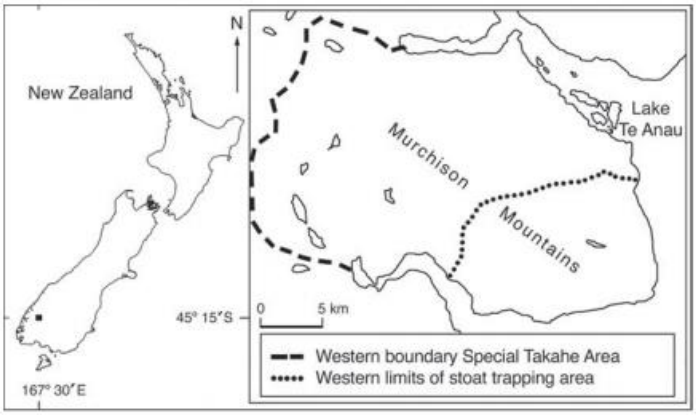


Figure 1-1 Map of the Special Takahē Area with the 2002 - 2008 experimental trapping boundary shown (dotted line). Figure from Hegg et al 2012.

Within the Special Takahē Area, various conservation management strategies have been implemented since 1949 (Hegg et al., 2012). Early conservation efforts, headed by the then New Zealand Wildlife Service, were primarily focused on natural history observations so as to leave the birds as undisturbed as possible (Lee & Jamieson, 2001). Observations of takahē numbers in the 1960s identified a steady population decline; by the early 1970s the continued survival of the takahē without human intervention seemed doubtful (Mills & Lavers, 1974). Between 1970 and 1980 research continued into the takahē's habitat requirements and environmental monitoring programmes were established. Data collected in the 1970s to early 1980s identified significant resource competition from *Cervus elaphus* (red deer) and some evidence of stoat predation on chicks. Stoat trapping was initiated in some valleys and alpine basins with the intention of quantifying the effect of stoat predation on takahē; the results were equivocal (Lee & Jamieson, 2001). Deer control intensified in 1976 with the introduction of helicopter shooting, deer numbers

then began a steady decline. Following the first takahē conservation symposium in 1978, an active management plan was developed for the takahē. Despite these ongoing management efforts the takahē numbers continued to decline until 1981 when they reached a minimum of 120 birds (Maxwell, 2001). Intensified management from 1980 including deer culling, increased predator control, captive rearing, and nest management has greatly contributed to increasing the takahē population, with a maximum of 297 takahē (168 in the Special Takahē Area) recorded in 2006 (Hegg, MacKenzie, & Jamieson, 2013).

Between 2002 and 2008, an experimental trapping programme was implemented within the Murchison Mountains to assess the effect of trapping on adult takahē survival; approximately 40% of the Special Takahē Area was designated for trapping while the remaining western area was left as an experimental control (Hegg et al., 2012). During the experimental period, in the 2007-2008 season, an unprecedented spike in takahē predation was recorded, presumably driven by prey switching (by stoats) following an extreme stoat plague (Hegg *et al*, 2012); only 54% of adult takahē in the area survived. The spike in takahē predation (46% mortality) reduced the takahe in the Murchison Mountains to the lowest level ever recorded (Greaves *et al*, 2008 unpublished). In response to the loss of nearly half of the known takahē in Fiordland, the Department of Conservation expanded stoat and rat trapping efforts into the control area. Hegg *et al* (2012) speculated that the stoat plague was driven by unusually high rat numbers which were in turn driven by a particularly heavy beech (*Nothofagus*) seed mast event.

1.2 Hyper predation and the 2007-2008 stoat plague

Unlike other native forest birds in the Murchison Mountains, adult takahē are unlikely to be directly threatened by rats. Rats however serve as a valuable food source for stoats which may increase the reproductive success of stoats, although the relationship between rat numbers and stoat abundance can be ambiguous (Elaine C. Murphy, Clapperton, Bradfield, & Speed, 1998). While the relationship between *Mus musculus* (mouse) population growth and increased stoat predation is well documented within the context of beech seed mast effects (first doccumented by King, 1983), the potential for brief stoat population plagues to occur after rat population outbreaks is scarce in the literature.

Recently, Ruscoe et al. (2011) inferred no significant increase in rats occurred following stoat eradication, however their results may be influenced by the relatively short time span of their study combined with mark-recapture to infer rat population sizes. Additionally, the study of Ruscoe *et al* (2011) was performed in podocarp forest rather than the predominantly *Nothofagus* sp. habitat in the Murchison Mountains; the ecological interactions between rats and stoats may substantially differ between podocarp and beech habitats (Des Smith 2013, pers comm, Wildland Consultants). Furthermore, as identified by Rayner, Hauber, Imber, Stamp, and Clout (2007), the effects of mesopredator release are prone to substantial spatial heterogeneity – as the experimental design of Ruscoe et al. (2011) tried to minimise heterogeneity among their sample sites, their generalisations may not be valid in all habitat gradients. The result of Ruscoe et al. (2011) was also contrary to existing literature which supports a release in the populations of rodents and other mammalian taxa following control operations targeted toward apex predators (for example: Blackwell, Potter, McLennan, & Minot, 2003; Caut et al., 2007; Courchamp, Langlais, & Sugihara, 1999, 2000; Le Corre, 2008; Ritchie & Johnson, 2009; Roemer, Coonan, Garcelon, Bascompte, & Laughrin, 2001; Smith & Quin, 1996).

A possible explanation for the magnitude of the 2007-2008 predation levels is that by culling the apex predator (stoats) may cause species (rats in this case) to increase in number in the absence of top-down control. Therefore, when conditions are right, rat numbers can increase to higher levels higher than previously possible resulting in sufficient biomass to fuel a latent increase in stoat fecundity, i.e. the hyper predation hypothesis (Smith & Quin, 1996).

Alternatively, stoat predation on birdlife caused by a prey switch following rat control has been documented (for example: Murphy & Bradfield, 1992; Murphy et al., 1998); on the surface, a prey switch explanation could easily explain the 2007-2008 takahē predation event. Contrary to what

would be expected due to a prey switching response by stoats, the observed takahē predation was significantly lower in area A (with trapping) than in area B (without trapping). Such a discrepancy between observed and expected predation rates could however be explained by an interaction of a stoat population increase and a prey switch response mediated by stoats emigrating from area A to area B. For example, within the trapped area (A) stoat numbers could have increased due to increased rat numbers (via a mesopredator release effect), and due to the large home range of stoats (Alterio, 1998), an overabundance of stoats may have emigrated into the un-trapped area (B). If there was no mesopredator release effect to increase rat numbers in area B, a discrepancy between the lower numbers of rats and the influx of stoats from area A could cause stoats to switch prey; adding to such an interaction, the lack of stoat control in area B could have further skewed the distribution of predation between areas A and B.

Exploratory data analysis of the raw trapping data, despite the inherent noise, indicates that baseline and mast-peak rat captures have been increasing since the commencement of the intensified stoat culling regime in 2002 (**Figure 1-2**). However, contrary to the speculation of Hegg *et al* (2012), rat numbers in the 2007 mast appear to be lower than in the 2011 mast event. When breaking the data down by season and financial year it is possible to see that capture probability of stoats rapidly increases in the summer of the 2006 financial (starting in June) year and overtakes the capture probability for rats by the winter of 2007 financial year, remaining in overabundance until the 2008 financial year; the prolonged overabundance of stoats coincides with no increase in trap checks (**Figure 1-3**). Furthermore, when observing the relative performance (yearly conditional probabilities, **Figure 1-4**) of trap success for rats and stoats, it is possible to observe a marked increase in stoat captures between 2006 and 2007 in traps that otherwise had low capture probilities while in rats, there is no specific uptake to be seen, only a dramatic crash in the success of traps that were effective in 2007. Taken together, the raw data suggests that stoats underwent a prey switch following a post-mast crash in the rat population which was compounded by an

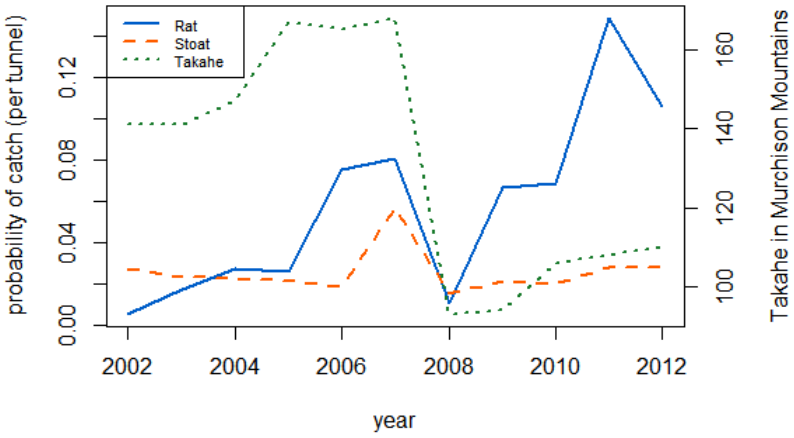


Figure 1-2 Marginal capture probabilities for rats and stoats (left axis) and total takahē numbers in the Murchison Mountains between 2002 and 2012

insufficient trapping effort combined with an expansion of stoats into areas they do not otherwise abound (as evidenced by increased capture probabilities in previously unsuccessful traps).

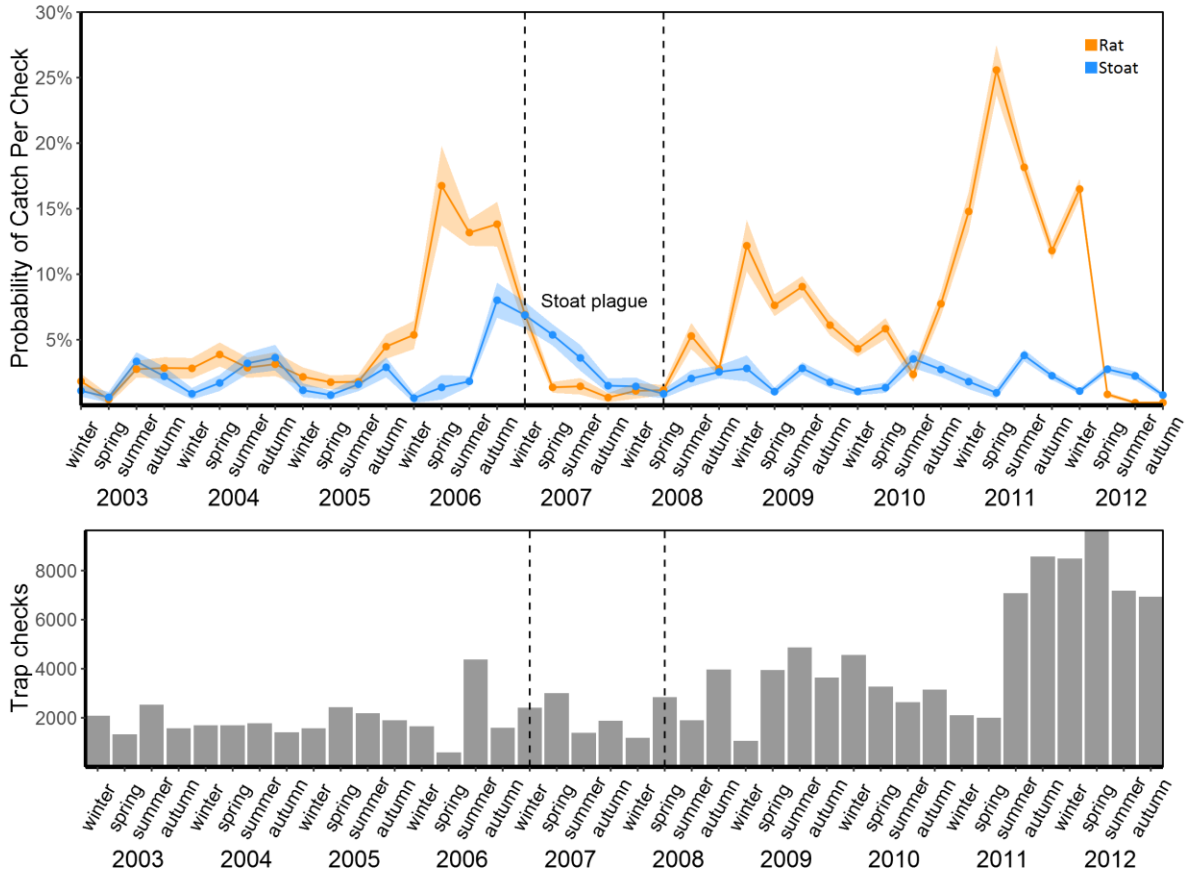


Figure 1-3 Capture probabilities per trap checked (top) and the total number of traps checked (bottom) for rats and stoats by fiscal year (beginning June 1) per season. Stoat plague, as shown by dashed lines, is set as per Heg et al (2012).

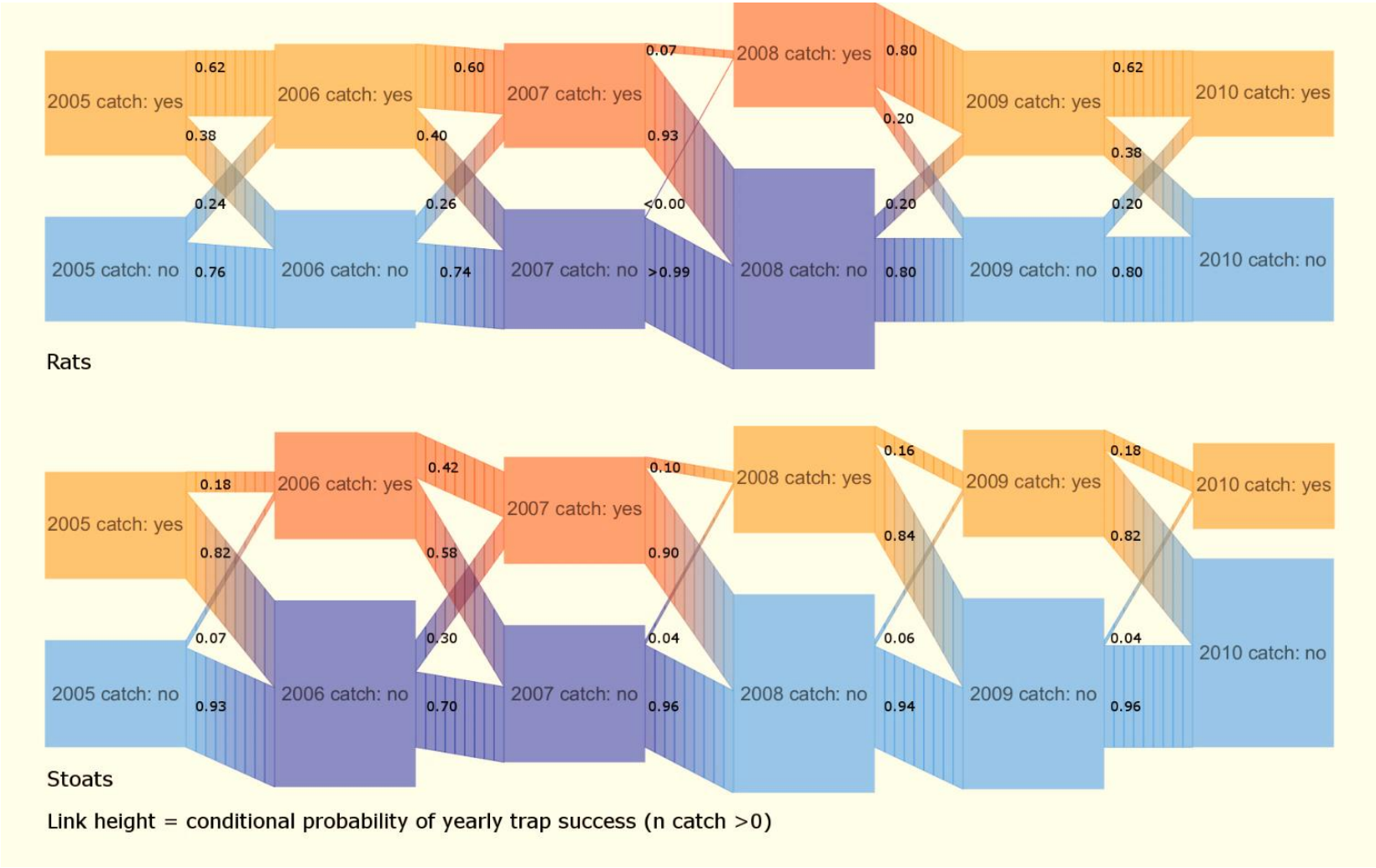


Figure 1-4 Sankey diagram of conditional probability (on previous year) flow between traps that were successful and unsuccessful in t-1 and t for rats (top) and stoats (bottom). Link widths are probabilities are read as: p(capture outcome in time t | capture outcome in time t-1). Anomalous years are highlighted for convienience.

1.3 Rationelle and Objectives

1.3.1 Project Rationelle

There exists a need to more fully understand the causal factors underlying the 2007-2008 stoat plague. Moreover, as part of the Department of Conservation takahē management plan, analysis of the trapping dataset from the Murchison Mountains is a priority as well as creating tools for the continued management in the Murchison Mountains. The Department of Conservation asked that I: 1) elucidate the underlying mechanisms of the 2007-2008 stoat plague, specifically investigating whether stoat control has inadvertently caused hyper predation; 2) create geographically referenced predictive models for stoat and rat eruptions and reinvasions; 3) summarise over ten years of trapping data for their internal usage; 4) assess the merits of catch per unit effort (CPUE) metrics within the Special Takahē Area; and 5) Generate functional statistical models for predicting trapping efficiency which can inform the Department of Conservation's management decisions.

I use the existing dataset rather than performing more data collection via field trials. Given that there already exists a substantial dataset, and collecting data on stoats is notoriously difficult (Des Smith 2013, pers comm, Wildland Consultants), the choice to analyse the existing dataset the logical course of action. While extensive statistical modelling can result in over parameterisation and circular inference, especially where complex models are required, carefully separating data into training and validation sets can give empirical measures of model accuracy (Lankin Vega, 2002; Steyerberg et al., 2001; Worner, Lankin, Samarasinghe, & Teulon, 2002). Although simple models tend to be favoured in ecology, possibly due to a common belief that they are more robust, their ability to make accurate predictions for any given system is generally lacking (Evans et al., 2013). My research is oriented toward solving a specific problem so I will aim to construct complex models insofar as they are supported by their validation success or information theoretic criterion. If DOC wish to generalise the model into areas for which highly specific predictors are as applicable (seed count data from Takahē valley for instance), I encourage them to use the software I have created to create fresh models. As highlighted by Evans *et al.* (2013), because complex models tend to represent many nested simple models, beginning with a complex model and simplifying where some parameters lack information (where they are uninstantiated as described by Orzack and Sober (1993), they can allow greater generality than simple models alone are able to achieve.

Sophisticated mathematical models are critical to many aspects of conservation and wildlife management. Whether optimising decision making procedures for culling and sustainable harvesting operations (Bunnefeld, Hoshino, & Milner-Gulland, 2011), forecasting the effects of land use on

species distributions (Rodríguez, Brotons, Bustamante, & Seoane, 2007), or optimising management strategies for the control of invasive predators such as feral cats (Loyd & DeVore, 2010). As the development of adequate predictive models for optimal trapping is the core objective of my thesis, and there exists a large active database, it was a more effective use of time for me to model the

I employ artificial neural networks (ANNs), machine learning algorithms that use functional artificial intelligence approaches to learn patterns within complex data structures. Generally, ANNs provide superior predictive accuracy for new data compared to standard statistical techniques (Lankin Vega, 2002; Razi & Athappilly, 2005; Worner et al., 2002). In addition to their predictive power, the advanced training procedure of artificial neural networks allows them to easily deal with common problems such as non-linearity, noisy data, non-independence, and genuine complex interactions; they are effectively free from apriori assumptions (Basheer & Hajmeer, 2000; Lankin Vega, 2002). I create neural networks via the NeuroFriendly application (**Chapter 4**) which I have built and optimised for datasets such as the Murchison Mountains trapping data. In addition to being free from prior assumptions, artificial neural networks can theoretically identify the conditional probabilities of catching one target given that the trap is available via decision nodes. Such pattern recognition ability is the strength of artificial neural networks; they possess an ability to "learn" that GLMs lack.

As demonstrated by Lankin Vega (2002) and Tan, Özesmi, Beklioglu, Per, and Kurt (2006), assessing a model's explanatory power by testing the model against the data that were used to create it can result in misleading inferences about the model's real-world predictive power. To avoid such overestimations of model fit, in the absence of AICc values, I test models against validation sets – portions of data not used in model creation. By using validation sets, I can empirically measure the ability of any given model, resulting in an R^2 value that legitimately represents the model's ability to explain variation in data due to its inputs and to therefore generalise to new data.

1.3.2 Research Objectives

- **To create a better CPUE metric than C/100tn.** Using catches per 100 trap nights as a measure for capture output per unit of effort input is sub-optimal as it is unable to differentially weight zero catch records. Here, I aim to create a similar effort-adjusted measure of catch data that can be more effectively used in predictive modelling (Chapter 2).
- **To create predictive models for forecasting stoat and rat trapping success.** Accurate predictive models are the ultimate goal of my MSc research. The environmental variables that can be modelled include: trapping effort, trap density, habitat, climatic conditions, seedfall, the type of trap box used, and other spatial variables such as the distance from freshwater (Chapter 4).
- **To create user-friendly software to facilitate the rapid application of spatiotemporal prediction models with a minimal time investment by DoC staff.** The intent of developing software tools is to allow predictive models to be applied in practise (Chapter 5).

Chapter 2

Residual trap success and spatial smoothing: techniques for measuring and modelling catch per unit effort on zero-inflated capture data with high levels of micro-site variation.

2.1 Abstract

In this chapter I develop the use of the residual trap success (RTS) index in concert with spatial smoothing as a technique to measure and model CPUE that is demonstrably more powerful than the traditionally used raw catch per 100 trap nights (C/100TN). Although a useful descriptive metric for trap success, C/100TN suffers from an inability to differentiate between traps with zero captures that are performing poorly and traps with zero captures that are performing as expected given the amount of time the trap was active. When traps that are performing poorly are indistinguishable from traps that are performing as would be expected, statistical models struggle to fit the data and are consequentially less accurate. Where there exists micro-site variability for which there are no suitable micro-site predictors, statistical models will suffer from using measurements and predictors of different spatial scales, here I show that, analogous to smoothing time series data, applying a spatial smoother can greatly increase model accuracy. As an alternative to C/100TN I propose using the residual trap success (RTS) index in conjunction with spatial smoothing to alleviate zero inflation and high micro-site variation. Using the C/100TN, RTS, and their respective spatial moving averages to model *Mustela erminea* (stoat) and *Rattus rattus* (rat) captures in the Murchison Mountains it can be shown that RTS and spatial smoothing yield improved modelling capability over C/100TN. The combination of spatial smoothing and RTS increase model fit (R^2) for stoats from 4% to 63% and from 57% to 88% for rats.

2.1.1 Introduction

Catches per trap night has been employed as an index of animal abundance has for over a century since Grinnell (Forsyth, Link, Webster, Nugent, & Warburton, 2005; 1914) first employed trap nights to "show in statistical form the association preference of four species of pocket mouse" (*Perognathus spp*.). Since at least 1931, the number of individuals caught per trap night has, for convenience, been multiplied by 100 (Carnes, 1931), and had corrections applied for trap availability given that sprung traps can be empty or capture non-target species (Nelson & Clark, 1973). Presently, catch per 100 trap nights (C/100TN) is arguably one of the most common metrics in New Zealand wildlife management. Despite the popularity of C/100TN (eg. Adams, 1984; Dowding & Murphy, 1994; Drever & Lewis, 2004), the metric is not without flaws. The first flaw, originally highlighted by Dice (1931), is that the rate of trap catches over time is unlikely to be linear, for example in Forsyth et al. (2005), possum capture probabilities decreases with increased abundance. Given a lack of linearity, one must question the (inferred) simplified extrapolation of multiplying the average catch per trap night by 100, especially in cases where the trapping sample rarely exceeds 100 trap nights. Although basic multiplication will not affect statistical models (beyond trivially changing parameter estimates), it should be considered whether a possible reduction in correctness is justified.

Concerns about linear assumptions aside, the major issue with C/100TN is that being a division based metric, it is unable to differentiate between traps with no captures but varying trap nights, resulting in a mislabelled dependant variable. For example, if a trap has caught nothing after one trap night then (0/1)*100 = 0 C/100TN; if a trap has caught nothing after 100 trap nights (0/100)*100 = 0 C/100TN. A trap that has caught nothing after 100 trap nights is almost certainly performing worse than a trap that simply may not have had adequate opportunity to catch anything after a single trap night. The result of low performing and typical (but short running) traps receiving the same score is that the data are fundamentally mislabelled. Mislabelled data means a statistical model will be unable to correctly predict at least some observations – consequentially model performance is compromised (e.g Long & Servedio, 2010). Some of the problems arising from zero captures can be thought of as mislabelled "dirty data" which we should seek to "cleanse" with some pre-processing (see Maletic & Marcus, 2005).

Here, I present a simple method of scoring trap performance that allows zero capture traps to be scored more accurately for modelling purposes. By simply taking the residual of accumulated captures over time or the residual trap success (RTS), the problem of zero captures is avoided. When

the residual of captures over time is used, traps with average performance, including traps that have not been operational long enough to successfully catch a target, receive a score of approximately zero. Traps which perform better than average receive a positive score and *vice versa* for poorly performing traps (**Figure 2-1**). In the previous example of two traps neither of which had caught anything after one and one hundred trap nights respectively, using C/100TN, both traps receive a score of zero. Using RTS the former receives a score of approximately zero while the latter receives some negative value (**Equation 2-1, 2-1a**).

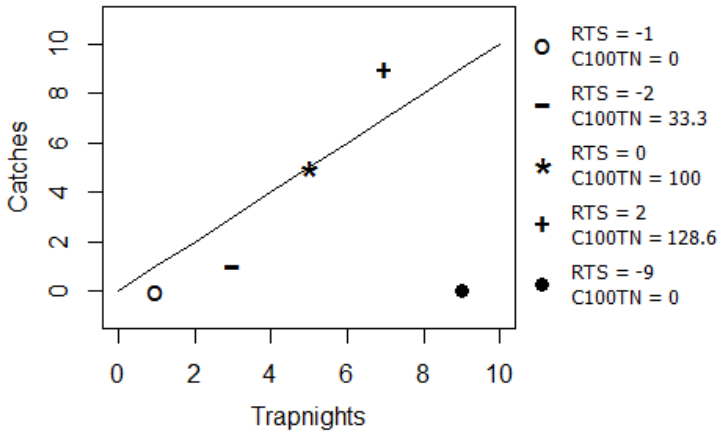


Figure 2-1 Hypothetical comparison of RTS and C/100TN scores for traps (assuming an average accumulation rate of one catch per trap night).

Equation 2-1. α is the model intercept, β is the parameter estimate for capture accumulation over time derived from the entire trap set.

$$C100TN_1 = \frac{0}{1} * 100 = 0; \;\; RTS_0 = 0 - (\alpha + 1\beta) \approx 0$$

Equation 2-1a. Where 0 captures are recorded over a longer time, C/100TN receives a 0-value, RTS receives a negative value

$$C100TN_{100} = \frac{0}{100} * 100 = 0; \;\; RTS_1 = 0 - (\alpha + 100\beta) \; < 0$$

The interpretation of RTS can be intuitively thought of as a positive number representing a trap that performs better than the average trap in a given set while a negative number represents worse than average trap success. For practical purposes, RTS can be considered a micro-site response variable at the scale of the individual trap which can be standardised to compare scores from different datasets. As a descriptive statistic for an entire set of traps, RTS is less desirable than C/100TN as it is the residual variation of individual data points which will have an average of 0 for the set. For describing subsets and individual traps, a residual trap success should be used given its ability to score zero-catch traps. For analysis, in particular machine learning or boosting algorithms where mislabelled data have a larger impact, RTS is more desirable than C/100TN. Both C/100TN and RTS can be thought of as indices of capture success per unit effort; C/100TN shows the raw value for a trap captures over time and RTS shows the relative success of traps given a general catch accumulation rate. Given that RTS is always calculated at the scale of individual traps, consideration must be given

to 1) the disparity between the scale of available predictors and the response variable, and 2) the potential for local heterogeneity caused by trap competition and the binary (and sparse) nature of trap catches. To address these considerations the effect of smoothing the spatial data (analogous to smoothing time series data) is also explored in this chapter. It should be noted that traditional binary models failed to adequately fit the data and did not yield any significant predictive ability.

2.2 Methods

2.2.1 Data Preparation

This chapter uses rat and stoat trapping data collected by the Department of Conservation's Takahē recovery team in the Murchison Mountains (Fiordland). Data were collected between 2002 and 2012. Two control areas were established within the Murchison Mountains, A and B, with predator control being absent in area B until after a stoat plague in 2007 (Hegg et al., 2012). To minimise zero-inflation the data used to perform comparisons were taken as the end-point data using total captures to date over the total trapnights to date to calculate C/100TN.

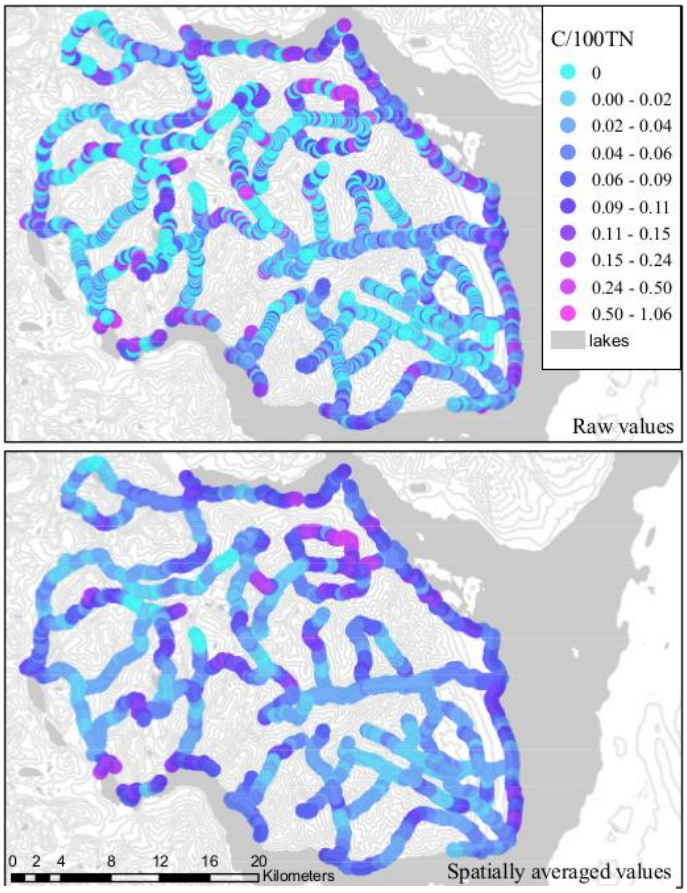


Figure 2-2 Stoat captures per 100 trap nights (C/100TN) before (top) and after (bottom) spatial smoothing within a 500 meter radius within the Murchison Mountains. Lakes and contour lines added for reference.

Residual trap success was calculated as the residual of the number of rats and stoats caught at each tunnel (at the most recent observation) given a linear model of catch accumulation over time for all traps (**Figure 2-3**). To account for a potential change in the abundances of rats and stoats following management, RTS scores were calculated on separate models for control areas A and B with a sub model for tunnels with high leverage. All calculations were performed using linear model function with a zero-intercept enforced in the base package of R version 3.1.3 (R Core Team, 2014). Trap metrics were spatially smoothed (zonal average) (e.g. **Figure 2-2**) within 500m and 1000m radii of each trap in Python via the ArcPY

10.2 API (ESRI, 2014). Spatial averaging was preferred to alternatives such as kernel density so as to: 1) maintain a simple moving average, and 2) keep values within the range of the original data to keep interpretation simple.

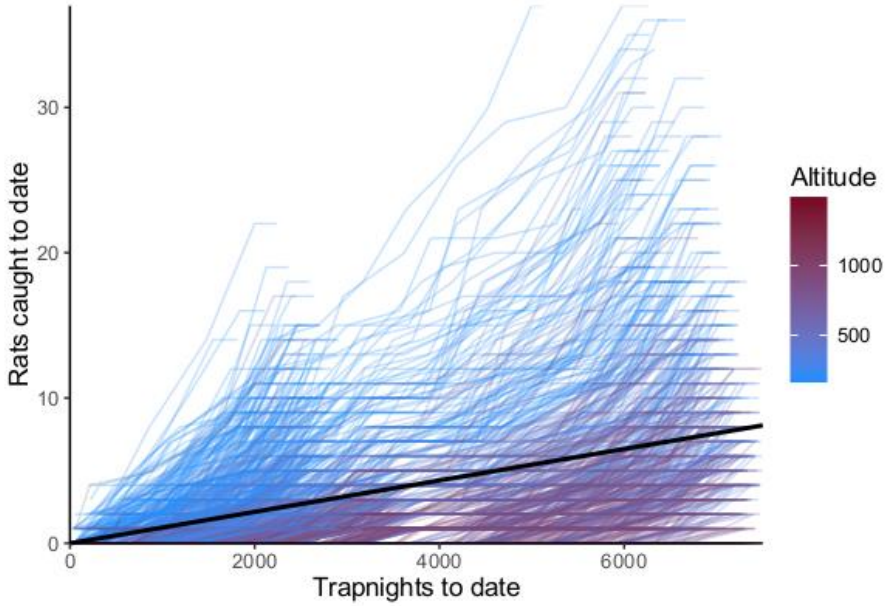


Figure 2-3 Cumulative rat captures (per tunnel) against cumulative trapnights (combined for tunnels with two traps) with a linear regression curve from which RTS may be calculated. Colourisation is by trap altitude as an example of the how RTS can be applied to modelling and classification tasks

2.2.2 Evaluation of metrics

The effects of changing from C/100TN, to residual trap success indexes and spatially smoothing data on model accuracy were assessed for both generalized linear effect models (GLM) and artificial neural network (ANN) models using predictor variables, identified by Christie *et al* (2006). Tests of predictive power for neural network models were performed on a randomly separated validation dataset. Base-line models were created for the raw C/100TN data of rats and stoats. Predictors from the C/100TN models were conserved for subsequent model comparisons, although the models are not necessarily optimised, using the same set of independent variables yield the clearest comparisons. Furthermore, by using predictors found to be most effective with C/100TN as a response in conjunction with the cumulative capture numbers after <= 10 years (therefore reducing the rate of zero-catch traps), comparisons were biased in favour of C/100TN. By swaying comparative analyses against RTS, it is hoped that RTS can be more thoroughly tested.

Calculation of GLMs was performed in R 3.1.3 via the base package using a Gaussian error distribution (given the continuous nature of C/100tn data). To elucidate the effect of varying degrees of zero-inflation on the relative performance of RTS, a bootstrapping simulation was performed resampling data from zero- and non-zero subsets of the data; 300 replicates were used at 50 levels of zero-inflation. Artificial neural networks were created for raw and spatially smoothed C/100TN, and RTS, data in NeuroFriendly (**Chapter 4**) using five node groups of three hidden neurons with an $N - 3[x5] - 1$ (**Figure 2-4**) topology where *N* is the number of model input parameters for 20,000 epochs with bootstrap-boost resampling (**Chapter 3**) in separate runs for rats and stoats.

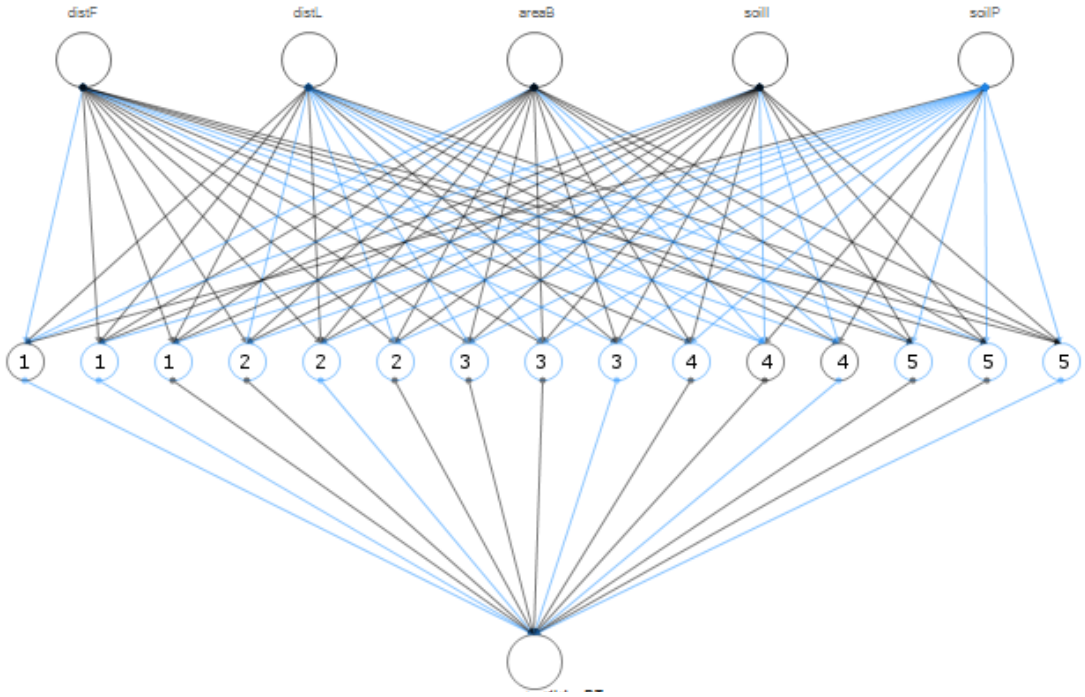


Figure 2-4 Network schematic diagram from Neurofriendly of the model used to benchmark RTS for stoats. Nodes annotated to clarify the node grouping structure.

2.3 Results

The transition between C/100TN and RTS yielded improvements in the fit (ability to explain variation R^2 in new data) and normality of residual error for stoats and, to a lesser degree, rats; spatial smoothing allowed for substantial gains in GLM fits primarily in the RTS metric of stoats (**Figure 2-5**, **Table 1**). The use of RTS over C/100TN removes an accumulation of residual error arounds scores of 0, although residual spread around high levels of either metric was still present. While spatial smoothing gave greater predictive power to GLMs for stoat C/100TN and RTS, there was a less dramatic increase in prediction success for rat C/100TN and RTS. From raw data to spatially smoothed data (500m radius) smoothing, there was a marked increase in model fit for stoats and rats. Increasing the smoothing radius from 500 to 1000 meters increased the fit of stoat models while making only minor improvements in rat models fits. A bootstrapping simulation shows the ratio of model fits between RTS and C/100TN decreases steadily as the proportion of non-zero data increases reaching convergence at approximately 1.10 where no zero-captures are included in the data (**Figure 2-6**)

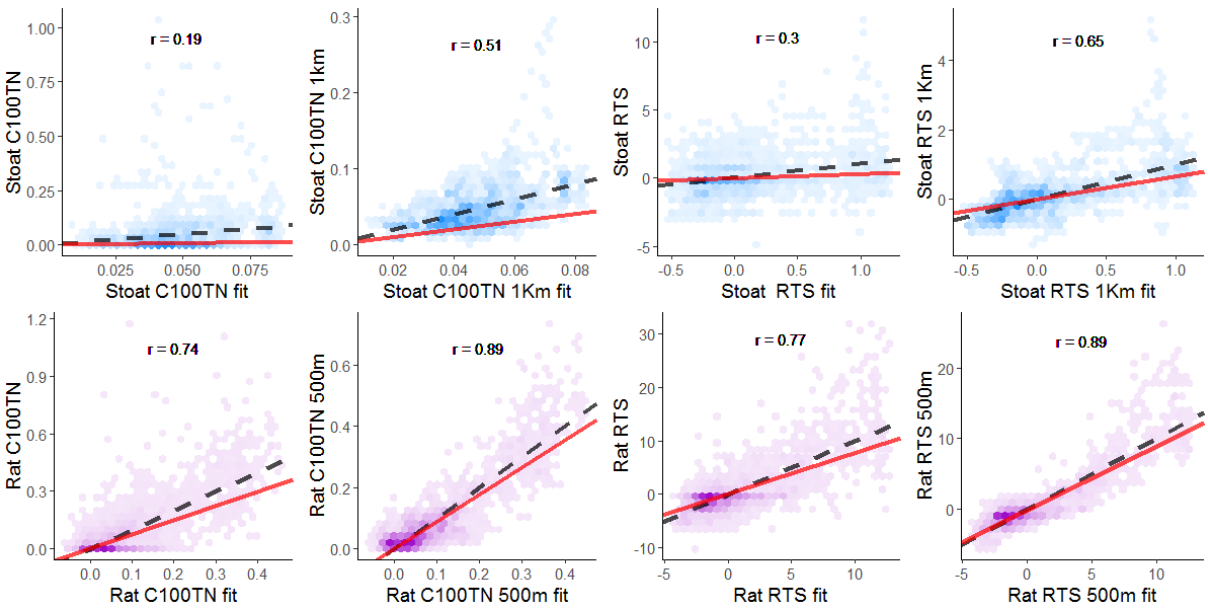


Figure 2-5 Comparison of GLM fitted values between C/100TN, RTS, and locally averaged C/100TN and RTS for stoats (top) and rats (bottom) from validation (new) data. Ideal predictive accuracy is shown with a black dashed line while the observed association between predictions and actual data are shown with solid red lines. Not shown are 500m smoothing for stoats and 1000m smoothing for rats due to redundancy. Points are drawn via hexagonal binning to depict density via alpha channel.

Table 1 Top - GLM model fit (R^2 for predicted vs actual value) and residual error normality (Shapiro-Wilk, W) for stoat and rat C/100TN and RTS raw data, 500m radius spatially averaged data, and 1000m spatially averaged data. Bottom – ANN validation data fit (R2) and residual error normality (*W*) for raw and optimally smoothed C/100TN and RTS indices.

GLM (model fit)	***Raw data***	***500m smoothing***	***1000m smoothing***
Stoat C/100TN	R^2 = 0.03; *W* = 0.58	R^2 = 0.17; *W* = 0.82	R^2 = 0.26; *W* = 0.83
Stoat RTS	R^2 = 0.09; *W* = 0.91	R^2 = 0.31; *W* = 0.88	R^2 = 0.43; *W* = 0.89
Rat C/100TN	R^2 = 0.55; *W* = 0.88	R^2 = 0.78; *W* = 0.95	R^2 = 0.81; *W* = 0.95
Rat RTS	R^2 = 0.59; *W* = 0.90	R^2 = 0.80; *W* = 0.89	R^2 = 0.84; *W* = 0.92
ANN (validation fit)			
Stoat C/100TN	R^2 = 0.04; *W* = 0.63	-	R^2 = 0.45; *W* = 0.92
Stoat RTS	R^2 = 0.12; *W* = 0.90	-	R^2 = 0.63; *W* = 0.90
Rat C/100TN	R^2 = 0.57; *W* = 0.87	R^2 = 0.88; *W* = 0.93	-
Rat RTS	R^2 = 0.63; *W* = 0.90	R^2 = 0.88; *W* = 0.88	-

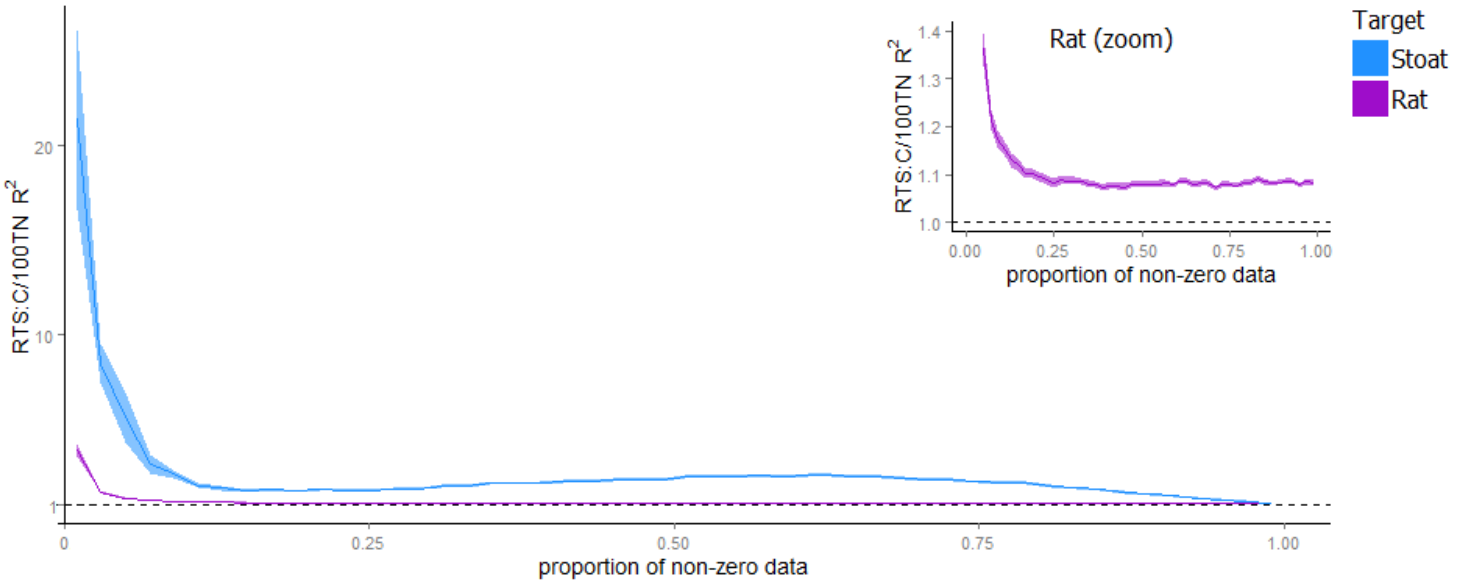


Figure 2-6 Bootstrapped mean ratio of RTS and C/100TN fits (GLM R2) at varying levels of zero inflation for stoat and rats ± 2 standard errors (shading about the mean line). Top-right contains a zoomed in plot of rat data for clarity, the y axis is truncated at a ratio of 1.4. Note that variable selection was performed on cumulative data (~10 years) with approximately 59% non-zero data, the apparent local peak in the stoat curve is likely an artefact.

Artificial neural networks (ANNs), although not directly comparably with GLMs and despite being trained on only half of the data, were generally more effective at modelling raw and smoothed C/100TN and RTS (**Table 1-1**). Both spatial smoothing and the transition to RTS gave substantial improvements in the validation fit for stoat data. For the rat data, with the exception of residual distributions, there was no notable difference between spatially smoothed C/100TN and RTS (and only minor differences for the raw indices) on model validation accuracy (**Figure 2-7**). When neural networks are applied to the landscape, there is an obvious difference between the C/100TN and RTS surfaces of stoats and a subtle difference for rats (**Figure 2-8**). For stoats, the difference between C/100TN and RTS predictions, largely a consequence of the poor C/100TN model fit, manifests as C/100TN producing a crude (irregular with overly sharp boundaries) map surface within the Murchison Mountains (sampling space) and a rough, sporadic surface in the surrounding area; RTS produces a smoother, more consistent surface. For rats, the general spatial patterns of RTS and C/100TN predictions are matching, however C/100TN produces much more extreme values outside of the sampling space and identifies different areas of the lake shore as hotspots.

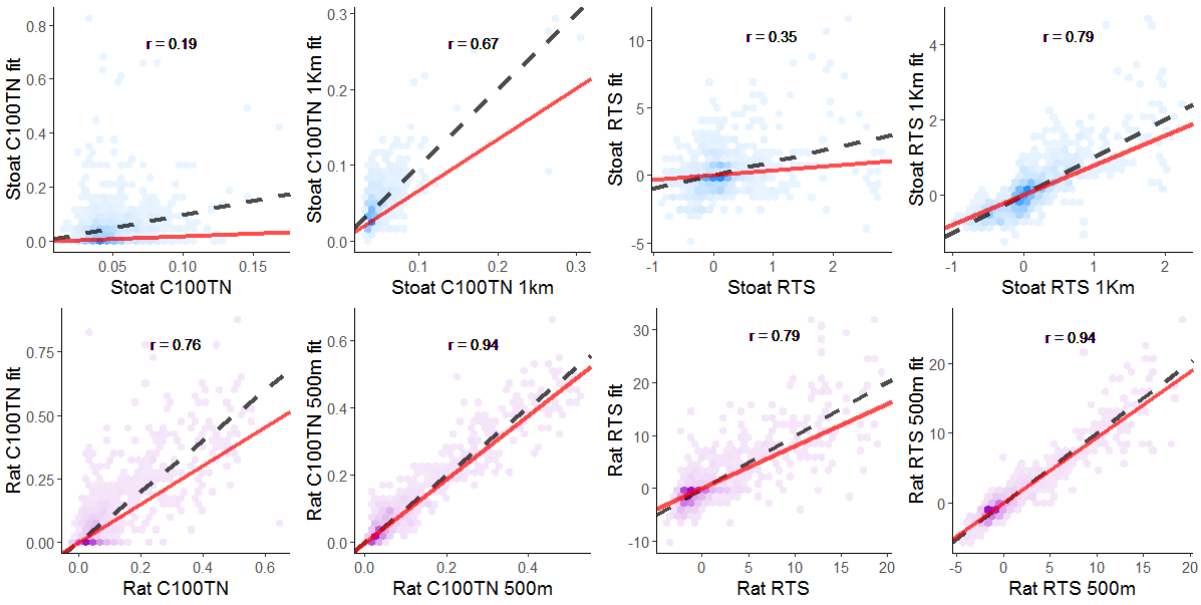


Figure 2-7 Comparison of average artificial neural network (ANN) validation fit (on 25% randomly omitted data not present in neural network training, repeated x20) between C/100TN, RTS, and locally averaged C/100TN and RTS for stoats (top) and rats (bottom). Ideal predictive accuracy is shown with a black dashed line while the observed association between predictions and actual data are shown with solid red lines. Not shown are 500m smoothing for stoats and 1000m smoothing for rats due to redundancy. Points are drawn via hexagonal binning to depict density via alpha channel.

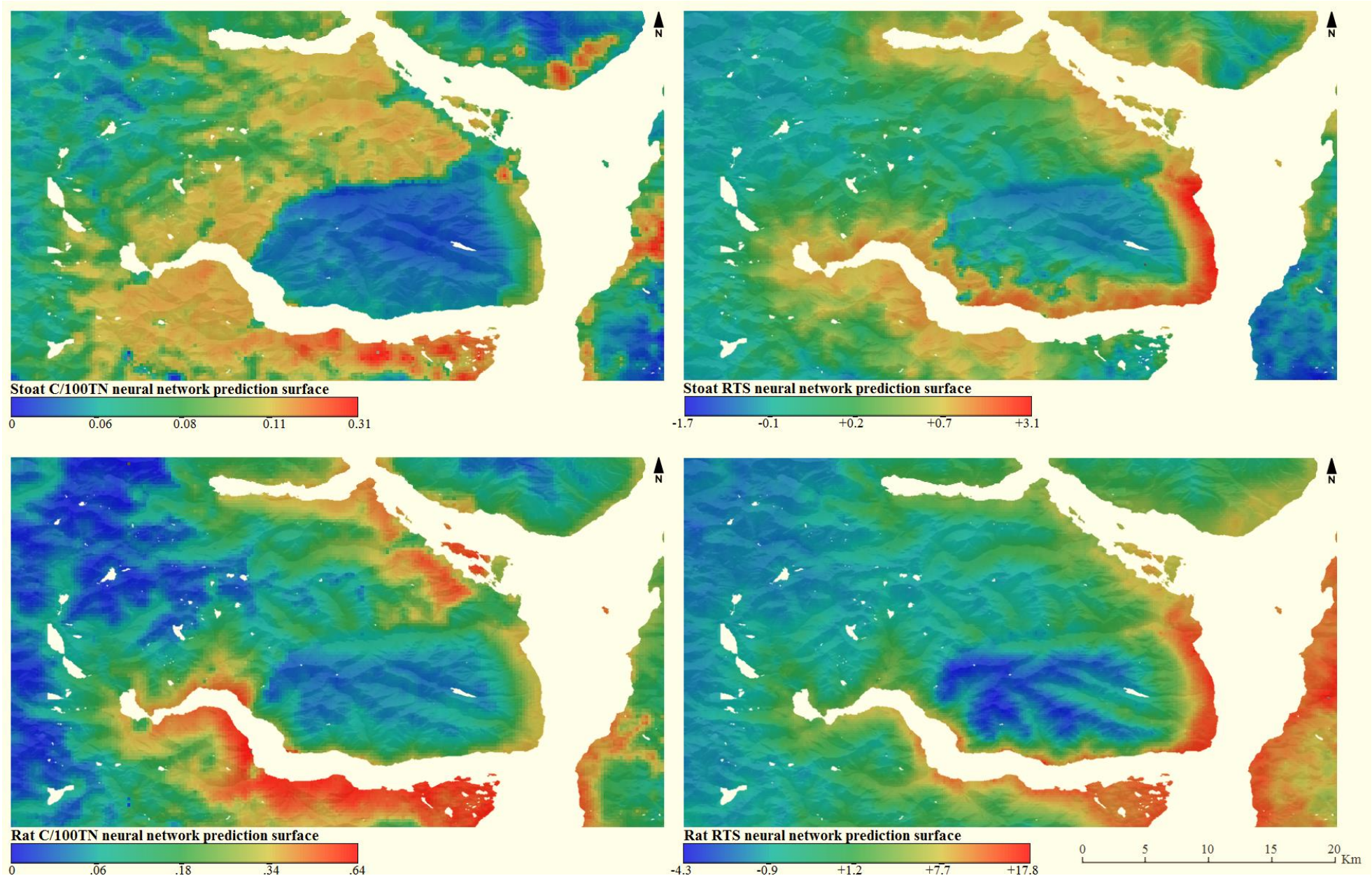


Figure 2-8 Neural network prediction surfaces (generated in MapFriendly) for cumulative (~10 year) rat and stoat C/100TN and RTS based on non-spatial environmental predictors: distance to forest edge, inland distance, whether traps were present in area B, soil induration, and soil particle size.

2.4 Discussion

In this chapter, the transition from C/100TN to RTS and from raw data to spatially smoothed data have been demonstrated as effective techniques to boost modelling success with zero inflated, micro-site heterogeneous data. The results of this chapter are generally in line with expectations that CPUE models for stoat data gained the most from RTS and spatial smoothing. While both RTS and spatial smoothing proved effective, the effect of spatially smoothing data was most apparent for the dataset used.

The effectiveness of spatial smoothing is perhaps the most obvious – modelling data at a micro-site scale with macro-site (100m resolution) environmental data introduces specific variation within the response variable that simply cannot be accounted for by the general predictors. Spatially smoothing the response variable prior to modelling allows the parameters to be set for the more general scale of the response. That stoats required a larger degree of spatial smoothing than rats is likely due to their larger home ranges and lower densities (Murphy & Dowding, 1994); it should be expected that the scale at which predictable patterns in stoat capture success become apparent is larger than the scale required for rats. Here, spatial smoothing provided the greatest gains in modelling success which should be expected as it allows predictors to match the scale of response variables while also smoothing out some zero inflation present in the data. An important consideration for spatial smoothing, like time series smoothing, is that if data are smoothed excessively, valuable information will be lost until eventually all data are reduced to the mean. It cannot be stressed enough that in practice, the degree of spatial smoothing must be carefully considered and rationally justified, in this case, the spacing of traps (100 – 200m) combined with the base-line capture probabilities were used to . Furthermore, the use of spatially explicit models on spatially smoothed data may, due to increased spatial auto correlation, permit gross over fitting, especially in the case of machine learning.

Residual trap success gave an approximate three-fold increase in model fit for raw stoat data and a lesser improvement for rat data. The greater effectiveness on stoat data is likely due to i) the higher proportion of zero captures for stoats than rats and, presumably, ii) the increased difficulty associated with modelling a generalist species. An important caveat for this chapter is that data were collected over ten years, as the timespan of data collection increases, the probability that a given trap has zero cumulative captures is reduced and the difference between C/100TN and RTS will decrease. It follows that RTS would be of most benefit for datasets collected over shorter time periods or where the target species abundance is low.

The accumulation model used to calculate RTS in this chapter was a basic linear model, however, in principal both simpler and more complex accumulation models could be used. A case for using a simpler accumulation model such as a fixed β (perhaps a set rate of one catch per 100 trap nights) is to allow easier comparison of results from different areas and data sets. Values of RTS calculated from a standard model (fixed parameters used in all datasets) could be easier to interpret, in practice, simply standardising the RTS scores would be more desirable (maintaining symmetry of RTS scores per dataset). More complicated accumulation models could, hypothetically, prove useful if they were able to include density-dependence on trap success, however preliminary attempts at deriving RTS from density-dependent models proved to be of little use. Presumably the relative uniformity of trap densities within the Murchison Mountains rendered the inclusion of a density parameter less informative.

Residual trap success should be used when i) traps have variable trap nights or zero-catch observations need to be ranked, ii) there exists a non-trivial amount of zero-inflation, and iii) when predictive models are desired (since C/100TN is fundamentally mislabelled as an effort input-adjusted response). Although C/100TN and RTS share many assumptions, RTS can mitigate the need to assume equal trap availability and, depending on the accumulation model used, a linear rate of catch accumulation. An obvious disadvantage of RTS is that it is experimental at this point – its use should be paired with C/100TN.

In this chapter, the performance gains in predictive modelling associated with a relatively small amount of pre-processing have been demonstrated via a novel, albeit simplistic, methodology. While the usefulness of RTS has not been demonstrated on other datasets, the results presented here are highly promising as a technique to effectively model zero inflated data. While it may be argued that C/100TN is similar to RTS insofar as being a transformation of the raw data, however, unlike C/100TN, RTS is able to cope with zero-capture records and has an adaptive transformation which is based on an empirically observed accumulation rate as opposed to trivial division. Future work should test RTS on different species, datasets, and, if possible, compare the correlation of the index with empirical abundance data. Although RTS is novel, I encourage managers to consider using it alongside C/100TN as it yields substantially improved model accuracy (due to the rather obvious failure of C/100TN to correctly label zero catch records), and, by differentiating zero-catch records, RTS is able to provide a clearer reflection of operational success.

Chapter 3

Development of a robust neural network training procedure to minimise overfitting with resampling and a dropnode variant

3.1 Abstract

To facilitate the long term application of neural network models in changing environments, it is necessary that the networks are periodically updated with new data such that they do not become redundant over time. When new data is added the network can resume learning; however, such an approach may lead to the network being over trained resulting in decreased forecasting abilities. To prevent a loss of forecasting ability due to over fitting, robust training procedures are required. Furthermore, to enable non-experts to safely update the network topology, any robust training procedure must be free from as many *apriori* assumptions and knowledge as possible. In this chapter I develop a robust training procedure requiring little user input to safely train a neural network without over fitting. Such a procedure is achieved using data resampling, boosting, and a modified version of dropout training. For practical application, a threaded software application is developed allowing users to easily input new data, update the network, and output predicted values.

3.2 Introduction

Predator control operations can be thought of as dynamic systems in that the ecosystems being monitored are subject to changes in response to management strategies and environmental conditions. Given the fluid nature of conservation management operations, adaptive management strategies have become widely used in the field with varying success (Keith, Martin, McDonald-Madden, & Walters, 2011; Westgate, Likens, & Lindenmayer, 2013). Adaptive management, first formally defined by Holling (1978) and Walters and Hilborn (1978) is essentially a recursive process of 1) setting conservation targets, 2) implementing management actions, 3) monitoring and analysing the system response, and then re-evaluating management targets and actions (**Figure 3-1**). Given the requirement for periodic analysis of system dynamics over time (t) when the functional responses at t_0 are almost certainly different to those at t_i, a static statistical model created at t_0 will lose predictive power over time until it becomes redundant. Iterative modelling of a complex system, in this case predator management (optimising pest removal per unit of input), as relatively small quantities of new data are acquired can be achieved by updating a trained artificial neural network (Joy & Death, 2004; Lek et al., 1996; Suryanarayana et al., 2008).

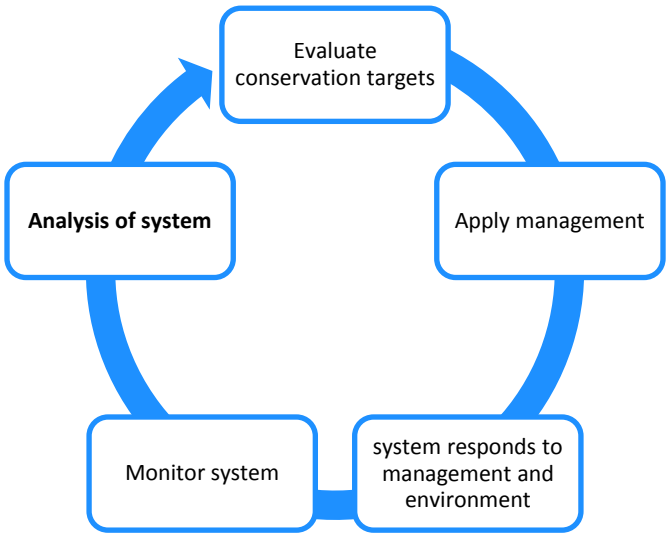


Figure 3-1 Simplified adaptive management cycle

Neural networks, despite their power, adaptability, and flexibility are liable to make poor generalisations when they are able to learn random noise apparent in the training data (Basheer & Hajmeer, 2000; Tetko, Livingstone, & Luik, 1995). As discussed by Özesmi, Tan, and Özesmi (2006), over training is a common problem in the field of artificial neural networks that is often overlooked or addressed with disparate methods. While there exist a number of methodologies for reducing the effects of overtraining, many of the available techniques require a priori knowledge from the user. For example, one of the more common preventative measures is to simply terminate training once the network error falls below an arbitrary threshold (Basheer & Hajmeer, 2000). Such early stopping can be effective, however the efficacy is typically limited by the requirement for knowledge as to what level of error is acceptable (Prechelt, 1998). Other optimisation methods include network ensembles where several neural networks are trained to subsets of the data and combined. As

highlighted by Zhou, Wu, and Tang (2002), such ensemble methods are sometimes handicapped when the network ensemble is a collection of solvers for several sub tasks rather than the main objective function. Zhou et al. (2002) apply the same criticism of ensemble networks to bagging (Breiman, 1996) and boosting (Schapire, 1990).

Hinton, Srivastava, Krizhevsky, Sutskever, and Salakhutdinov (2012) derive a novel and computationally efficient alternative to other network ensemble methods by simply dropping out a portion (50%) of neurons for each training case. After dropout training is complete, the full network is used with its weights scaled, creating an approximate averaged network of individual solvers trained to the same data. Using dropout training, Hinton *et al.* (2012) successfully broke several classification benchmark records. Despite the success of dropout training on large classification networks, the scalability of dropout training to small regression networks has yet to be tested. Mathematically, scaling dropout training to small networks presents a challenge in that as the number of neurons retained (m) veers further away from the binomial mean (np), neuron activation functions become either over or under saturated. For example, with 20 hidden neurons and a drop probability of 0.5, the cumulative weights of the hidden neurons will be ≤25% or ≥150% of their target value 4% of the time; for 200 hidden neurons such perturbations would occur with a probability of = 8×10^{-13}. Furthermore, the random dropping of individual nodes may be detrimental for networks with multiple, interrelated outputs such as that required for a model of stoat and rat abundances.

Here I develop an approximate fusion of bootstrap resampling, boosting and dropout training optimised for small numerical prediction networks with interdependent outputs. By applying data resampling techniques at each epoch and introducing “friendly dropout” I create a robust machine learning procedure suitable for continuous updating without over training. To remove the impetus of selecting the maximum number of iterations prior to training, a “brute force” method was employed such that the software creates a shadow copy of the network every time the validation error dips below the lowest recorded validation error, then, after training, the network is reverted to the state at which its validation error was the lowest. By simply reverting to the best state, the need to guess when the minimal validation error will occur for early termination is removed. To facilitate application of the model, I develop a simple user interface aimed toward conservation managers, specifically Department of Conservation staff. Such an applied modelling framework can be used as a tool in the adaptive management of stoats and rats in the Murchison Mountains without requiring the end user to groupss specialised knowledge of machine learning.

3.3 Methods

3.3.1 Training procedure

Base Training procedure

Neural network training was performed using online single step learning such that during each training epoch, weights are adapted to each training element in a randomised order. Online learning, although prone to higher residual error, is more suited to data updates than batch learning especially for larger datasets where real-time performance is not needed (Wilson & Martinez, 2003). Each element in the training set was jittered by adding a small amount of Gaussian noise (μ=1, σ=0.05) to each input value every epoch. Adding jitter has been shown have a positive effect on network generalisation (e.g. Levin, Lieven, & Lowenberg, 2000; R. Reed, Marks, & Oh, 1995; Zhang, 2007) while acting analogously to Lagrangian regularisation (Russell Reed, Oh, & Marks, 1992). Explicit regularisation is implemented by adding a small cost function to for larger weights.

The default learning rate and momentum terms used were 0.1 with a weight decay constant of 1/N (where N is the epoch limit). During training, the learning rate and momentum are decreased over subsequent epochs so as to encourage searching at the beginning of learning and fine tuning toward the end of training. The decline function used is $x_n = x_{n=0}/(1 + ((9/N)n))$ where x_n represents the starting values of the learning rate or momentum *N* and *n* represent the maximum and current epoch respectively. The effect is a tenfold decrease from the default values of 0.1 to 0.01 at epoch N. Testing was conducted using locally smoothed residual trap success (1.5km, kernel density smoothing) measurements with a single randomly assigned training and validation set. This data was used for testing as the additional smoothing allowed for greater over-fitting potential owing to the reduced variability between trapping tunnels.

The network topology used in all tests was 18 inputs, 11 hidden neurons (with hyperbolic tangent activations), and 2 output neurons (with sigmoid activations) for stoat and rat lRTS. The software automatically performs min-max normalisation on the data prior to training. The min-max transformation is simply $f(x_{ij}) = \frac{(x_{ij} - min_j)(u-l)}{max_j - min_j} + l$ where *u* and *l* are the upper and lower normalisation bounds desired. To prevent neuron saturation (see Basheer and Hajmeer (2000)), *u* and *l* are set to -0.9 and 0.9 for inputs and 0.1 to 0.9 for outputs. The discrepancy in normalisation ranges is necessitated by the hyperbolic activation of hidden neurons and the sigmoidal activation of output neurons. For fairness topology and parameter settings were deliberately chosen arbitrarily prior to testing such that they could not be tweaked in favour of any particular algorithm. The base of the neural network class is adapted from McCaffery (2014).

Partial bootstrap aggregate (bagging) and "banking"

Traditional bootstrap aggregation (bagging) relies on creating a set of models trained to bootstrap samples of the training data the outputs of which are then combined, effectively bagging can be thought of as a type of model averaging. To address problems concerned with the computation and application of multiple sub-models, a simplified version of training with bootstrap resampling was developed such that at each training epoch the network receives a bootstrap resampled version of the training data. By resampling during training, the network at each epoch can be thought of as the aggregate of all previous learning opportunities that adjusts for a new subset of data. Although bootstrap resampling with replacement introduces redundancies via repeated training patterns, online training is, in theory, insensitive to repetition of data (Wilson & Martinez, 2003). Given that the full training set is not likely to be presented at any given time, changes which reflect the underlying functional relationships being modelled will reduce training error.

A second approach to resampling was similar to delete-m jack knifing in that I simply resampled a smaller proportion of the training data without replacement. Effectively this forces the training to converge on solutions that improve generalizability (much like bootstrap resampling); however, as each epoch needs to cycle through a smaller training set, computational time is reduced. In this way, some of the training data are 'banked' every epoch – hypothetically there should not be a substantial difference in the result given by partial bootstrap resampling or data banking. The two methods are tested as, given the computational efficiency of data banking, if banking conferred the same benefit it was to be used preferentially over partial bootstrap resampling.

Dropout and friendly nodes

> *"Everybody needs a friend. Even a tree."*
> *~ Bob Ross*

Dropout training was tested as per Hinton *et al* (2012), with the exception that input nodes were not dropped; as Hinton *et al* (2012) were primarily focussed on image recognition tasks, they could safely assume several inputs would be uninformative, for my purposes I had already established which inputs to use via GLMMs and AICc. Furthermore, in preliminary tests, dropout performed significantly worse than regular back propagation (Rumelhart, Hinton, & Williams, 1986), especially when applied in online training. To address concerns about the scalability of dropout training to smaller neural networks, I attempted a modification.

The modification is inspired by artist Bob Ross (1942 – 1995), a painter often quoted assigning friends to inanimate objects. While the original dropout training was inspired by the genetic variability introduced by sexual reproduction, friendly dropout is inspired by the simple observation

that often collections of complementary elements (in various contexts) perform better than either a single collection or an ensemble of individual elements. In the same way a painting with sparse individual trees is aesthetically less pleasing than a canvas with several aggregations of trees, I posit that a neural network effectively comprised of independent nodes may be less effective than a network comprised of independent groups of nodes. By assigning neurons into a set number of groups before training, a degree of coadaptation is permitted within each group while still efficiently producing a quasi-model averaged result. Where the hidden nodes cannot be evenly divided into groups, the leftover nodes are assigned as "loners" which must cooperate with each group. Such 'loner' nodes have their weights updated in a batch fashion after each group has updated, essentially the rationale for updating 'loners' in this way is to allow them to adapt to all groups equally. The notion of providing an option to have 'loner' nodes be present at all times is that to successfully coexist with the different groups, the loner node(s) must find a function that is common to all groups ($\bar{y}$ for example) thereby reducing redundancy at the expense of increased coadaptation. Each group of neurons can therefore be thought of as a miniature neural network as opposed to treating each neuron as an independent learner. Furthermore, by maintaining a fixed number of neurons at any given time during training, the variable saturation of neuron activation functions is effectively eliminated.

Implementation of non-parametric boosting algorithm to complement friendly dropout

To maximise the benefit of combining dropout and resampling, the training procedure was adjusted so as to capitalise on the quasi-model averaging effects of dropout training. To prevent the network from becoming stuck in a local minima due to excess noise at the beginning of training, the training process is divided into three phases (**Figure 3-2**). The

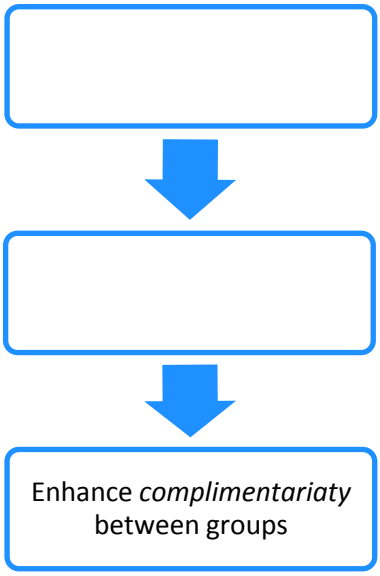


Figure 3-2 Sub-boost training procedure concept for combining dropout with data resampling

The first phase of training prioritises exploration through the solution space whereby, a single group is selected per epoch on a rotation schedule – note that unlike Hinton *et al* (2012)'s dropout procedure, each active set of nodes is now briefly exposed to all training data. The second phase prioritises generalisation by randomly resampling the training so as to force the groups to search for solutions with greater generalizability. Finally in the third phase of training, node groups are encouraged to settle on complimentary (and

not redundant) solutions via cyclic quasi-boosted resamples such that the each group is trained on data the previous group failed to fit (**Figure 3-3**). I have opted to call the procedure sub-boosting as it is a reduced version of a true boosting procedure that operates on sub-models.

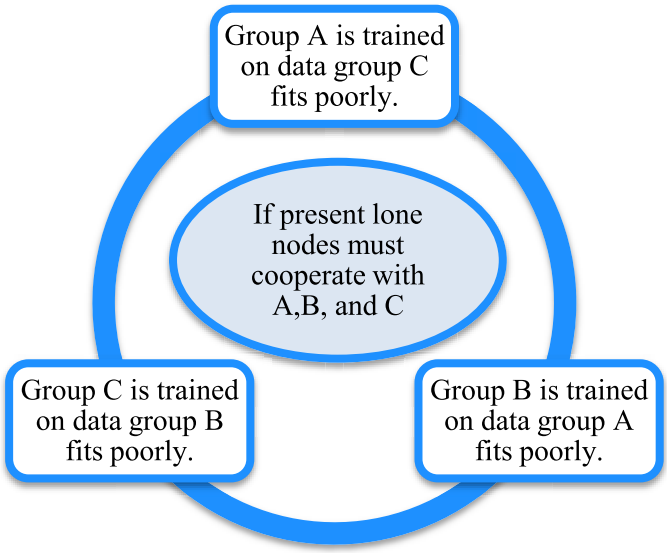


Figure 3-3 Final phase of the sub-boosted friendly dropout training procedure with three node groups (A, B, and C).

The rationale for cyclic resampling with boosting is that rather than performing a quasi-model averaging of independent, and possibly redundant sub-models, each sub-model acts to fit observations that other sub-models do not fit as well. When sub-models are averaged, they can produce an optimal combination of complimentary solutions that explicitly act to minimise each other's error. In the case of simple sub-model averaging, for example, if all candidate models over fit observation *i*, the averaged model prediction for *i* will certainly be worse than the prediction of the most powerful model; the converse would be true for equally accurate yet dissimilar models (see **Figure 3-4**). quasi-boosted friendly dropout attempts to reduce such redundancy by encouraging complimentary solutions.

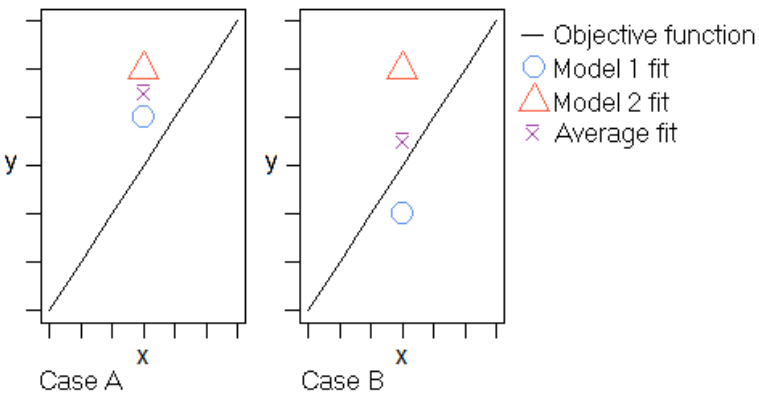


Figure 3-4 hypothetical problem when averaging similar models - in case A where model 2 similar to, yet weaker than model 1 the model averaged prediction is worse than the prediction of model 1. In case B where models 1 and 2 maintain their accuracy but the average prediction has more of its residual error cancelled

For creating boosted resamples, I loosely employ the basic concept of the gentle AdaBoost (see Freund & Schapire, 1995; Vezhnevets & Vezhnevets, 2005), whereby data are resampled according to their associated error. Data with large residuals, that is observations the network performs poorly on, are selected for the next epoch whilst the 25% of data with the lowest error are ignored. The pseudo code for the process is simply:

```
For (i observations in training data){
      residual = (compute.value(data[i]) - data[i])^2
      Add residual to array_residuals}

While (i < dataNew.Length){
      MaxIndex = index of (array_residuals.Max)
      dataNew[i] = dataOriginal[MaxIndex]
      array_residuals[MaxIndex] = -1
      i++
      }
```

Given the inherent risk associated with boosting algorithms over-prioritising data points that are simply confounded or mislabelled (Long & Servedio, 2010), and the need for robustness, I allow boosting to occur only in the later stages of training. Furthermore, the weights and biases present in the network when the validation error is lowest are saved; should the boosting phase lead to overtraining, the network weights are reverted. Preliminary work suggested that any form of boosting at the beginning of learning (when the network's weights are close to random) paralyses the gradient decent process or leads to massively over-fitted solutions.

3.3.2 Testing Procedure

Neural Network Testing

Training procedures were tested with both randomly separated training and validation data as well as a systematically created subset whereby data from a fixed ¼ of trap lines were omitted as validation data (**Figure 3-5**). The rationale of using both randomly and systematically partitioned training and validation sets is to allow repeated tests on randomly split data (where the training and validation sets are similar) while also permitting a single benchmark dataset of somewhat dissimilar elements (line omitted) that can be used to measure and compare the extrapolation performance of different models trained in standardised sub-optimal conditions.

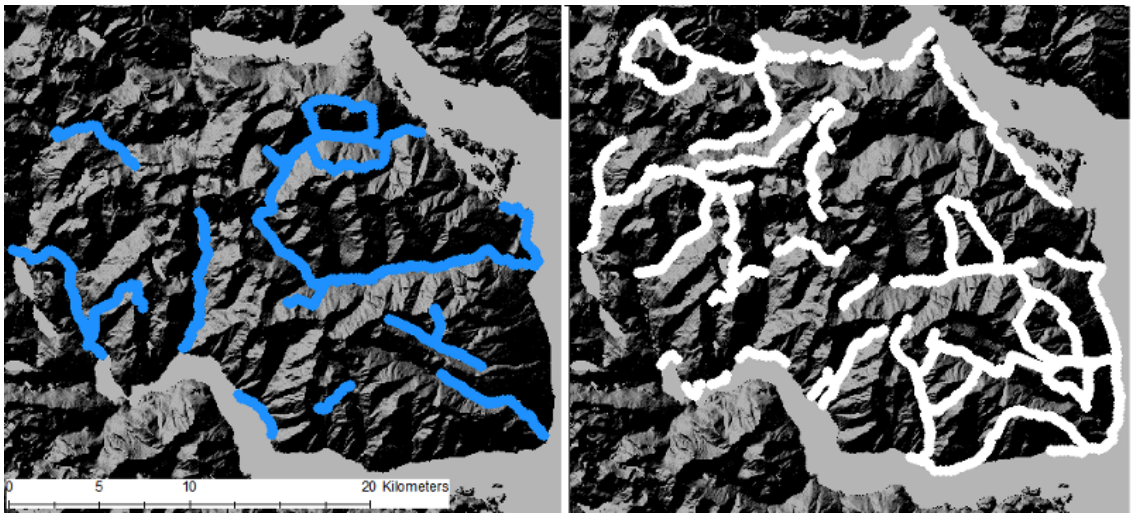


Figure 3-5 Subsets of trapping tunnels used for validation (left) and training (right).

3.4 Results

3.4.1 Training Procedures

Comparison of base training procedures

The basic back propagation algorithm performed moderately well on the training data of both datasets, however, with the line subset dataset, it very quickly began to over fit the data quite severely. Of the resampling algorithms, simple resampling with replacement (bagging) proved to be most powerful on all but the line subset validation data where the computationally faster "banking" reached a lower minimum error (where the network weights are saved). Friendly dropout performed poorly on both training sets, however, the validation performance was significantly better than all other algorithms ($p < 0.01$); furthermore, the validation error was still dropping at the end of training with the randomly split validation dataset (**Figure 3-6**).

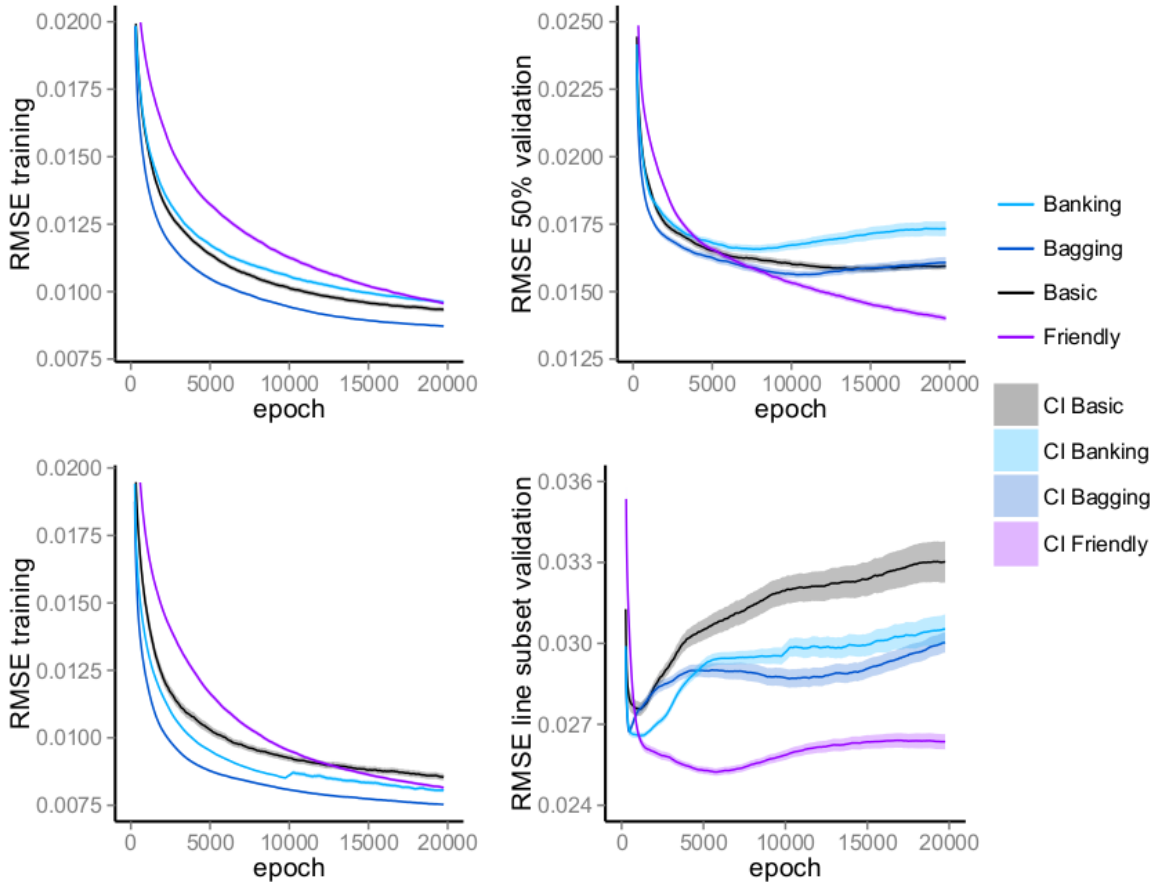


Figure 3-6 Moving average error records from 20,000 epochs of basic back propagation, data resampling via bagging (bootstrap resamples) and banking (withholding data), and friendly dropout training. Error rates are shown for training (left) and validation (right) data for the randomly selected (top) and lune-omitted (bottom) training and validation sets. Bans are 99% confidence intervals from the moving average.

Combinations of training procedures

When combining data resampling with friendly dropout, the validation error fell further, however, only the combination of data bagging and friendly dropout produced a lower minimum error than friendly dropout alone (**Figure 3-7**). Boosting, on average, provided no net gain (**Figure 3-8**), however, during individual runs, it was shown to either lower or increase validation error rather than simply having no effect.

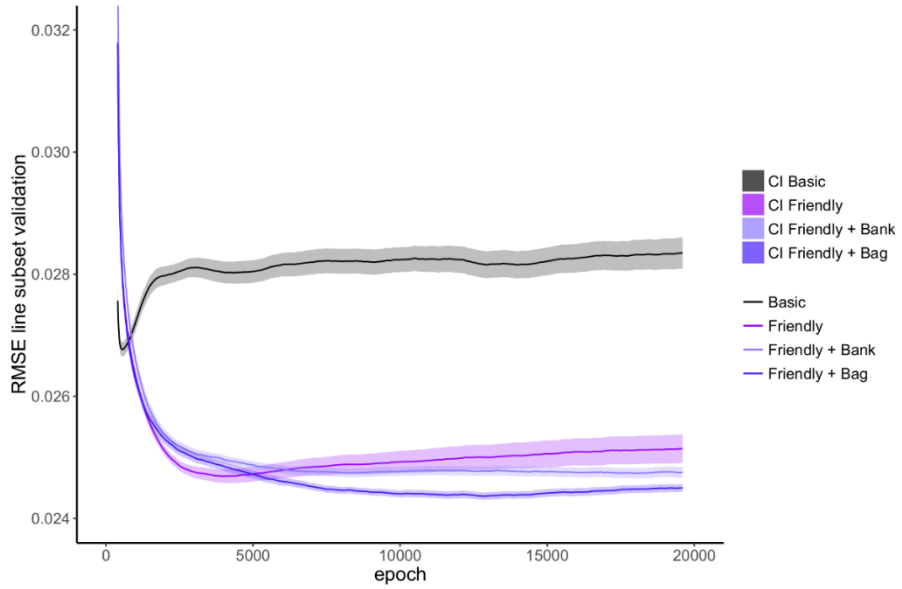


Figure 3-8 Error records from 20,000 epochs of basic back propagation and friendly dropout training with combinations of dropout and data resampling. Bands are 99% confidence intervals from the moving average. The data used were the more rigorous line-omitted set.

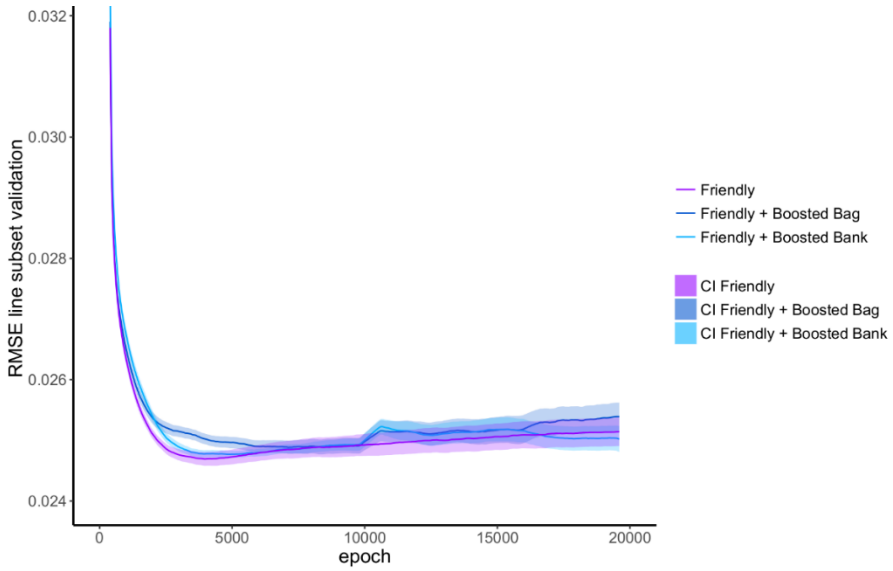


Figure 3-7 Error records from 20,000 epochs of basic friendly dropout and friendly dropout training with combinations of dropout and data resampling with an added boosting phase. Bands are 99% confidence intervals from the moving average. The data used were the more rigorous line-omitted set.

When comparing the improvement in rat and stoat predictions in the line-omitted validation data (i.e. when testing real extrapolation power), in a 10,000 sample randomisation test, we see that for rats, resampling gave the greatest improvements compared to friendly dropout whereas for stoats, the opposite was true (**Figure 3-9**). The distribution of resampled model fits is approximately normal, and there is no apparent evidence from the model fit simulation to suggest that the gradient decent curves generated are artefacts of high-leverage points.

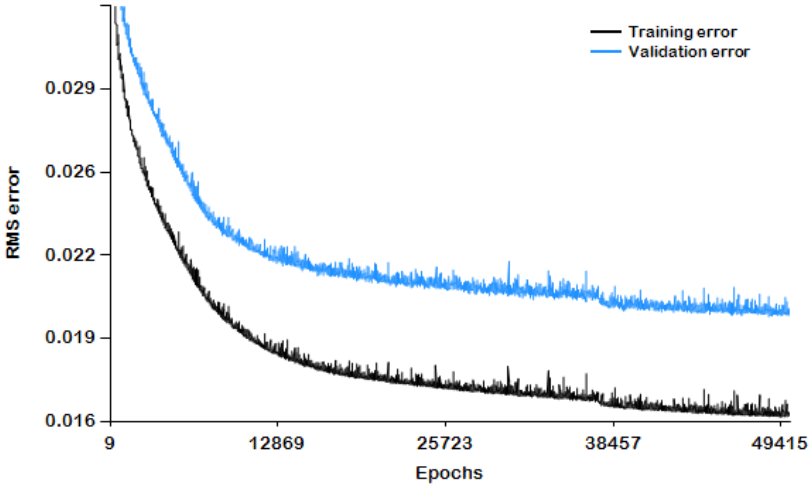


Figure 3-9 Error records from 50,000 epochs of friendly dropout training (using three groups of eleven neurons), phase one ends at 2000 epochs, phase two ends at 37500 epochs, phase three continues until the end of training. The data used were randomly partitioned

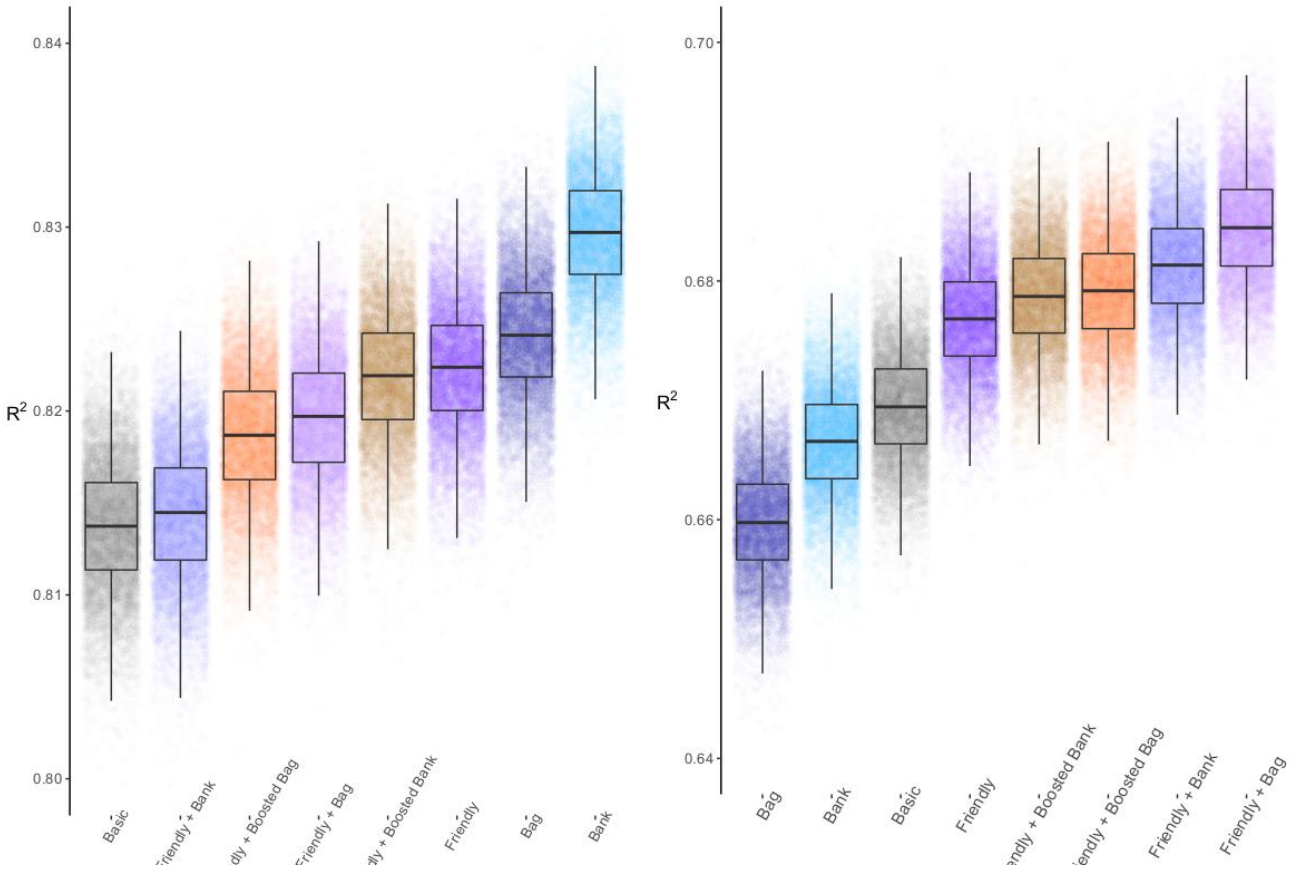


Figure 3-10 R^2 measures from 10,000 repetitions of a simple randomisation test (n = 5000 observations) of the observed and predicted values from all of the neural networks used in this section.

3.5 Discussion

In this chapter, I succeeded in improving upon the common back propagation machine learning algorithm, reducing model overfitting and increase overall prediction power on novel data. The data used for testing the algorithms were created with excessive spatial smoothing, as well as the spatial coordinates so as to maximise the potential for overfitting the data. Regardless, I was still able to get a definite improvement in the performance of artificial neural networks by using data resampling and a modification of Hinton (2012)'s dropout algorithm adjusted for regression tasks. I expect that, in practise, where the data are not prepared so as to maximise overfitting, that the predictive accuracy of the algorithms tested in this chapter will be further improved. By using such sub-optimal data, I was able to test my training procedures under more extreme conditions, therefore I have reasonable confidence that my results are reliable under normal conditions.

Friendly dropout with bootstrap resampling provided the lowest over-all validation error on new data (the trap line-omitted data set) by combining the added power of quasi-model averaging (via subnetworks of node groups) and the robustness of bootstrap resampling. Incorporating a boosting phase into the training procedure provided no benefit on aggregate as, in individual runs, they would cause either a marked increase in overfitting or a decrease in validation error. Such inconsistent results from boosting are likely a result of where the network is in solution space at the time boosting begins, as eluded to by Long and Servedio (2010), boosting algorithms can over prioritise fitting random noise rather than legitimate signal detection depending on the circumstances. While I would be hesitant to dismiss incorporating a boosting phase altogether, I would not recommend the uncritical use of boosting by conservation management staff.

There was a marked difference in the validation accuracy for rats and stoats between the different training methods, namely that for rats, simple resampling performed the best while the opposite was true for stoats. Such a result is not too surprising given that rat captures are generally much easier to model and would therefore benefit more from increased robustness over more signal detection, whereas stoat captures, being considerably harder to model, are likely hindered by the added noise of data resampling alone. The difference between the optimal models for rats and stoats implies that each species should have its own predictive model with bank resampling for the rat data and friendly dropout with bootstrap (bag) resampling for stoats.

Chapter 4

Development of software for the creation and spatial application of artificial neural network models: NeuroFriendly and MapFriendly

4.1 Abstract

To facilitate the creation and dynamic application of neural network models in a spatial context I develop two applications, NeuroFriendly and MapFriendly. As my intended end users are conservation managers rather than analysts or data scientists, I have opted to create both platforms with simplified, yet fully featured graphical user interfaces (GUI). Artificial neural networks can be created and applied to data in spreadsheet format in NeuroFriendly using the training techniques explored in this thesis. Saved neural network models (.nnx files) can be applied to data on a spatial landscape using MapFriendly which creates heat maps based on predicted vales. In this chapter, I describe the development process and core functionality of NeuroFriendly and MapFriendly as well as laying out a framework for their continued development.

4.2 Introduction

As articulated by Loraine and Helt (2002, in the context of the human genome project), the depth, complexity, and dimensionality of biological data requires advanced visualisation tools to comprehend the available information. C. R. Johnson et al. (2011) speculate that given the amount of complex scientific data becoming readily available, data visualisation – efficiently interpreting large datasets and simulation models – will be one of the larger challenges facing researchers in the 21st century. Simply providing predicted pest abundances would be of little practical use without providing the spatial context of where conservation managers should focus their actions; the spatial nature of the models I will generate should be presented in a geographic context. An applied statistical model requiring an unnecessarily large time investment from the target end-user is, in my view, ineffective; an applied model that few can apply is redundant. Code-driven analytical software such as R (R Development Core Team, 2012), though well-suited for creating new models, requires a relatively large time investment by the end-user interested in applying existing models. As my primary intended end user, the Department of Conservation, will be using the models generated in my thesis, a more efficient means of presenting my predictive models is required.

Another way of presenting dynamic simulation models is via a graphical user interface (GUI), a standalone application (app) allowing a menu-driven "point and click" interface. With a GUI, users can select the simulations they wish to run (based on pre-compiled model parameters), run the simulation, and be presented with the data already in a spatially referenced graphic display (GIS rendering) within a matter of seconds. DoC staff could then easily generate new predictions as they access new information such as climate and seed fall records. By further integrating the core functionalities of GIS software such as ArcGIS (ESRI, 2011) to create an all-in-one simulation application, DoC staff could save more time and money should they wish to do further geographic analysis without reformatting data or buying new software licences.

Here, I developed two software applications, one for the creation and updating of artificial neural networks (Neurofriendly) and another as a basic GIS platform for creating spatial predictions from saved artificial neural networks (Mapfriendly). The software was developed under the Microsoft .net framework (4.5) which is part of Microsoft's common language runtime (CLR). The advantage of creating an application under the Microsoft CLR is that the program can run on any modern computer in a Windows NT environment. C# was chosen as the coding language so as to enable easier modification by other users within the Visual Studio IDE which is arguably the best available software development environment (Griffiths, Adams, & Liberty, 2013). Although C# can be noticeably slower than lower level languages such as or C++ or FORTRAN, real time performance is

not required, as such I have opted for ease of implementation and maintenance over raw processing speed. Furthermore, with regard to performance concerns, as highlighted by Griffiths *et al.* (2013), the JIT compiler used by C# can mitigate, to some extent, the performance losses compared to lower level languages using static compilers.

4.3 Main Components

4.3.1 NeuroFriendly

NeuroFriendly requires input data to be in comma separated (CSV) format with predictors in the first n columns and the target values to be contained in the last columns; data labels, if used must be present in the first column. The app will check for the presence of non-numeric input in the first column, if non-numeric data is found, the first column is set as a label column. If numeric labels are used, the user is provided an option to force these to be counted as labels instead of data. Normalisation of data is performed internally allowing the user to supply raw input, the parameters used for transforming data are saved within the network allowing the model to be applied to new data as easily as possible.

NeuroFriendly can natively be used for creating regression oriented neural networks for any numerical data set; for optimal classification tasks, a simple code modification to change the output activation function to softmax. The softmax code is included but not currently implemented in NeuroFriendly. If performance is needed above model power, parallel implementations of resilient back propagation (Riedmiller & Braun, 1993) and batch back propagation can be used in place of the default online training algorithm. Data resampling, friendly dropout, boosting (see Chapter 3) can also be implemented per user selection as can Hinton et al (2012)'s dropout algorithm on hidden neurons. Random noise (jitter) can also be added during training to help mitigate overfitting (default: μ =1, σ = 0.05). Regularization (additional error penalty for large weights) is also added to help equalise the input to hidden weights. Currently, only one layer of hidden neurons is supported as, for regression tasks, the value of adding multiple hidden neuron layers is questionable.

Trained neural networks can be saved as version locked serialised XML files with a custom .nnx suffix. XML format, although larger than a customised binary format allows easier modification should the need arise, version locked files with a custom suffix is to prevent runtime errors should a user select an incorrect file. The XML serialisation code is derived from Rozas (2008). A simple real time network visualizer is provided as are graphs showing the training error as well as current model fits and input weighting. Input weighting plots show the absolute contribution of all weights

stemming from each input thus allowing some visualisation of input importance. For more details, refer to the NeuroFriendly software manual in Appendix 1.

Saved neural networks can be used to generate predictions for new datasets, the new data must have the same input order as the data used to train the network, the output columns (if populated), are skipped.

Given that most modern computers have multicore processors, and a large proportion of the CPU load in NeuroFriendly (such as UI updates and error calculation) can be executed asynchronously without affecting network training, I opted to distribute the training process across multiple threads. From the UI, the training thread is launched as a separate thread, the training thread then launches a thread for data resampling and error calculation. The resampling and error calculation threads are programmed to execute at a maximum of only once per epoch. The error calculation thread arguably carries a higher workload than the main learning thread as it must compute the error for two datasets (training and validation), this may result in error logging at a rate less than once per epoch. As error logs need not be stored for each epoch to provide useful information, some latency here is acceptable. The resampling thread has relatively few calculations to perform with the exception of residual calculation thus it should execute once per epoch; should latency occur in the resampling thread, a serial implementation can be used at the expense training speed. To enable writable data to be safely passed between threads where use of an object's sync root is unfeasible, deep copies are created via reflection.

4.3.2 MapFriendly

MapFriendly is a geospatial platform for applying neural networks, specifically those created by NeuroFriendly, to a given landscape. Additionally, MapFriendly offers some basic GIS functionality care of the DotSpatial library (DotSpatial, 2014). This edition of MapFriendly is "hardcoded" for the Department of Conservation meaning there is default data for the Murchison Mountains bundled with the software. The software can still be used with other datasets, however, preparation of other datasets may require other software such as ArcGIS (ESRI, 2014) or the R statistical language (R Core Team, 2014). This version is also hardcoded to work with WGS84 UTM zone 59s projections, furthermore, some functionality was omitted for simplicity.
To assure compatibility, a NeuroFriendly version is bundled with the app such that the user can 1) work between the two apps more efficiently, and 2) always have access to a compatible version of NeuroFriendly in the event of software updates.

MapFriendly works with vector, raster, spreadsheet, and neural network data formats. To generate spatial predictions from a trained neural network, the user must: 1) load a neural network model, 2) load a CSV of data to predict (including latitude and longitude columns), and 3) use the "Generate Predictions" function to apply the model and interpolate the predictions into a raster surface. When loading data to be used for network predictions, MapFriendly uses the loaded NeuralNetwork to find the correct data columns regardless of input order, if an exact match is not found, the application will search the data for closely matching columns which can be used following confirmation from the user. MapFriendly has two default colour schemes, a Matlab-like rainbow and a dark blue to light yellow colour ramp suitable for colour blind persons. The user can also control their map aesthetics manually if desired. Some basic GIS tools such as conversion, interpolation, and terrain analysis are included as a convenience. The app also includes a model viewer for neural networks, and a data visualisation tab for any available data (for example, plotting a histogram of the mean distance from a river for all traps). Also included is functionality to format and export a map image with scale bar, north arrow, etc. MapFriendly is built around the GDAL (2015) and DotSpatial 1.7 libraries which provide the core geospatial rendering functionality. Within the DotSpatial library, I have modified some of the interpolation and raster colourisation code to execute in parallel allowing for finer scale interpolations and faster rendering than is possible with the source library. A copy of the core neural network class from NeuroFriendly is built into MapFriendly allowing all class methods to be executed within MapFriendly as required.

4.4 User Interface

Where possible, the user interface design of NeuroFriendly and MapFriendly was built to Mandel's (1997) “golden rules.” Specifically, I focused on the following:

- Displaying descriptive text and messages via buttons and interface features which contain descriptive function names.
- Ease of navigation by reducing, wherever possible, the number of hidden menus.
- Accessibility for users of different skill levels with a simplified interface and descriptive tool-tip text.
- Reduction of the user’s memory requirement by implementing tabbed data views and minimising unnecessary scrolling.
- Visual cues via icons alongside text.
- Progressive disclosure of functionality and sustained user context as the user moves through the dataflow by not enabling features until they appear in the user’s potential workflow.
- Visual clarity with tabbed browsing instead of multiple hidden windows or simultaneous displays.
- Maintained visual consistency between NeuroFriendly and MapFriendly.
- Aesthetic appeal by using a modern flat styled design.

The user interfaces of NeuroFriendly and MapFriendly can be seen in depth in **Appendix A**, the GUIs were designed in Microsoft Visual Studio (2012 -2015) and have had their aesthetic design checked and modified (via suggestion) by professional software engineers. Wherever possible, automated error correction procedures are used to manage exceptions and plain-English error messages with hints for trouble shooting are provided. Generic Try-catch exception handling is, however, avoided wherever possible as, in the event of a runtime error which could alter model output, it is safer (and less CPU intensive) to simply allow the application to crash rather than permitting mistakes in inference.

4.5 Roadmap For Continued devolopment

The continued development of NeuroFriendly is intended to include:

- Classification networks, the functionality for which is already in the program's code, however these are not enabled as the algorithms have not been tested extensively since the first prototype of the app was created.

- Multi-layer neural networks and cov-nets will be added, however their inclusion will require substantial changes to the model visualisation tab as well as the underlying dropout algorithms.

- GPU acceleration will be enabled via C++ amp, however this feature is not currently enabled due to technical restraints and testing requirements. The C++ amp library (AmpFriendly) will be extended to include generic accelerator functions. C++ amp was chosen as NVIDIA CUDA is platform specific and OpenCL would require more development time (and is not fully supported on NVIDIA GPUs), however, if significant performance gains can be made via OpenCL, GPU acceleration features will, whenever possible, be implemented in OpenCL.

- A C++ based R package is also planned following further development in the C++ amp library, effectively, this is intended to work as a port for existing C++ code.

The continued development of MapFriendly is intended to include:

- Datamining from shapefiles and raster layers to allow easier exploratory modelling and a simpler, all-in-one, process for preparing prediction data.

- Support for linear models and R model objects is planned via the Rcpp library (Eddelbuettel & Francois, 2011), however this will require some substantial UI overhauls to accommodate different model types.

- GPU acceleration may also be implemented if performance issues arise in the future.

Appendix A

Software Manuals

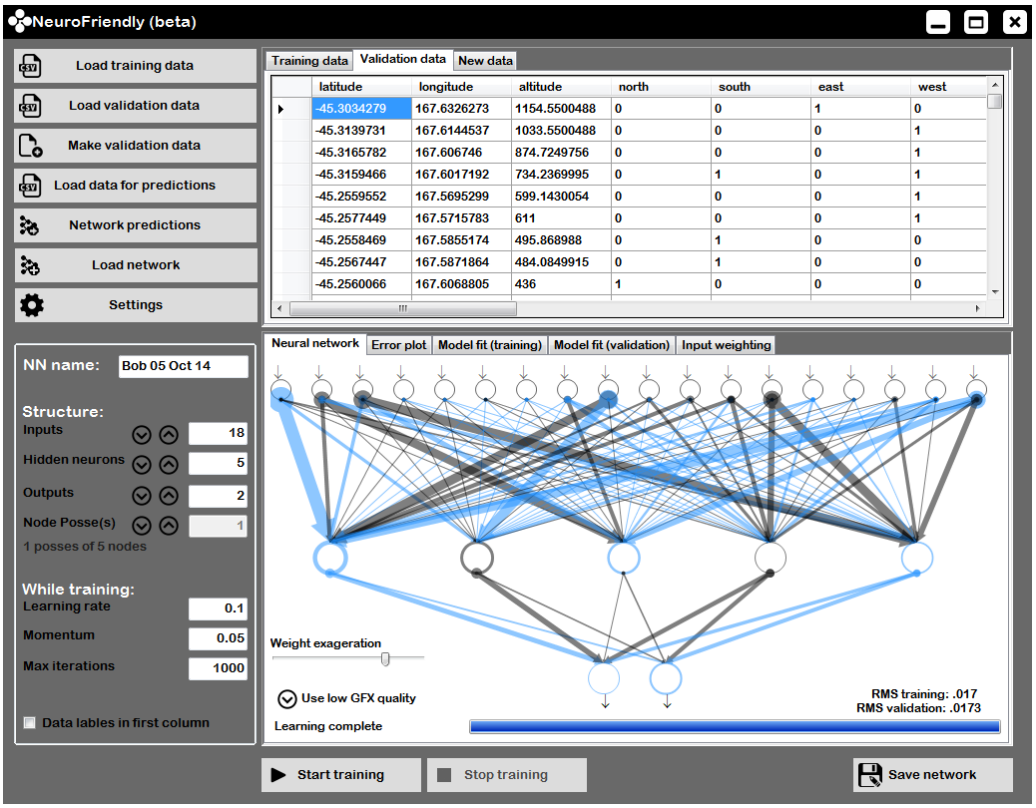
Contained are user manuals fot the applications devoloped in this thesis. Each manual is intended as a stand-alone doccument to be released with each application.

A.1 NeuroFriendly User Manual

NeuroFriendly

Neural networks. Friendly drop nodes. Easy.

NeuroFriendly Version 1.0 user manual



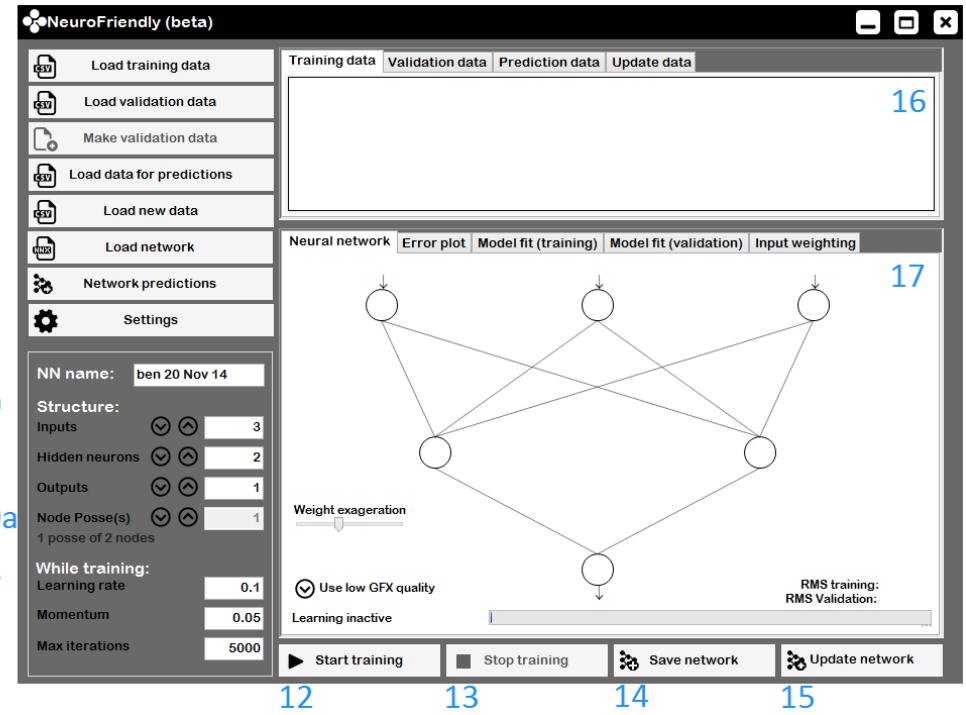
Benjamin H. Wiseman, Centre for Wildlife Management and Conservation, Lincoln University, Canterbury, New Zealand

Overview

NeuroFriendly is a platform for training and updating artificial neural networks, with an emphasis on regression style problems. NeuroFriendly supports dropout and friendly dropout training alongside options to resample training data during training and inject artificial noise to help mitigate over training. The application is designed with single layer feed forward regression neural networks with online training in mind, however, other types of network can be implemented by modifying the source code or by request.

Quick start guide

4.5.1..1 User interface overview



Misc.

16: Use these tabs to inspect the training, validation, prediction, and update data. Files cannot be permanently modified from data tables within NeuroFriendly. Note no data is loaded in the example image hence there is no table shown.

17: Use these tabs to switch between the model view with training progress and network diagram (**blue connections and nodes** represent **negative** weights and biases**, black is positive**); a plot of the RMS error decent during training, scatterplots of the networks fit to the training and validation data, and a graph of the input-to-hidden weights of all input variables.

Data IO:

1: Load a CSV of training data (what the network will learn).

2: Load a CSV of Validation data (how the usefulness network will be tested).

3: Make a validation data from a random subset of the training data.

4: Load a CSV of data that contains predictor variables to generate a forecast.

5: Load a CSV of data that contains new data to update an existing neural network.

6: Load an existing neural network (.NNX)

7: Generate a forecast of the prediction data from either the active neural network or a saved neural network.

14: Save a trained network.

Neural network and training:

8: Change advanced training settings such as selecting which dropnode and resampling schemes to employ.

9: Set the model name (by default it will be your computer's username and date).

10: Adjust the number of input (independent variables), Hidden (model flexibility), and output (dependent variables). Input and output numbers are automatically adjusted once training data are loaded.

10a: Adjust the number of node posses (groups). Use the toggles to automatically divide the nodes into equal sized posses for friendly dropout training. Add loners by adjusting the number of hidden neurone afterward.

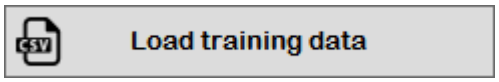
11: Adjust basic training parameters such as how quickly the network adapts during training (learning rate), how smoothly the network will train (momentum), and how long the network will train (max iterations).

12: Start training; 13: Stop training early.

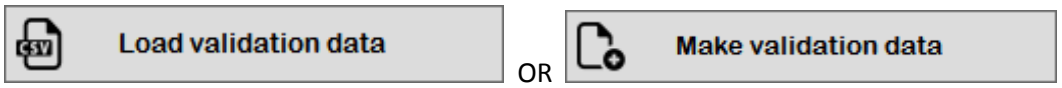
15: Retrain an existing network with new data or train to validation data.

4.5.1..2 Training a new neural network ASAP:

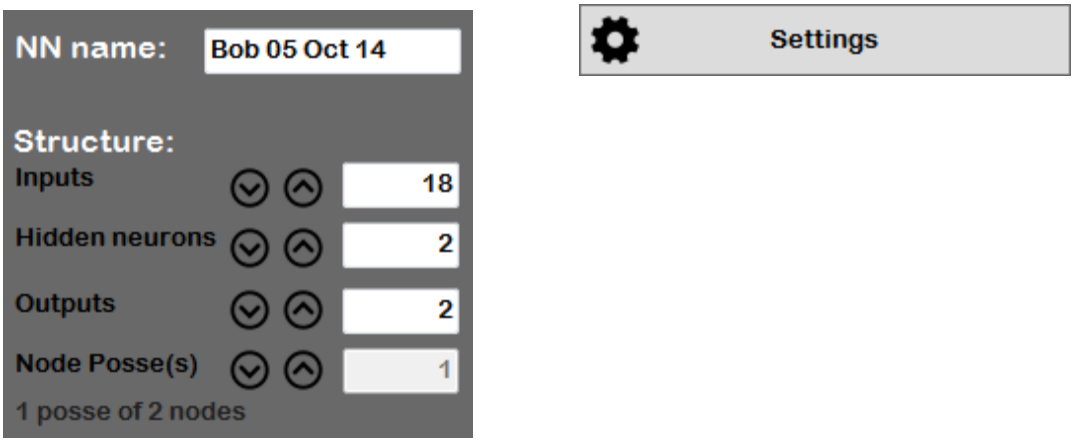
Step 1 Load training data



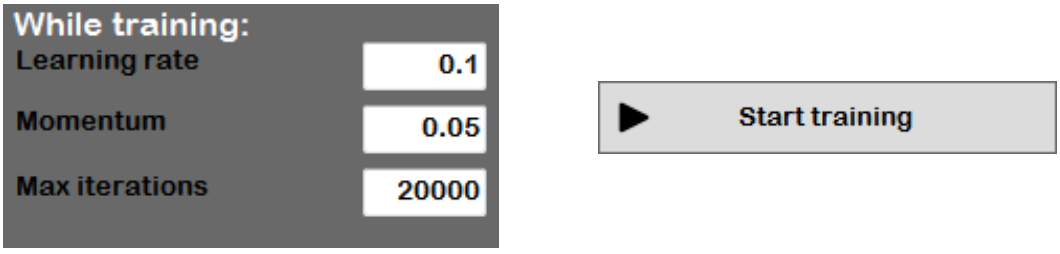
Step 2 Load or create validation data



Step 3 Set the topology of a new neural network and adjust advanced settings. The settings menu allows you to adjust drop nodes, resampling, etc. and is covered in depth later in the manual.



Step 4 Specify the learning rate, momentum, and number of iterations and begin training

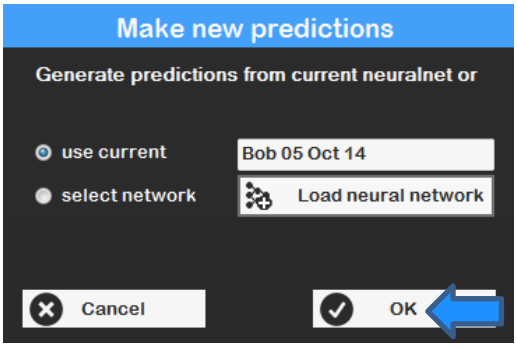


Step 5 Save network when finished

Applying a trained network to data:

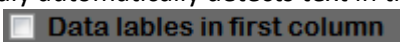
Step 1 Load the data for which you want predictions

Step 2 Press “Network predictions” Select the current network or use a previously saved one, press Ok to save predictions in a new CSV



Getting data in

4.5.1..1 Data format

NeuroFriendly requires data in CSV (comma separated value) format with one row per observation. Input data must be numeric[1] with the exception of the first column which (optionally) may contain label data. NeuroFriendly automatically detects text in the first column, however, if your labels are numeric, you must tick the 

check box on the front panel.
Dependant (*Y*) variables should be the last columns in the file; you may specify an arbitrarily large number of outputs. Do not include extra columns in your data as the software will attempt to work with these.
For example, in a spreadsheet application where we wish to predict two Y variables from several *X* variables at labelled points, the data may look as follows:

Labels	X_0	X_1	X_2	X_3	Y_0	Y_1
AAB01	42.000	2.718	0.577	8675.309	0.608	0.512
AAB02	35.400	3.142	1.618	1729.000	0.780	0.600

Note that data normalisation (required for machine learning) is performed automatically in the background when the network is learning. Trained networks store the values used to normalise data – don't worry about the tedium of storing values and back transforming data.

[1] Future versions will support transforming non-numeric factor variables automatically. To programmatically transform factors into binary dummy variables in R, one can use the `model.matrix` function.

4.5.1..2 Training and validation data

The training data is the bulk of the data that the network will be trained on, the validation set is used to test the accuracy of a network as it learns on a training set. Both training and validation data are required to begin the training a network. The column order of the training and validation data must match exactly (in the format specified above), furthermore the column order of the training data dictates the column order of any data the network it to be applied on. We can either create a separate validation set (a specific year for example) or a random validation set can be partitioned from the training data.

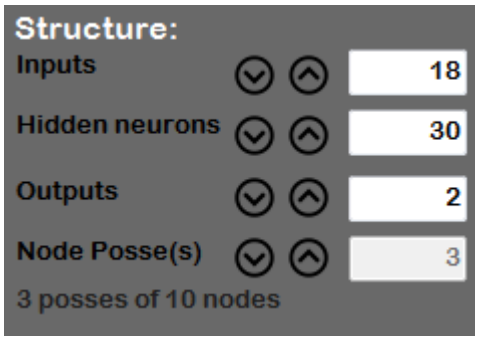
When making a validation set, there are some things to consider:

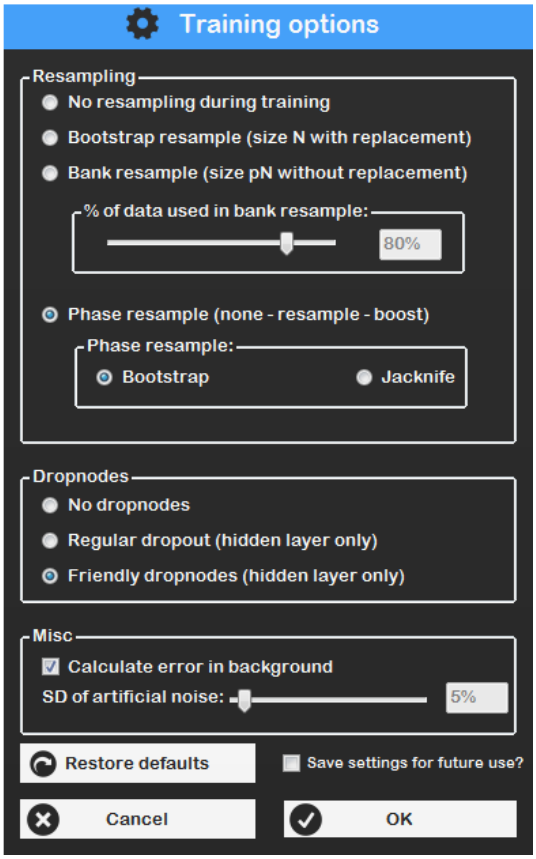
- If the training data is too small the model will rapidly over fit the data and perform poorly in practical applications.
- If the validation data is too small the estimate of accuracy becomes unreliable.
- Generally, the validation set should be between 20% and 50% of the full data.

Creating a neural network model

4.5.1..1 Training a new network

Before training a network, you must specify the number of columns used for independent and independent variables – for convenience when training data is loaded, the network size will automatically change to the number of columns minus one. Changing the number of input nodes will cause a corresponding opposite change in the number of output nodes such that the total input and output nodes will equal the number of columns in your data (you needn't count your inputs if you know how many outputs you have). Next, adjust the number of hidden neurons to taste, there are no particular rules governing how many hidden neurons should be used, however a common rule-of-thumb is to use $\frac{1}{2}$ to $\frac{2}{3}$ the number of inputs. If you are using friendly dropout, you can easily specify more and split them into "posses", for example 30 hidden nodes split into 3 posses of 10. The toggle buttons for node posses will automatically set the number of posses such that nodes are evenly divided between them. To add "loner" nodes (which are present in all posses), increase the number of hidden neurons after setting the posse size. While multiple loners can be added, the presence of more than one, or large number of loners relative to the number of nodes in a posse can increase validation error and destabilise the training procedure.





From the setting menu you can toggle more advance training options. Namely if and how data are resampled during training, if and how hidden neurons are dropped, how much synthetic noise to inject during training, and whether to run an extra background thread for displaying training and validation error while training.

Data resampling techniques are a means to reduce overtraining by forcing the network to generalise as it trains (note that after a great many iterations the network will have an opportunity to learn all of the data).Bootstrap resampling resamples data uniformly with replacement (i.e. some items are omitted, some are repeated) and tends to result in slightly lower validation error while noticeably smoothing the training error. "Bank" resampling is similar to statistical delete-m jackknifing whereby some portion of the training data is randomly withheld each iteration so as to prevent the network from generalising while speeding up the training process (as less data are cycled through). "Phase" resampling allows the network to train normally for the first quarter of training before resampling (with either bootstrap or bank resampling) until the last quarter where it performs weighted resamples (similar to statistical boosting). Phase resampling can lead to better overall performance, however it is more sensitive to weak or noisy data. Injecting random noise has two purposes, the first is mathematical as it acts as Lagrangian regularisation while the second, more practical benefit is to increase the

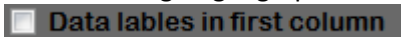
generalisation ability of the network. Of course, one should be careful not to inject too much noise. The noise is normally distributed with a standard deviation between 0% and 30%, by default 5% is selected. Dropout training greatly reduces over fitting by acting like a quasi-model-averaging procedure, however for small (<100 hidden neurons) networks, it can greatly reduce the overall performance. Friendly dropout (the core of this application) was developed to address the issue of randomly removing individual nodes during training by simply removing fixed sets of nodes allowing a quasi-averaged model without losing the stability and power of dropout training on smaller networks. Friendly dropout (in testing) has proven to give lower validation error than traditional back propagation (or dropout) at the cost of more over fit than dropout. Friendly dropout is the default option.
Finally, before training you can adjust the learning rate, momentum, and maximum iterations if desired. The learning rate and momentum will decrement over time such that they will be at $\frac{1}{10}$ of their initial values at the end of training. The decay of the learning rate and momentum are currently hardcoded (as are weight decay and weight restrictions). Currently NeuroFriendly takes a "snapshot" of the network at the lowest validation error – this admittedly brute force feature frees you from guessing how many iterations (epochs) are optimal.

4.5.1..2 Saving and loading networks network

NeuroFriendly neural networks are saved in .nnx format. Networks can be saved and loaded via the UI, and if necessary edited in XML format. To allow updating, saved networks retain the training data, normalisation values, learning rate, momentum, training and validation errors obtained when validation error was at its lowest, and as other training related values such as weight deltas, gradients, topology, and so on. Consequentially, .nnx files will be at least as large as the training data however they can be restored and updated with new data more easily. Neural network models are currently version locked – you can only load networks made with the same version of the software.

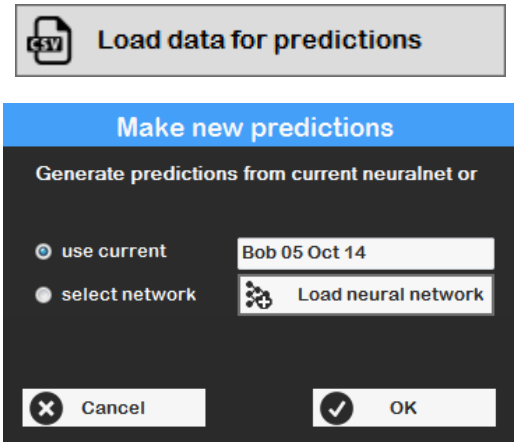
4.5.1..3

4.5.1..4 Forecasting from a neural network

The data you wish to apply a network to must be formatted as the training data for the network was, however the output columns can either be absent or present. Again, text labels will automatically be detected (useful for binding to geographic information) however if numerical identifiers are used you must tick the 

check box (under settings). To add data to be forecasted from, press "Load data for predictions".

To apply a NeuroFriendly network to your new data, simply click "Network predictions", select the network you wish to use, and press ok. You will be asked where you would like to save the copy of your new data with predictions appended.

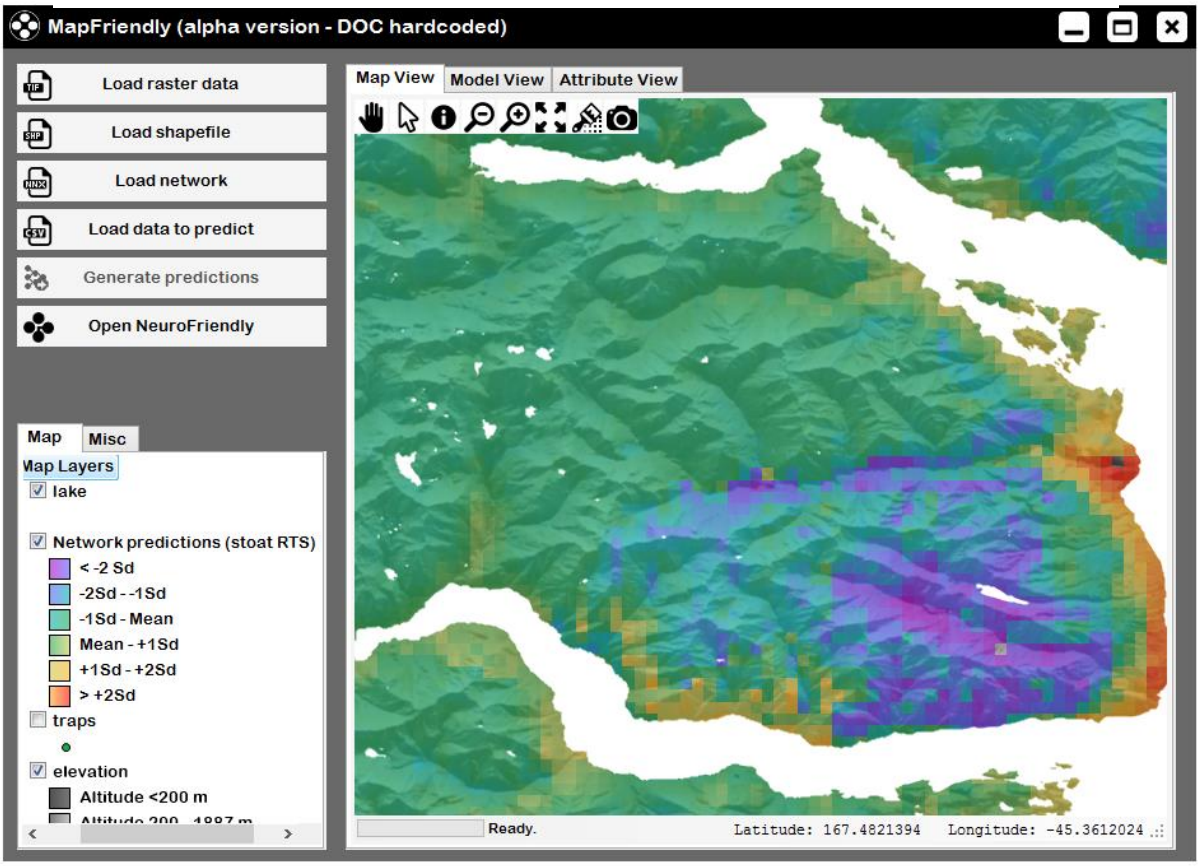


A.2 MapFriendly User Manual

MapFriendly

Neural network predictions to GIS. Easy.

MapFriendly Version 0.9 (DOC Hardcoded edition) user manual

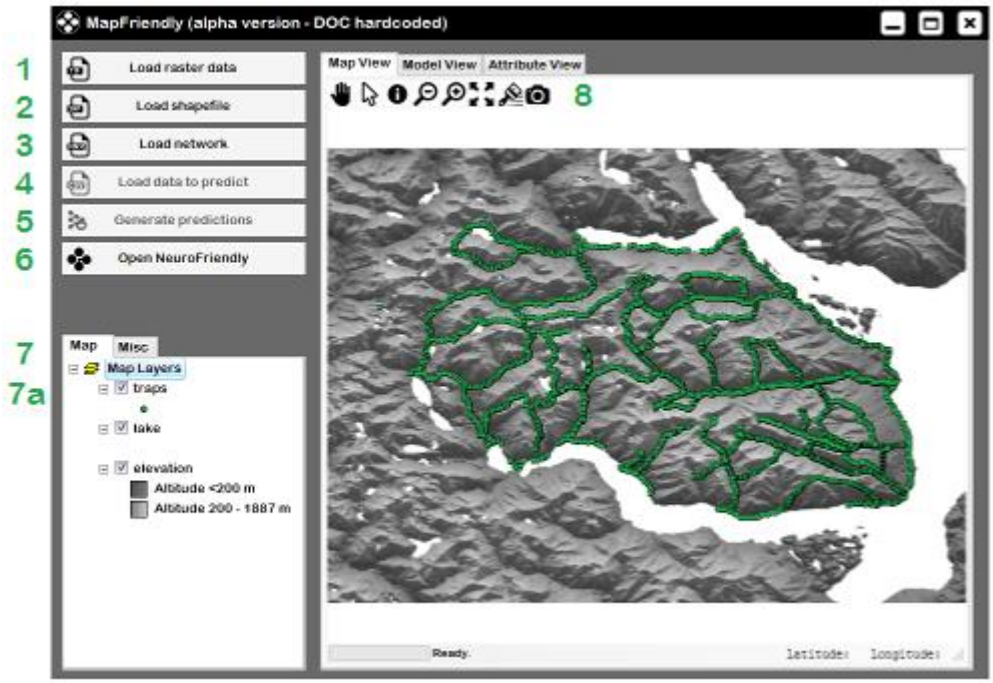


Benjamin H. Wiseman, Centre for Wildlife Management and Conservation, Lincoln University, New Zealand.

Overview

MapFriendly is a geospatial platform for applying neural networks, specifically those created by NeuroFriendly, to a given landscape. Additionally, MapFriendly offers some basic GIS functionality care of the DotSpatial library (DotSpatial, 2014). This edition of MapFriendly is “hardcoded” for the Department of Conservation meaning there is default data for the Murchison Mountains bundled with the software. The software can still be used with other datasets, however, preparation of other datasets may require other software such as ArcGIS (ESRI, 2014) or the R statistical language (R Core Team, 2014). This version is also hardcoded to work with WGS84 UTM zone 59s projections, furthermore, some functionality was omitted for simplicity.

Quick start guide



User interface overview

Data IO:
1: Load raster (grid) data such as digital elevation maps, or prediction surfaces.
2: Load shapefiles (vector data: points, lines, and polygons) such as trap locations, transects, lakes, etc.
3: Load a neural network (.nnx) created in NeuroFriendly.
4: Load a CSV containing points (with decimal latitude and longitude) **the neural network can be applied to**. This CSV must also contain data for the particular network's inputs.

Neural network operations
5: Apply the currently loaded **neural network** to the prediction data. This will also create interpolated raster images of the area for the network outputs.
6: Opens a **NeuroFriendly app.** This assures compatible versions of NeuroFriendly and MapFriendly.

Map tools
7: The **map legend** – layers can be rearranged here and manually edited by right-clicking on them.
7a: Miscellaneous tools – Some basic geospatial operations can be performed via the tools in the "misc." tab.
8: Map tools from left to right: **pan** mode, **selection** mode, i**dentity** mode, **zoom out**, **zoom in**, **zoom to extent,** raster **colouriser** (allows predefined colour schemes to be applied to raster layers), **export map** image (opens an image editing, exporting, and printing window).

Tabs: Use tabs to examine the **map**, the **model** (neural network), or the **data** (layers)

From a model to a map ASAP

Step 1 The app will automatically load a base map containing a digital elevation model, a layer containing trap boxes, and a mask for lakes.

Step 2 Assuming you already have a trained neural network from NeuroFriendly, load the model you wish you use with the Load Network 

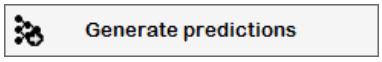
button. The model can be viewed in the Model View tab. If you need to create a neural network, launch NeuroFriendly.

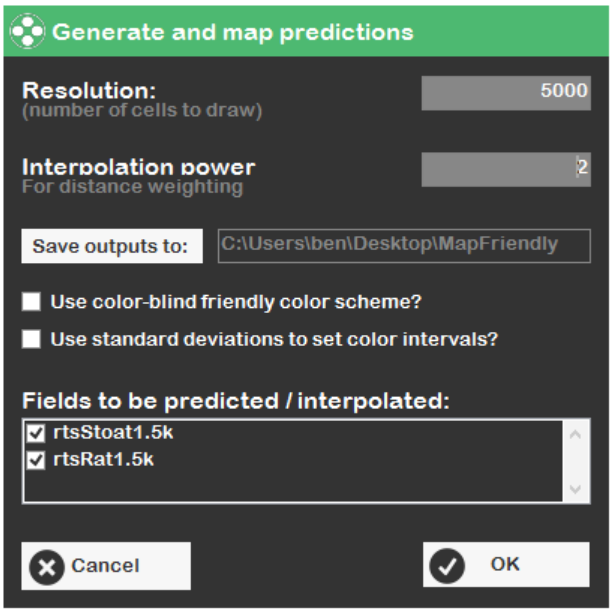
Step 3 Load a set of points containing the relevant variables that the model will be applied to with the Load data to predict button. 

For DOC the prediction data sets are provided with their associated models.

Step 4 Generate predictions by pressing the Generate Predictions button, this will add predicted values to the data and create a raster surface.

Set the resolution of the raster surface you wish to create, the directory where you wish to save the outputs, and the colour scheme of the raster.

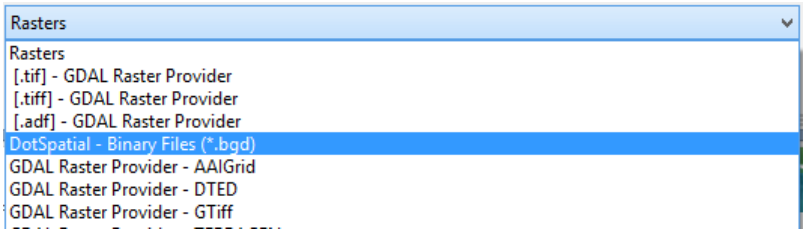




Data requirements

4.5.1..1 Spatial data

Through the GDAL library [Citation], MapFriendly is able to load most common vector and raster formats. Note however, that the spatial tools provided by DotSpatial [citation], in particular, raster related tools will only work with data in the DotSpatial default formats such as binary grid data (.bgd).

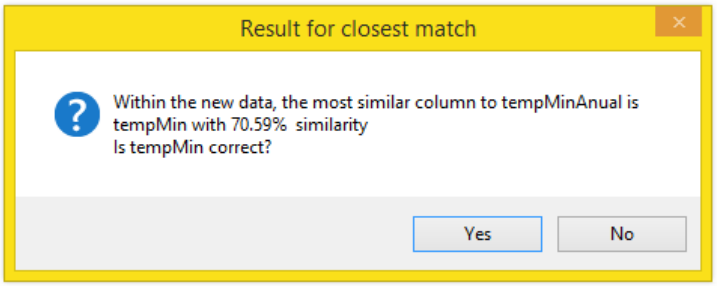


Neural networks

Artificial Neural Networks created in NeuroFriendly may be loaded into MapFriendly, future releases may accept other formats such as simple C objects. To ensure compatibility try to load neural networks created with the version of the NeuroFriendly app bundled with MapFriendly as networks created with newer versions of NeuroFriendly may not be backward compatible.

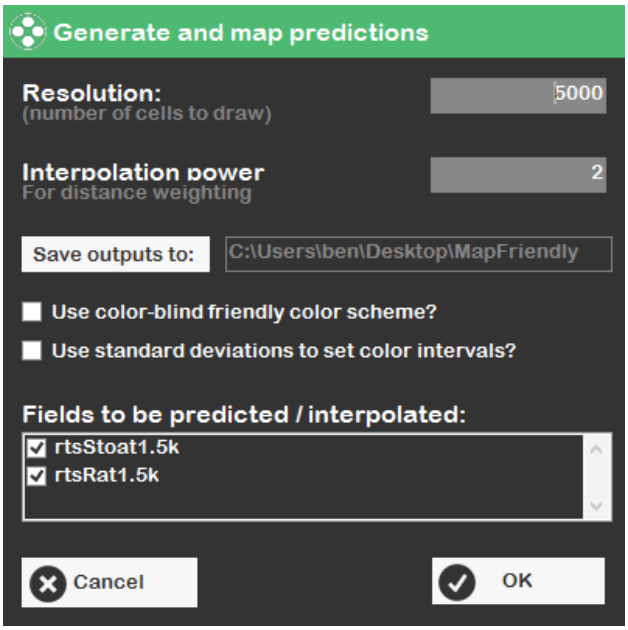
4.5.1..1 Prediction data points

The file containing points for which predictions are desired should be in comma separated (.csv) format. The spatial location of the points must be in decimal latitude and longitude contained in columns called “latitude” and “longitude”. The file must also contain the data for each of the neural network’s inputs with matching column headers. The order is not important, nor is exact spelling; in the event of a typographic error, the neural network will be used to find an approximate match.



Options in depth

4.5.1..1 Generate predictions



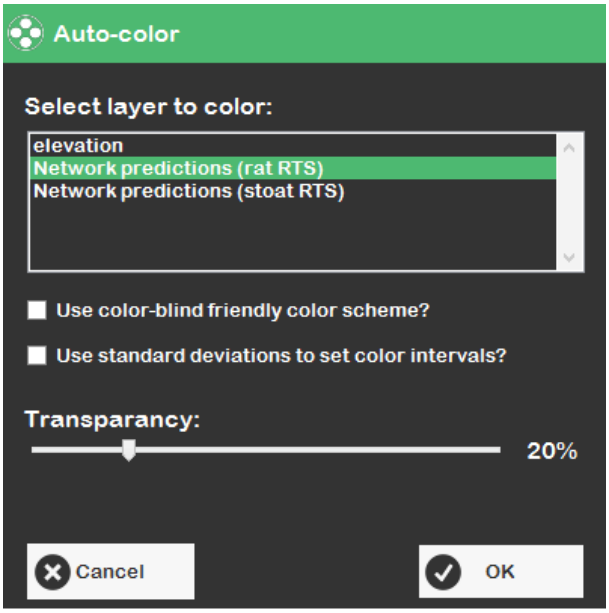
MapFriendly's prediction function applies an artificial neural network to a set of spatially referenced points (prediction data) and interpolates with inverse distance weighting (IDW) to create a raster surface. By default a 5000 cell raster will be created with quadratic powered IDW in a new folder at the directory of the neural network for each of the neural network's outputs. Increasing the resolution will create a smoother image at the expense of computational time. Computation is evenly divided among available CPU cores thus it will vary greatly between machines. For super high resolution images using a computer with more CPU cores can save a significant amount of time. The interpolation power should be either 1 or 2 unless a higher power is specifically needed.

In the output folder, three separate subfolders will be created containing:

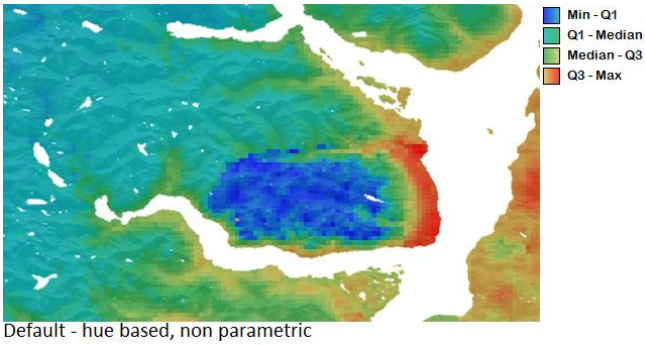
1. A CSV version of the modified prediction data
2. Raster surfaces (as .bgd) and their associated projection (.prj) files. To save the raster(s) as a different format, use the export option from the legend (explained in legend operations).
3. Shapefiles with the predicted values as points (for export to other GIS platforms). This should contain the .shp, .shx, .dbf, and .prj files.

The colour scheme of the prediction outputs that will be displayed can be pre specified. See below (Colourise a raster) for details. The currently loaded neural network is used to automatically populate the list of fields to be predicted. If these names are not what you expect to see, your neural network may have trained on data other than what you intended.

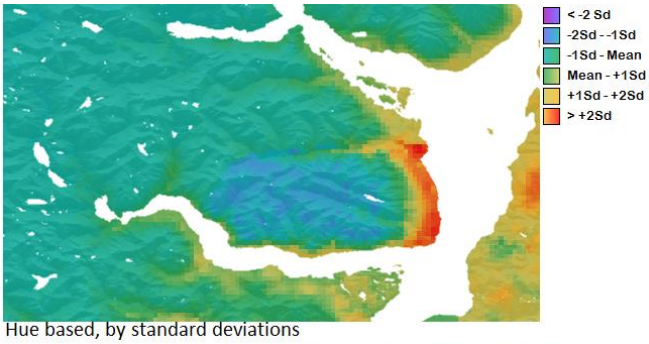
4.5.1..2 Colourise a raster



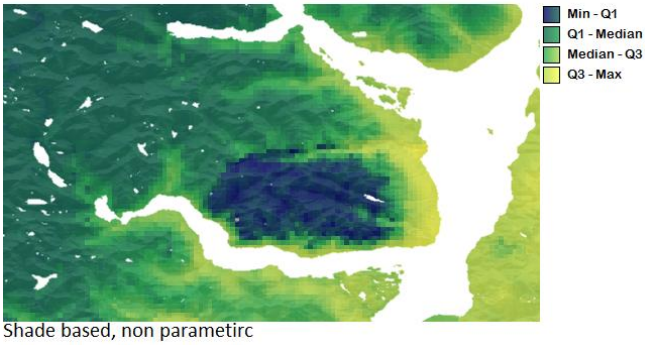
The raster colouriser allows you to (re)apply the default prediction colour schemes to any given raster. By default the output will create a hue-based scheme, for colour blind individuals an explicitly shade-based scheme will be created if "use colour-blind friendly colour scheme?" is ticked.
The default cut-off points for the colour scheme are based on the 25th, 50th (median), and 75th percentiles of the raster surface's values. Alternatively, one and two standard deviations around the mean by ticking "use standard deviations to set colour intervals." Colouring by standard deviation is useful when the raster data (visible from the attribute view) is approximately normally distributed (bell shaped). The colour scheme can easily be modified later via the raster colouriser button. The results of the different colour options are shown below with non-normally distributed data (right skew) at 20% transparency over a hill shade layer:



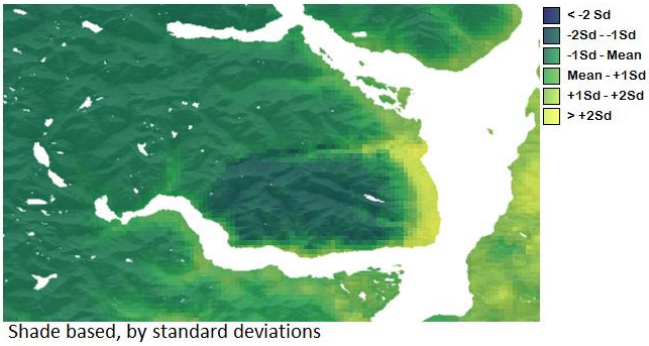
Default - hue based, non parametric



Hue based, by standard deviations



Shade based, non parametirc

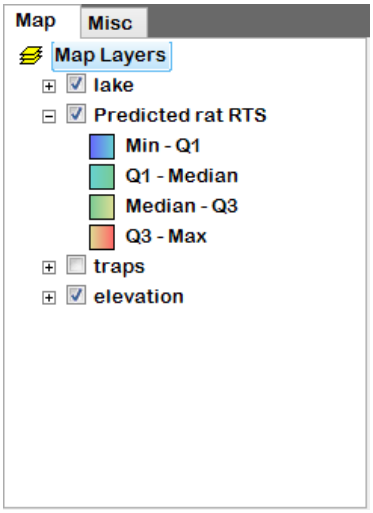


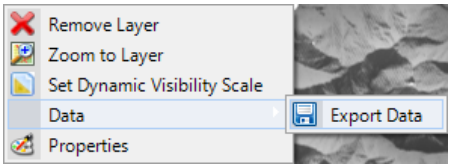
Shade based, by standard deviations

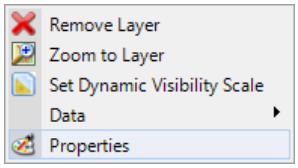
Legend operations

Some options can be reached by right-clicking legend items. The options available vary slightly between raster and vector layers. These options should be self-explanatory and are not needed for MapFriendly's basic operation. Perhaps the most useful are the export and display properties.

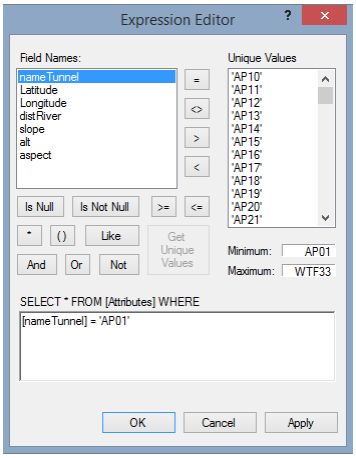
Exporting layers can be performed by right clicking a layer's name on the legend, selecting data, then clicking export data. Exporting raster layers allows you to save copies in other formats such as .tiff for use in other GIS applications. Exporting vector layers allows you to make shapefiles from subsets of a given layer (all features, selected features, or features in view). Graphical properties can be edited by right-clicking a layer and selecting properties. The properties window allows you to change the symbology of a layer as well as viewing the layer's metadata.

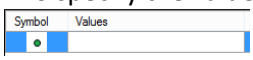




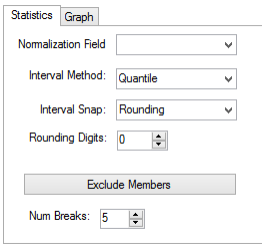


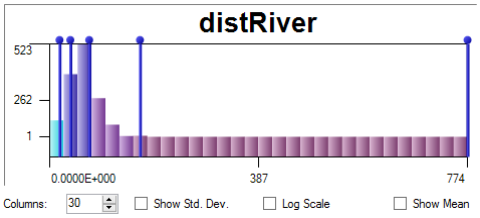
Vector symbology

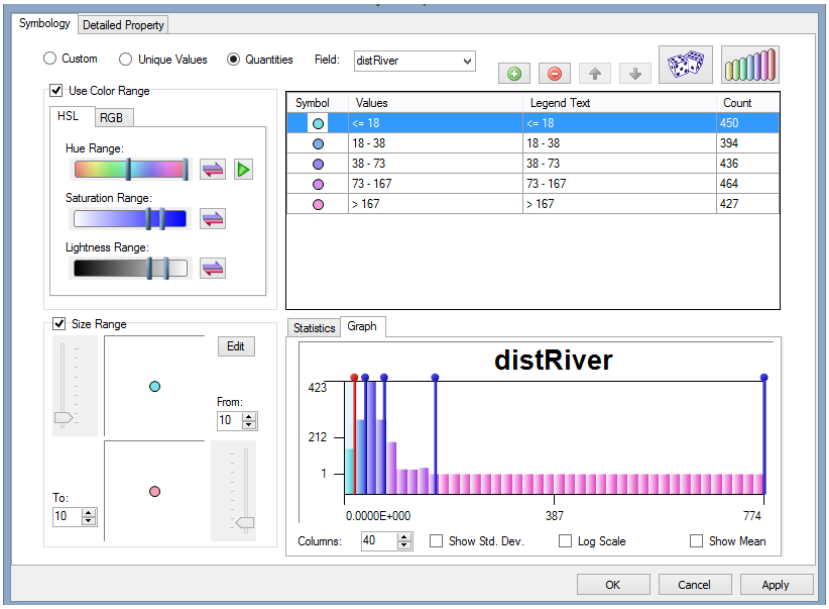


The properties window is largely identical for shapefiles and raster data sets, however, as shapefiles tend to have multiple fields which may be non-numeric, you must specify which field and which items you wish to base your symbology on. To do this, select from either custom (you select which items to colour), unique values (all values get a unique colour and or size), or quantities (colour by numerical categories).
For custom symbology, make sure custom is checked and then add or remove the categories you wish to display with the plus (or minus) buttons  . To specify the values double click the corresponding cell under values. 

This will bring up an expression editor you can use to select specific values from a given field. Double click on the symbol to edit.
If you wish to use unique values, simply check unique values and select a field. The symbol list will automatically be populated. This option is useful when there exist relatively few unique values.
To apply symbology via quantity, simply check quantities and select a numeric field. Random or continuous colour categories can be automatically generated with their respective buttons:  The ramp button will also apply changes made to the colour and/or size ranges. More advanced options for setting symbology categories can be accessed from the statistics and graph tabs. Most notably, the interval method can be changed from equal intervals, quantiles, or manually set intervals, the number of breaks (categories) can also be changed here. Within the graphs tab, you are able to manually drag the breaks to visually adjust your classifications.

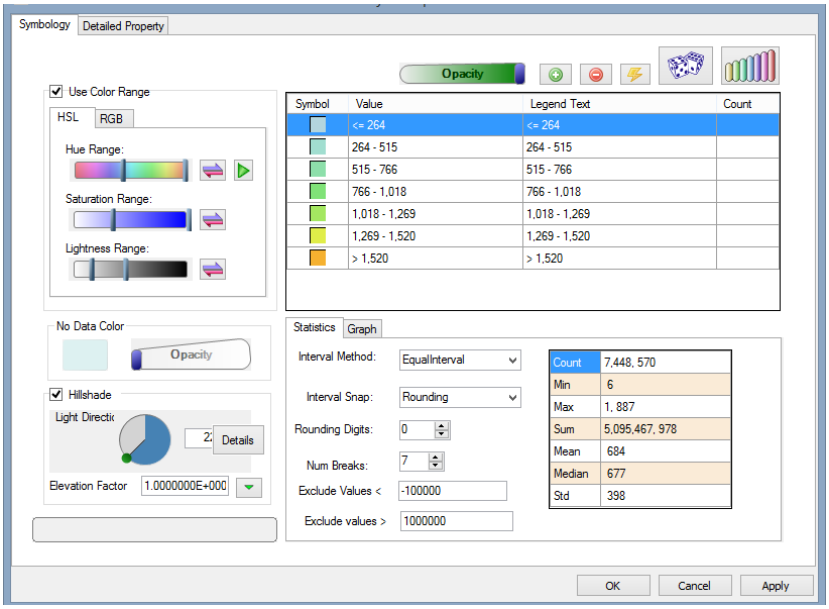


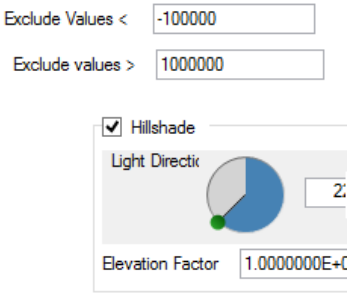




Raster Symbology

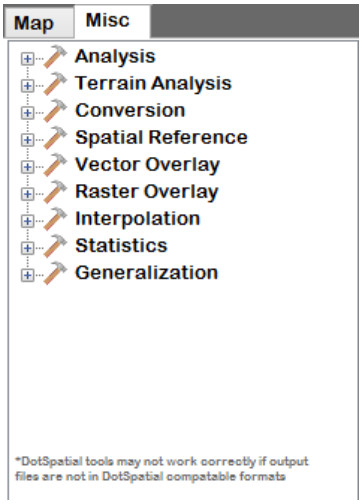
The raster symbology is essentially the same as the vector symbology properties without the option to change field. You can also access other pre-defined raster colour schemes via the  button. To colour values within a specific range, change the thresholds in the exclude values </> boxes in the statistics tab. You can also create a hillshade from elevation data.





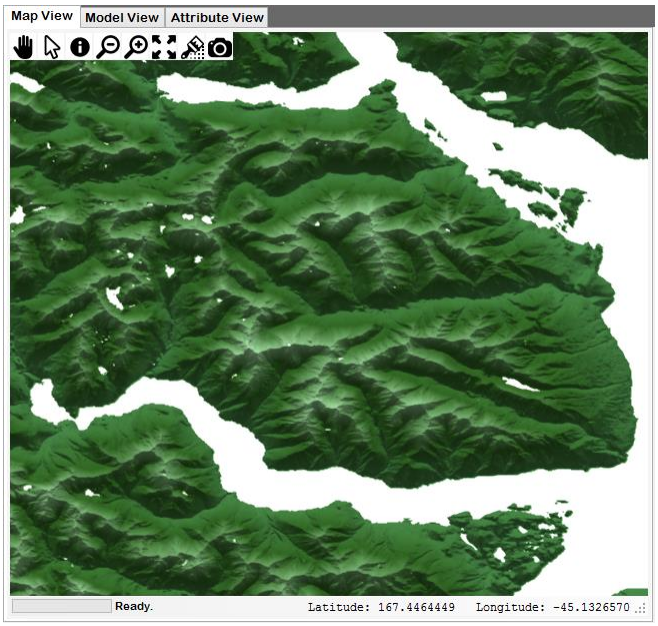
Miscellaneous tools

A set of basic geospatial tools is provided with the DotSpatial library. From here some basic GIS operations can be performed such as creating buffers, clipping features, etc. As these tools are not created as a part of MapFriendly, updates to MapFriendly are highly unlikely to include updates to these tools.
One should be careful to save outputs in default DotSpatial formats: shapefiles or binary grid data. For specific help with each tool, refer to the information panel from within the tool.



Display tabs

4.5.1..1 Map view



The map view is essentially the main interface of MapFriendly displaying the map and map controls.

The pan tool – use this to navigate the map by dragging it about (default cursor).

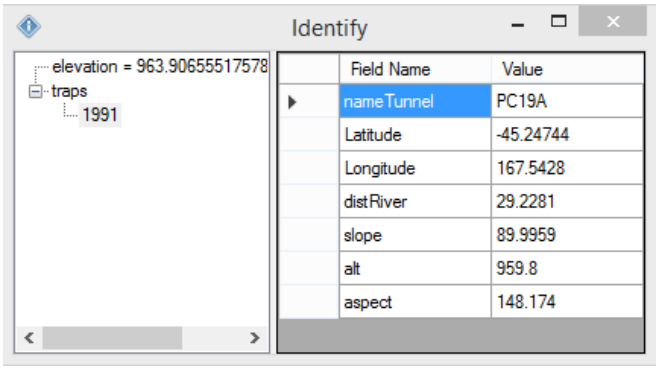
The selection tool – use this to make the cursor select features. Hold control for multiple selection.

The identity tool – use this to view the properties of all layers at a given point upon clicking a portion of a map.

Zoom out, zoom in, and zoom to the map's full extent. Zooming can also be performed by scrolling the mouse wheel while panning the map.

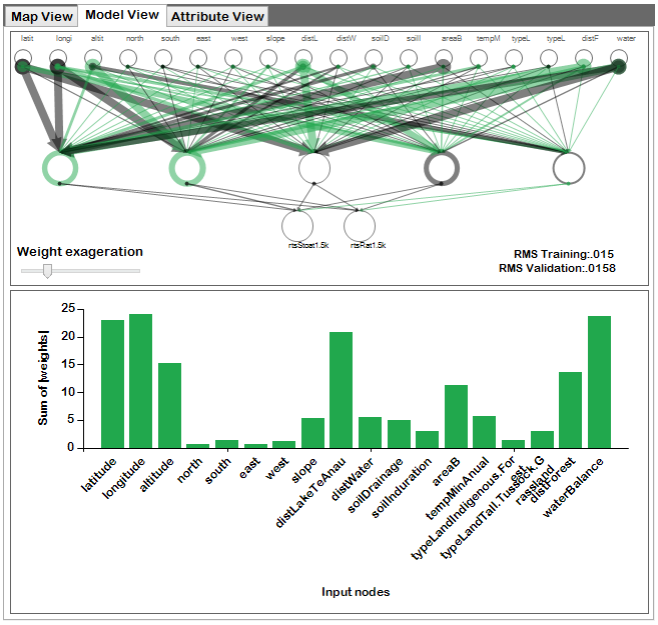
Launch the raster colouriser to apply the default prediction colour schemes to a raster layer.

Open image export and printing menu to add geospatial items such as north arrows and scale bars to your map and either export it as an image or send it to a printer.



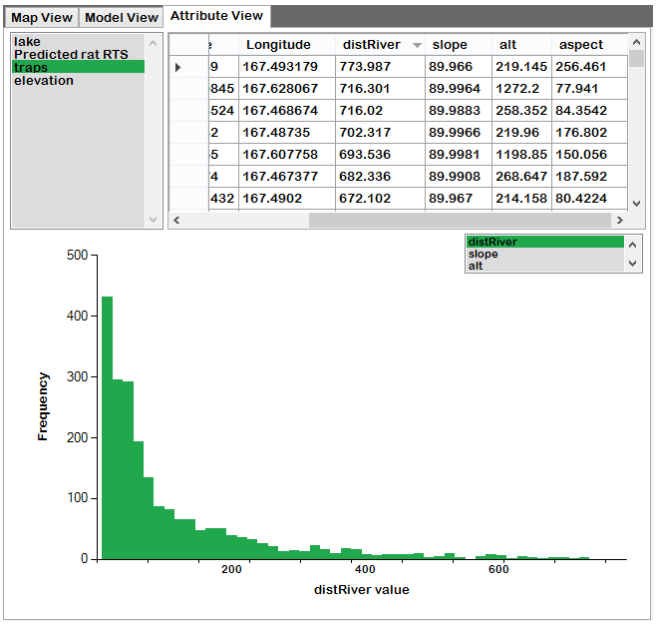
4.5.1..2 Model view

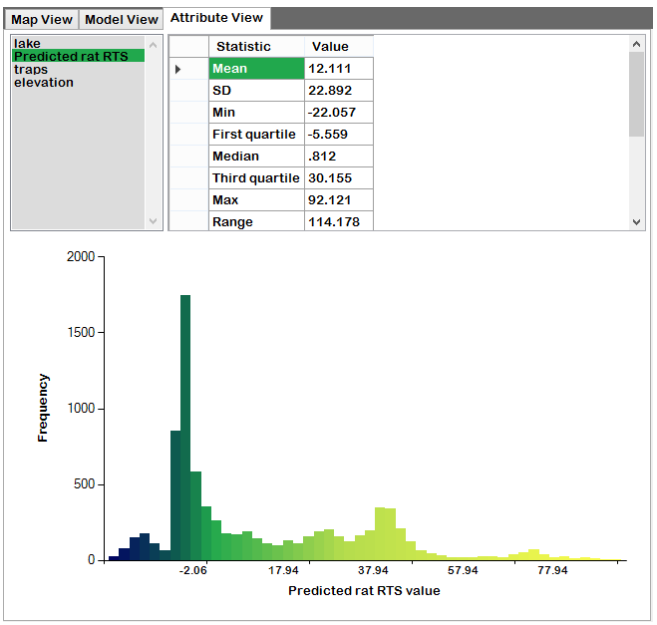
This tab allows you to inspect the currently loaded neural network along with its recorded RMS training and validation errors. On the network display, black synapses and nodes are those with positive weights and biases, green represents negative values. The weights can be exaggerated to scale the line width proportional to the weights. The column graph of absolute input-to-hidden node weights gives a rough idea as to which variables are the most important. Categorical (dummy) variables will have their contribution divided between categories.



4.5.1..3 Attribute view

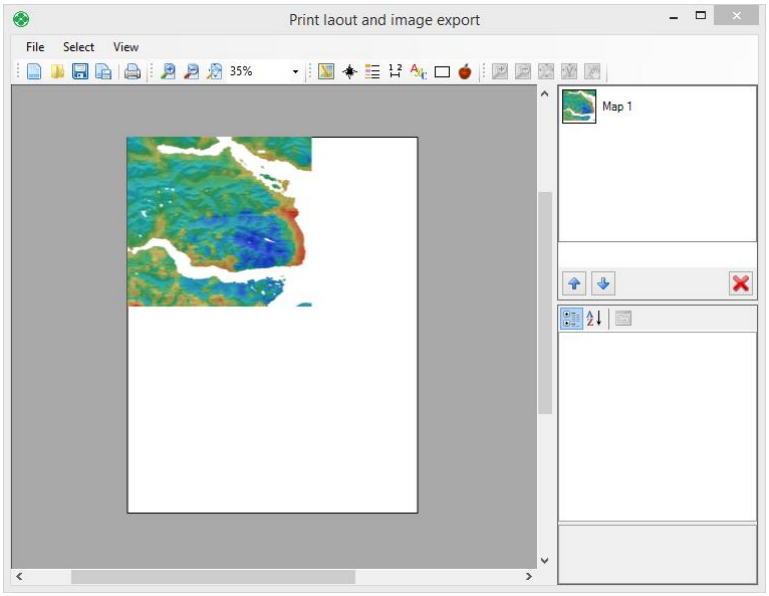
The attribute view allows you to quickly inspect attribute tables for vector layers (left), statistics for raster layers (right), and histograms of numerical data. For vector layers, select a field to graph from the list that appears on the top-right hand side of the histogram.



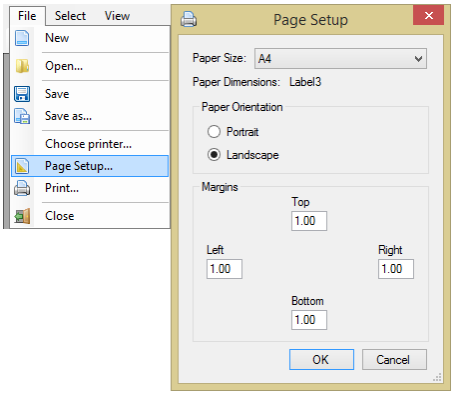


Export and print your map

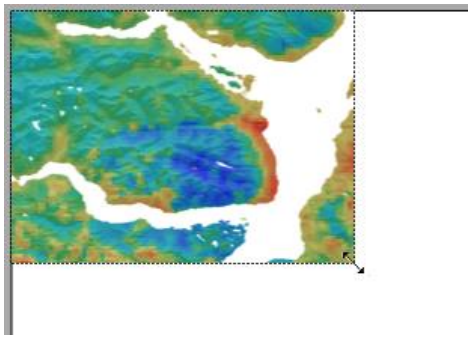
Exporting and printing your map as an image is relatively straight forward, however, some trial-and-error will probably be needed to create the map image you want. If you have used other GIS software, such as ArcGIS, formatting the look and layout of the map will be familiar to you. Upon first opening the print and layout window, the map will almost certainly not look how you want it. Here, we will cover the basic operations and features used to create a more useful map. The following steps need not be performed in order, they are meant as a simple tutorial to give you a feel for the available options.



Step 1 Change the page orientation to landscape. By selecting Page Setup under the file menu. Tick Landscape and set the page size to A4 for easy insertion into documents and print scaling.



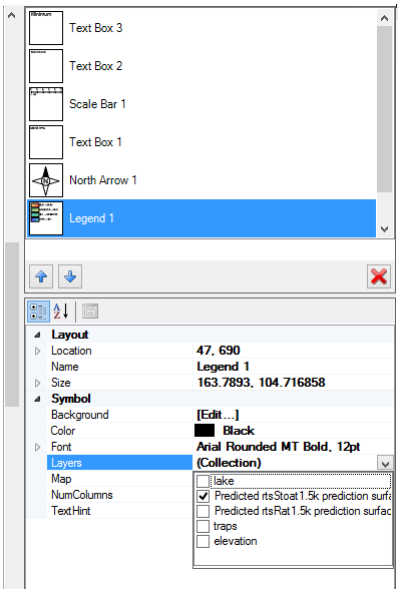
Step 2 Make the map frame fill more of the page by simply clicking on the map image and dragging the borders to the desired size. You may also drag the map image around the page or right-click and use the margin alignment or margin fit options



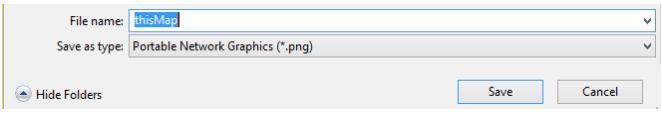
Step 3 To move and adjust the map within the map frame, use the map tools on the top-right  these tools allow you to zoom in, zoom out, zoom to the full extent, zoom to the extent shown in the map view tab, or to pan about the map. To use these tools you must select the map (they will appear in grey if the map is not selected). For the pan tool, you must un-select it when you are finished moving the map within the frame.

Step 4 Insert map objects such as additional inset map frames (not demonstrated here), north arrows, legends, scale bars, lables, rectangle frames, or bitmap images (not demonstrated here). To do this, select the map and use the tools located at the top-center of the window  each object will first need to be drawn, simply drag and drop the rectangle to set the initial size and possition of each object. These can be altered later.

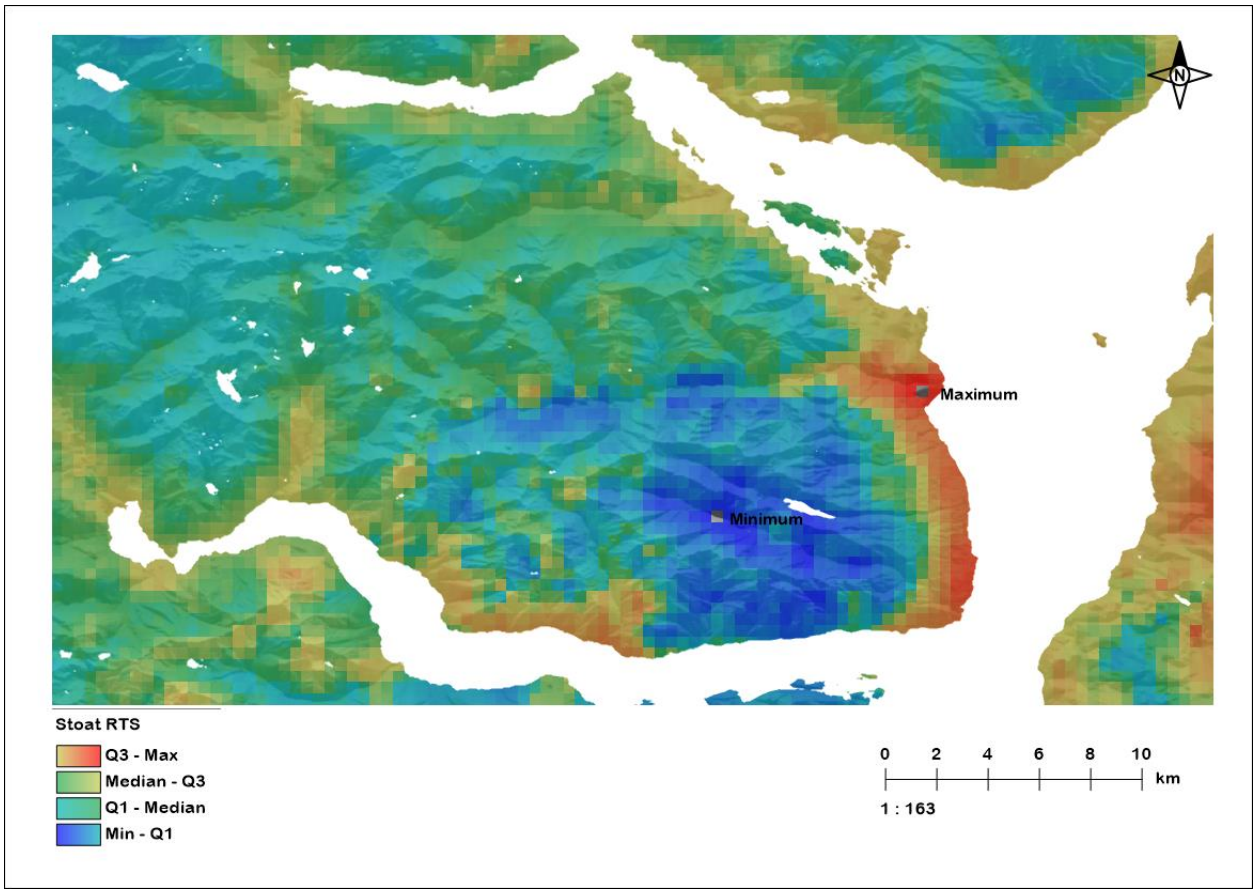
Step 5 Fine tuning objects and selecting objects (now) contained within another map frame can be done using the panels on the left hand side. For example. To change which layers appear on the legend, select the legend (upper pane), then in the properties box (lower pane), click on layers and tick the layers you wish to display. You can also use the properties pane to edit the text shown in text boxes, the number of breaks shown in the scale bar, etc. To select objects that have been placed on top of another object (like the map frame) which may interfere with selecting and moving said object, use the upper pane to select the object and/or move the object above or below other object



Step 6 Saving, exporting and printing your map can be performed inside the file menu. You may either save the map as  a layout for later use as a .mwl DotSpatial layout file or save it directly to an image file in the protable network graphics (.png) format.



You should now be able to print your map directly, or use it in other doccuments as a picture file. Feel free to explore the different options available. By saving your layout file (.mwl) as you work, you do not need to worry about “ruining” your map. Alternatively, if you are more comfortable with software such as ArcGIS, simply export the data layers and finalise your map that way.



References

Adams, L. W. (1984). Small Mammal use of an Interstate Highway Median Strip. *Journal of Applied Ecology, 21*(1), 175-178. doi: 10.2307/2403045

Alterio, N. (1998). Spring home range, spatial organisation and activity of stoats Mustela erminea in a South Island Nothofagus forest, New Zealand. *Ecography, 21*(1), 18-24. doi: 10.1111/j.1600-0587.1998.tb00390.x

Anderson, D. R. (2008). *Model based inference in the life sciences: a primer on evidence*: Springer.

Anderson, D. R., & Burnham, K. P. (2002). Avoiding Pitfalls When Using Information-Theoretic Methods. *The Journal of Wildlife Management, 66*(3), 912-918. doi: 10.2307/3803155

Ballance, A. (2001). Takahe: the bird that twice came back from the grave. *The Takahe: Fifty Years of Conservation Management and Research*, 18-22.

Barry, S. C., & Welsh, A. H. (2002). Generalized additive modelling and zero inflated count data. *Ecological Modelling, 157*(2–3), 179-188. doi: http://dx.doi.org/10.1016/S0304-3800(02)00194-1

Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods, 43*(1), 3-31. doi: http://dx.doi.org/10.1016/S0167-7012(00)00201-3

Blackwell, G. L., Potter, M. A., McLennan, J. A., & Minot, E. O. (2003). The role of predators in ship rat and house mouse population eruptions: drivers or passengers? *Oikos, 100*(3), 601-613. doi: 10.1034/j.1600-0706.2003.11026.x

Bolker, B. M., Brooks, M. E., Clark, C. J., Geange, S. W., Poulsen, J. R., Stevens, M. H. H., & White, J.-S. S. (2009). Generalized linear mixed models: a practical guide for ecology and evolution. *Trends in Ecology & Evolution, 24*(3), 127-135. doi: http://dx.doi.org/10.1016/j.tree.2008.10.008

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123-140.

Bunnefeld, N., Hoshino, E., & Milner-Gulland, E. J. (2011). Management strategy evaluation: a powerful tool for conservation? *Trends in Ecology & Evolution, 26*(9), 441-447. doi: http://dx.doi.org/10.1016/j.tree.2011.05.003

Carnes, E. (1931). Rat-flea survey of the port of St. Thomas, Virgin Islands. *Public Health Reports (1896-1970)*, 2558-2562.

Caut, S., Casanovas, J. G., Virgos, E., Lozano, J., Witmer, G. W., & Courchamp, F. (2007). Rats dying for mice: Modelling the competitor release effect. *Austral Ecology, 32*(8), 858-868. doi: 10.1111/j.1442-9993.2007.01770.x

Christie, J., Kemp, J., Rickard, C., & Murphy, E. (2006). Measuring stoat (Mustela erminea) and ship rat (Rattus rattus) capture success against micro-habitat factors. *New Zealand Journal of Ecology, 30*(1), 43-51.

Courchamp, F., Langlais, M., & Sugihara, G. (1999). Cats protecting birds: modelling the mesopredator release effect. *Journal of Animal Ecology, 68*(2), 282-292. doi: 10.1046/j.1365-2656.1999.00285.x

Courchamp, F., Langlais, M., & Sugihara, G. (2000). Rabbits killing birds: modelling the hyperpredation process. *Journal of Animal Ecology, 69*(1), 154-164. doi: 10.1046/j.1365-2656.2000.00383.x

Dice, L. R. (1931). Methods of indicating the abundance of mammals. *Journal of Mammalogy, 12*(4), 376-381.

Dowding, J. E., & Murphy, E. C. (1994). Ecology of ship rats (*Rattus rattus*) in a kauri (*Agathis australis*) forest in Northland, New Zealand. *New Zealand Journal of Ecology, 18*(1), 19-28.

Drever, M. C., & Lewis, J. C. (2004). Capture rates of Norway Rats (*Rattus Norvegicus*) in a seabird colony: a caveat for investigators. *Northwestern Naturalist, 85*(3), 111-117. doi: 10.1898/1051-1733(2005)085[0111:CRONRR]2.0.CO;2

Eddelbuettel, D., & Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *2011, 40*(8), 18. doi: 10.18637/jss.v040.i08

Ernest, S. K. M., Brown, J. H., & Parmenter, R. R. (2000). Rodents, plants, and precipitation: spatial and temporal dynamics of consumers and resources. *Oikos, 88*(3), 470-482. doi: 10.1034/j.1600-0706.2000.880302.x

ESRI. (2011). ArcGIS Desktop: Release 10.1 (Version 10.1): Environmental Systems Research Institute.

ESRI. (2014). ArcGIS Desktop (Version 10.2). Redlands, CA: Environmental Systems Research Institute.

Evans, M. R., Grimm, V., Johst, K., Knuuttila, T., de Langhe, R., Lessells, C. M., . . . Benton, T. G. (2013). Do simple models lead to generality in ecology? *Trends in Ecology & Evolution*(0). doi: http://dx.doi.org/10.1016/j.tree.2013.05.022

Forsyth, D. M., Link, W. A., Webster, R., Nugent, G., & Warburton, B. (2005). Nonlinearity and seasonal bias in an index of brushtail possum abundance. *Journal of Wildlife Management, 69*(3), 976-984.

Freund, Y., & Schapire, R. E. (1995). *A desicion-theoretic generalization of on-line learning and an application to boosting.* Paper presented at the Computational learning theory.

Griffiths, I., Adams, M., & Liberty, J. (2013). *Programming C# 5.0 early release*: O'Reilly.

Grinnell, J. (1914). *An account of the mammals and birds of the lower Colorado Valley: With especial reference to the distributional problems presented* (Vol. 12): University of California Press.

Hegg, D., Greaves, G., Maxwell, J. M., MacKenzie, D. I., & Jamieson, I. G. (2012). Demography of takahe (*Porphyrio hochstetteri*) in Fiordland: environmental factors and management affect survival and breeding success. *New Zealand Journal of Ecology, 36*(1), 75.

Hegg, D., MacKenzie, D. I., & Jamieson, I. G. (2013). Use of Bayesian population viability analysis to assess multiple management decisions in the recovery programme for the Endangered takahe Porphyrio hochstetteri. *Oryx, 47*(01), 144-152.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hitchmough, R., Bull, L., & Cromarty, P. (2007). *New Zealand Threat Classification System lists: 2005*: Department of Conservation.

Holling, C. S. (1978). Adaptive environmental assessment and management. *Adaptive environmental assessment and management.*

Johnson, C. R., Moorhead, R., Munzner, T., Pfister, H., Rheingans, P., & Yoo, T. S. (2011). NIH/NSF Visualization Research Challenges Report.

Johnson, J. B., & Omland, K. S. (2004). Model selection in ecology and evolution. *Trends in Ecology & Evolution, 19*(2), 101-108. doi: http://dx.doi.org/10.1016/j.tree.2003.10.013

Joy, M. K., & Death, R. G. (2004). Predictive modelling and spatial mapping of freshwater fish and decapod assemblages using GIS and neural networks. *Freshwater Biology, 49*(8), 1036-1052. doi: 10.1111/j.1365-2427.2004.01248.x

Keith, D. A., Martin, T. G., McDonald-Madden, E., & Walters, C. (2011). Uncertainty and adaptive management for biodiversity conservation. *Biological Conservation, 144*(4), 1175-1178. doi: http://dx.doi.org/10.1016/j.biocon.2010.11.022

King, C. (1983). The relationships between beech (Nothofagus sp.) seedfall and populations of mice (Mus musculus), and the demographic and dietary responses of stoats (Mustela erminea), in three New Zealand forests. *The Journal of Animal Ecology*, 141-166.

Lankin Vega, G. (2002). Neural network models for the prediction of autumn migration of the cereal aphid Rhopalosiphum padi at Lincoln, Canterbury, New Zealand.

Le Corre, M. (2008). Conservation biology: Cats, rats and seabirds. *Nature, 451*(7175), 134-135.

Lee, W. G., & Jamieson, I. G. (2001). *The takahē: fifty years of conservation management and research*: Univ of Otago Pr.

Lek, S., Delacoste, M., Baran, P., Dimopoulos, I., Lauga, J., & Aulagnier, S. (1996). Application of neural networks to modelling nonlinear relationships in ecology. *Ecological Modelling, 90*(1), 39-52. doi: http://dx.doi.org/10.1016/0304-3800(95)00142-5

Levin, R. I., Lieven, N. A. J., & Lowenberg, M. H. (2000). Measuring and improving neural network generalization for model updating. *Journal of Sound and Vibration, 238*(3), 401-424. doi: http://dx.doi.org/10.1006/jsvi.2000.3105

Long, P. M., & Servedio, R. A. (2010). Random classification noise defeats all convex potential boosters. *Machine Learning, 78*(3), 287-304.

Loraine, A., & Helt, G. (2002). Visualizing the genome: techniques for presenting human genome data and annotations. *BMC Bioinformatics, 3*(1), 19.

Loyd, K. A. T., & DeVore, J. L. (2010). An evaluation of feral cat management options using a decision analysis network. *Ecology and Society, 15*(4), 10.

Maletic, J. I., & Marcus, A. (2005). Data Cleansing *Data Mining and Knowledge Discovery Handbook* (pp. 21-36): Springer.

Mandel, T. (1997). *The Elements of User Interface Design*. Canada: John Wiley & Sons, Inc.

Maxwell, J. (2001). Fiordland takahe: population trends, dynamics and problems. *The Takahe—fifty years of conservation management and research. University of Otago Press, Dunedin, New Zealand*, 61-79.

McCaffery, J. (2014). Neural Network Dropout Training -- Visual Studio Magazine. Retrieved 2 Jun., 2014, from http://visualstudiomagazine.com/articles/2014/05/01/neural-network-dropout-training.aspx

Mills, J. A., & Lavers, R. (1974). *Preliminary Results of Research Into the Present Status of Takahe (Notornis Mantelli), in the Murchison Mountains*: New Zealand Department of Internal Affairs.

Murphy, E., & Bradfield, P. (1992). Change in diet of stoats following poisoning of rats in a New Zealand forest. *New Zealand Journal of Ecology, 16*, 137-137.

Murphy, E. C., Clapperton, B. K., Bradfield, P. M. F., & Speed, H. J. (1998). Effects of rat-poisoning operations on abundance and diet of mustelids in New Zealand podocarp forests. *New Zealand Journal of Zoology, 25*(4), 315-328. doi: 10.1080/03014223.1998.9518161

Murphy, E. C., & Dowding, J. E. (1994). Range and diet of stoats (*Mustela erminea*) in a New Zealand beech forest. *New Zealand Journal of Ecology, 18*(1), 11-18.

Nelson, L., & Clark, F. W. (1973). Correction for sprung traps in catch/effort calculations of trapping results. *Journal of Mammalogy*, 295-298.

Orzack, S. H., & Sober, E. (1993). A Critical Assessment of Levins's The Strategy of Model Building in Population Biology (1966). *The Quarterly Review of Biology, 68*(4), 533-546. doi: 10.2307/3037250

Özesmi, S. L., Tan, C. O., & Özesmi, U. (2006). Methodological issues in building, training, and testing artificial neural networks in ecological applications. *Ecological Modelling, 195*(1–2), 83-93. doi: http://dx.doi.org/10.1016/j.ecolmodel.2005.11.012

Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks, 11*(4), 761-767. doi: http://dx.doi.org/10.1016/S0893-6080(98)00010-0

R Core Team. (2014). R: a language and environment for statistical computing (version 3.1. 0). Vienna: R Foundation for Statistical Computing. *Available at ht tp://www. R-project. org*.

R Development Core Team. (2012). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2007: ISBN 3-900051-07-0.

Rayner, M. J., Hauber, M. E., Imber, M. J., Stamp, R. K., & Clout, M. N. (2007). Spatial heterogeneity of mesopredator release within an oceanic island system. *Proceedings of the National Academy of Sciences, 104*(52), 20862-20865. doi: 10.1073/pnas.0707414105

Razi, M. A., & Athappilly, K. (2005). A comparative predictive analysis of neural networks (NNs), nonlinear regression and classification and regression tree (CART) models. *Expert Systems with Applications, 29*(1), 65-74. doi: http://dx.doi.org/10.1016/j.eswa.2005.01.006

Reed, R., Marks, R. J., II, & Oh, S. (1995). Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter. *Neural Networks, IEEE Transactions on, 6*(3), 529-538. doi: 10.1109/72.377960

Reed, R., Oh, S., & Marks, R. (1992). *Regularization using jittered training data.* Paper presented at the Neural Networks, 1992. IJCNN., International Joint Conference on.

Riedmiller, M., & Braun, H. (1993, 1993). *A direct adaptive method for faster backpropagation learning: the RPROP algorithm.* Paper presented at the Neural Networks, 1993., IEEE International Conference on.

Ritchie, E. G., & Johnson, C. N. (2009). Predator interactions, mesopredator release and biodiversity conservation. *Ecology Letters, 12*(9), 982-998.

Rodríguez, J. P., Brotons, L., Bustamante, J., & Seoane, J. (2007). The application of predictive modelling of species distribution to biodiversity conservation. *Diversity and Distributions, 13*(3), 243-251. doi: 10.1111/j.1472-4642.2007.00356.x

Roemer, G. W., Coonan, T. J., Garcelon, D. K., Bascompte, J., & Laughrin, L. (2001). Feral pigs facilitate hyperpredation by golden eagles and indirectly cause the decline of the island fox. *Animal Conservation, 4*(4), 307-318. doi: 10.1017/S1367943001001366

Rozas, A. (2008). XML serialisation of complex .net objects. *CodeProject.* Retrieved 01/05, 2014, from http://www.codeproject.com/Articles/30270/XML-Serialization-of-Complex-NET-Objects

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533-536.

Ruscoe, W. A., Ramsey, D. S. L., Pech, R. P., Sweetapple, P. J., Yockney, I., Barron, M. C., . . . Duncan, R. P. (2011). Unexpected consequences of control: competitive vs. predator release in a four-species assemblage of invasive mammals. *Ecology Letters, 14*(10), 1035-1042. doi: 10.1111/j.1461-0248.2011.01673.x

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197-227.

Smith, A. P., & Quin, D. G. (1996). Patterns and causes of extinction and decline in Australian conilurine rodents. *Biological Conservation, 77*(2–3), 243-267. doi: http://dx.doi.org/10.1016/0006-3207(96)00002-X

Steyerberg, E. W., Harrell Jr, F. E., Borsboom, G. J. J. M., Eijkemans, M. J. C., Vergouwe, Y., & Habbema, J. D. F. (2001). Internal validation of predictive models: Efficiency of some procedures for logistic regression analysis. *Journal of Clinical Epidemiology, 54*(8), 774-781. doi: http://dx.doi.org/10.1016/S0895-4356(01)00341-9

Suryanarayana, I., Braibanti, A., Sambasiva Rao, R., Ramam, V. A., Sudarsan, D., & Nageswara Rao, G. (2008). Neural networks in fisheries research. *Fisheries Research, 92*(2–3), 115-139. doi: http://dx.doi.org/10.1016/j.fishres.2008.01.012

Tan, C. O., Özesmi, U., Beklioglu, M., Per, E., & Kurt, B. (2006). Predictive models in ecology: Comparison of performances and assessment of applicability. *Ecological Informatics, 1*(2), 195-211. doi: http://dx.doi.org/10.1016/j.ecoinf.2006.03.002

Tetko, I. V., Livingstone, D. J., & Luik, A. I. (1995). Neural network studies. 1. Comparison of overfitting and overtraining. *Journal of chemical information and computer sciences, 35*(5), 826-833.

Trewick, S., & Worthy, T. (2001). Origins and prehistoric ecology of takahe based on morphometrics, molecular, and fossil data. *The takahe: fifty years of conservation management and research. University of Otago Press, Dunedin*, 31-48.

Vezhnevets, A., & Vezhnevets, V. (2005). *Modest AdaBoost-teaching AdaBoost to generalize better.* Paper presented at the Graphicon.

Walters, C. J., & Hilborn, R. (1978). Ecological optimization and adaptive management. *Annual Review of Ecology and Systematics*, 157-188.

Westgate, M. J., Likens, G. E., & Lindenmayer, D. B. (2013). Adaptive management of biological systems: A review. *Biological Conservation, 158*(0), 128-139. doi: http://dx.doi.org/10.1016/j.biocon.2012.08.016

Whittingham, M. J., Stephens, P. A., Bradbury, R. B., & Freckleton, R. P. (2006). Why do we still use stepwise modelling in ecology and behaviour? *Journal of Animal Ecology, 75*(5), 1182-1189. doi: 10.1111/j.1365-2656.2006.01141.x

Wilson, D. R., & Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks, 16*(10), 1429-1451.

Worner, S., Lankin, G., Samarasinghe, S., & Teulon, D. (2002). Improving prediction of aphid flights by temporal analysis of input data for an artificial neural network. *New Zealand Plant Protection*, 312-316.

Zhang, G. P. (2007). A neural network ensemble method with jittered training data for time series forecasting. *Information Sciences, 177*(23), 5329-5346. doi: http://dx.doi.org/10.1016/j.ins.2007.06.015

Zhou, Z.-H., Wu, J., & Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artificial Intelligence, 137*(1–2), 239-263. doi: http://dx.doi.org/10.1016/S0004-3702(02)00190-X