WEIGHTED MACHINE LEARNING FOR SPATIAL-TEMPORAL DATA

by

Mahdi Hashemi

B. S. in Surveying Engineering, University of Tabriz, Tabriz, 2009

M. S. in GIS Engineering, K. N. Toosi University of Technology, Tehran, 2011

Submitted to the Graduate Faculty of the School of

Computing and Information in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

Mahdi Hashemi

It was defended on

September 25, 2017

and approved by

Hassan A. Karimi, Professor, School of Computing and Information

Prashant Krishnamurthy, Professor, School of Computing and Information

Ching-Chung Li, Professor, School of Electrical and Computer Engineering

Paul Munro, Associate Professor, School of Computing and Information

Dissertation Advisor: Hassan A. Karimi, Professor, School of Computing and Information

WEIGHTED MACHINE LEARNING FOR SPATIAL-TEMPORAL DATA

Mahdi Hashemi, PhD

University of Pittsburgh, 2017

Sometimes not all training samples are equal in supervised machine learning due to their different accuracy, reliability, source, relevance, or other reasons. Non-weighted machine learning techniques are designed for equally important training samples: (a) the cost of misclassification is equal for training samples in parametric classification techniques, (b) residuals are equally important in parametric regression models, and (c) when voting in non-parametric classification and regression models, training samples either have equal weights or their weights are determined internally by kernels in the feature space, thus no external weights. In this thesis, we develop the weighted versions of Bayesian predictor, perceptron, multilayer perceptron, SVM, and decision tree and show how their results would be different from their non-weighted versions.

Applying machine learning techniques to spatial-temporal data poses the question that how the recorded location and time for training samples should contribute to the training and testing process. The prior knowledge of how spatial-temporal phenomena are autocorrelated cannot be properly captured by machine learning techniques which either ignore location and time altogether, or consider them as input features. Not to mention that the latter approach leads to increased sparseness of data in the feature space and more free parameters in the predictor;

thus demanding for larger training datasets. We use the prior knowledge about the spatial-temporal autocorrelation to determine how relevant each training sample would be, given its spatial and temporal distances to the irresponsive (unlabeled) sample. Weighted machine learning techniques use this prior knowledge by taking the relevance of training samples with regard to the irresponsive sample into account as training samples' weights. The proposed approach overcomes the aforementioned issues by enriching the training process with the prior knowledge about spatial-temporal autocorrelation. Because the spatial-temporal weight of training samples depends on the irresponsive sample's location and time, the machine needs to be trained separately for each irresponsive sample. However, we show that in practice using only a small subset of training samples with largest spatial-temporal weights not only mitigates the training time but also results in the best accuracy in most cases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I have been bestowed with the great privilege to learn from numerous gracious teachers during the 12 years of school, followed by 12 years in academia. I take this opportunity to express my sincere gratitude and respect to all my teachers for their unsparing effort in pulling me up the ladder of science step by step. Their role in shaping my career, life, and understanding of the world goes beyond words.

I would like to thank my PhD adviser of four years, Dr. Karimi, whose guidance shed light on the path to my achievements, whose support was there when I needed, and whose mentoring and advices will accompany me throughout my career.

I thank Dr. Munro for his contribution in both my PhD dissertation and comprehensive examination. I thank Dr. Li for taking the time to be on my PhD committee and the interesting course I passed with him. I thank Dr. Krishnamurthy for sparing the time to help me with my dissertation. I thank them all for offering their valuable comments to improve my dissertation.

I could not be where I am, without the sacrifices my mother made on my behalf. I dedicate this work to my mother and sister, Fariba and Parvin, for their unconditional support, trust, and love.

1.0 INTRODUCTION

1.1 DEFINITIONS AND SYMBOLOGY

In this thesis, for the purpose of clarification and unification:

- We refer to the machine learning technique's inputs as features or feature vector. They are also called predictors, independent, or explanatory variables in some literature.
- If the output of a machine learning technique is continuous, it is a regressor, otherwise, if the output is categorical, it is a classifier. Predictor refers to both of them.
- The terms estimate and label are used to refer to the output of regressors and classifiers, respectively, and the term response is used to refer to the output of either.
- If the response is known for an observation, it is called a responsive sample/point/observation and if the response is not known (only the feature vector is observed), it is called an irresponsive sample/point/observation.
- Training samples (responsive or irresponsive) are the samples used to train the predictor and test samples are the ones used to test its accuracy. Note that, although this thesis is only focused on supervised learning, we avoid using the terms training and test samples to distinguish between samples with and without responses because unsupervised (e.g., autoencoders) and hybrid (e.g., pre-trained MLPs) learners are trained by irresponsive samples [1].

- We represent the number of features or the length of the feature vector with l and the number of training samples with N.
- Matrices are shown with uppercase and vectors with lowercase letters.
- The feature vector of the i-th training sample is represented by x_i which is a $l \times 1$ vector and its response, either numerical or categorical, is shown with y_i.
- The features or inputs matrix is represented by X, where each row is one feature vector. Therefore, X is a $N \times l$ matrix where N is the number of training samples and l is the number of features.
- The bold $\boldsymbol{x}_i$ indicates a feature vector with an additional element of 1 at its end; therefore, $\boldsymbol{x}_i$ has $l+1$ elements.
- The bold $\boldsymbol{X}$ indicates the features matrix with an additional column of length N with all its elements being 1; therefore, $\boldsymbol{X}$ has N rows and $l+1$ columns. Adding this column to the features matrix simplifies the calculation of intercept for some machine learning techniques.
- The vector containing all training sample responses is indicated by y which is a vector of length N.
- The weight vector for the linear predictors is shown with w and the threshold (intercept) with w_0. The bold $\boldsymbol{w}$ shows the weight vector including the threshold (intercept) as its last element.
- The weight for the training sample i is denoted by g_i. Without the subscript, g represents all weights in one $N \times 1$ vector and G indicates the $N \times N$ diagonal matrix including g as its diagonal.
- The identity matrix is represented with I.

1.2 PROBLEM STATEMENT

“Data from locations near one another in space are more likely to be similar than data from locations remote from one another” [2; 3; 4; 5]. This observational fact is called spatial autocorrelation [6; 7; 8; 9; 10; 11; 12; 13; 3; 4] and makes spatial data different from other types of data. More formally, spatial autocorrelation is the result of first- and second-order effects in spatial processes [2; 14; 3], where the former refers to environmental effects and the latter refers to interactions between samples. The same definition is true in time [4; 15; 16], referred to as temporal autocorrelation. Temporal data also might have an additional cyclic autocorrelation [17; 18; 19; 20; 21] termed cyclic temporal autocorrelation. Because of spatial and temporal autocorrelations, spatial-temporal data are not truly random. In other words, phenomena do not vary randomly through space and time.

Spatial autocorrelation and temporal autocorrelation are the backbone of spatial and temporal data analytics. For example, if the temperature at location A is 30°C, the temperature at location B, 1 m away from A, is also 30°C, the temperature at location C, 100 m away from A would be very close to 30°C, and the temperature at location D, 2 km away from A, is more uncertain and can be different (less or more) than 30°C. More examples can be found in the related literature [6; 7; 8; 9; 10; 11; 12; 13; 3; 4]. This temperature example can also be used to explain the temporal autocorrelation; more we temporally distance from the time that the temperature was measured at location A, more different the temperature can be from 30°C, and less certain we are about the temperature at location A. Temperature also has a strong cyclic temporal autocorrelation; temperature rises from winter to spring and keeps rising until summer, then starts falling from summer to autumn and keeps falling until winter and this cycle repeats itself. More examples can be found in the literature [17; 18; 19; 20; 21]. This cyclic behavior

makes temporal autocorrelation more complicated to model than spatial autocorrelation. Other examples of spatial-temporal data are elevation, air or water pollution, soil type, weather, population, landuse, and landslide.

Franklin [22], in her review paper, introduced the spatial dependence/autocorrelation as a source of information which has yet to be exploited in vegetation prediction models. O'Sullivan and Unwin [2] raised the concern with applying machine learning techniques to spatial data by briefly mentioning, in their book on geographic information analysis, that special characteristics of spatial data are ignored in regression and classification models applied by geographers. Shekhar et al. [23; 11; 13; 4] showed that spatial autocorrelation limits the usefulness of conventional classification and regression techniques for extracting spatial patterns. Santibanez et al. [6; 7] also raised this issue by stating that “machine learning algorithms are in general, not designed to deal with spatially autocorrelated data.” The assumption of independent and identically distributed random variables is not valid for spatial data because spatial autocorrelation causes the prediction residuals to exhibit clustering over geographic space [6; 7; 8; 9; 10; 11; 13; 4; 24; 25]. With respect to this issue, some researchers [8; 9; 10] suggested a spatial version of the linear least squares model which computes the weight vector as $(\boldsymbol{X}^T C \boldsymbol{X})^{-1} \boldsymbol{X}^T C y$ by defining C as some indicator of spatial neighborhood among responsive samples, $\boldsymbol{X}$ as the feature matrix, and y as the response vector. Other extensions of the linear least squares model, attempting to incorporate the spatial neighborhood, have also been proposed in the literature [26; 27; 28; 29] and have been able to improve the prediction accuracy. However, this issue is not the main focus of their work as they apply spatial neighborhood rather than spatial autocorrelation, do not consider the time, and do not go farther than linear least squares. On the other hand, spatial interpolation methods such as kriging or k-nearest neighbors (kNN) in

Euclidean space rather than feature space [2; 23; 13; 3; 4] ignore non-spatial features which makes them unreliable when spatial autocorrelation is weak or useless when the irresponsive sample is spatially far from responsive samples.

On the other hand, some researchers [17; 18; 19; 20] showed the reversibility (cyclic behavior) of landuse changes in time. Mertens and Lambin [17] showed that landuse predictions are more reliable in long term when more historic training samples are available. However, developing machine learning techniques that capture the cyclic behavior of temporal phenomena and adjust their predictions based on the irresponsive sample's time is not addressed in the literature.

1.3 PROPOSED APPROACH

Figure 1.1 shows a schematic spatial-temporal training dataset with four patterns and two features. Machine learning techniques learn to distinguish among patterns based on features. These patterns are recognized by measuring the similarity among features of training samples. Spatial-temporal data [30; 31] record the location and time of each observation (denoted by Lat_i, Lon_i, $Time_i$ in Figure 1.1) along with other features. Current machine learning techniques treat spatial-temporal problems no differently than other types of problems. Current machine learning techniques do not take into account spatial and/or temporal autocorrelations, neither in training nor in testing the predictor. That results in poor performance of machine learning techniques in the presence of spatial-temporal data [6; 7; 8; 9; 10; 11; 13; 4; 32; 15]. On the other hand, taking location and time as features in the training process is not the best way to incorporate the result of autocorrelation [23; 11; 13; 4] as it leaves autocorrelated prediction residuals behind [24; 25;

33], not to mention it will increase the sparseness of training samples in the feature space, as schematically shown in Figure 1.2. It also increases the number of free parameters in the predictor and consequently the demand for larger training datasets, referred to as curse of dimensionality [34; 35; 36].

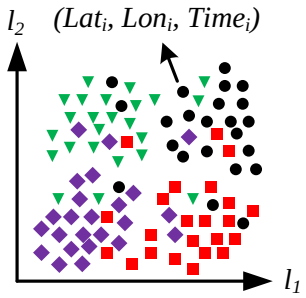


Figure 1.1. A 2D dataset with four patterns.

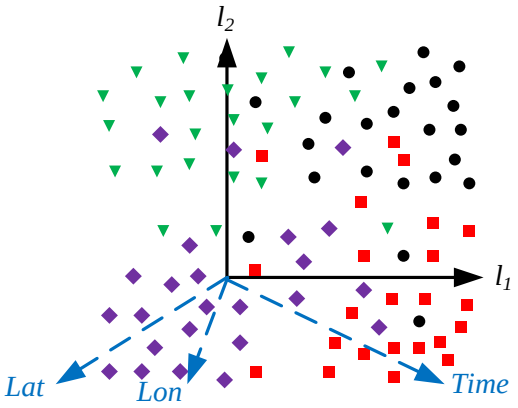


Figure 1.2. Considering space and time as features increases the sparseness of training samples in the feature space and the number of free parameters in the predictor.

We develop weighted machine learning techniques which are different from non-weighted ones in that they consider the weights of training samples and bias the predictor in favor of more important training samples, rather than being fair with regard to all training samples. Weighted versions of Bayesian classifier [37; 38], linear predictors (least squares [39], perceptron [40], and SVM [41; 42; 43]), and nonlinear predictors (decision tree [44], multilayer perceptron [45; 46; 47], and nonlinear SVM [48]) will be developed.

The spatial-temporal autocorrelation model shows how the autocorrelation (similarity between observations as a function of the space or time lag between them) among observations changes over space and time. The autocorrelation model considers general behaviors of spatial-

temporal phenomena: (a) as spatial or temporal distance between observations increases, their systematic similarity decreases, and (b) observations show periodic similarities over time. Instances of these behaviors are abundant in real life. For example, (a) temperature is more similar between two locations/times that are close to each other but we do not expect it to be similar between two locations/times that are too far from each other, and (b) temperature has a well-known yearly cycle. As another example, if it is raining in a city, (a) it is expected that it is also raining in nearby cities but we are uncertain about far cities and we are less certain about raining next week than next hour, and (b) raining shows periodic behaviors with different cycles with the yearly cycle being the most conspicuous. These behaviors can be extended to other spatial-temporal phenomena such as, elevation, air or water pollution, soil type, population, landuse, and landslide.

Figure 1.3 shows training samples in the space-time domain. Spatial and temporal autocorrelations are schematically shown in this figure, where samples close to each other in the space-time domain have similar responses and the responses also repeat themselves periodically over time. Due to the nature of spatial-temporal data, where spatial and temporal autocorrelations are directly related to spatial and temporal distances, not all training samples are equally important in predicting the output of a new sample. We develop a model that captures the characteristics of spatial-temporal autocorrelations and uses it to assign a spatial-temporal weight to each training sample based on its spatial-temporal distance to the irresponsive sample. Figure 1.4 schematically shows the spatial-temporal weights assigned to training samples with respect to the irresponsive sample in the space-time domain. In fact, training samples which are spatially and temporally uncorrelated with the irresponsive sample are less constructive to the training process. Training the predictor in the feature space must be biased in favor of training samples

with larger spatial-temporal weights, as shown in Figure 1.5. The weighted machine learning techniques take advantage of the spatial-temporal weight for training samples to bias the predictor in favor of more important training samples and exclude the location and time from input features to the machine.

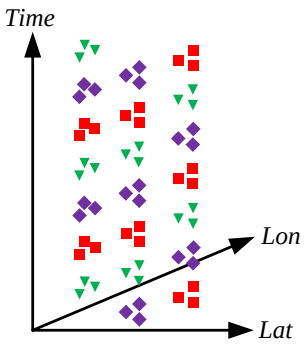


Figure 1.3. Spatial-temporal data in the space-time domain.

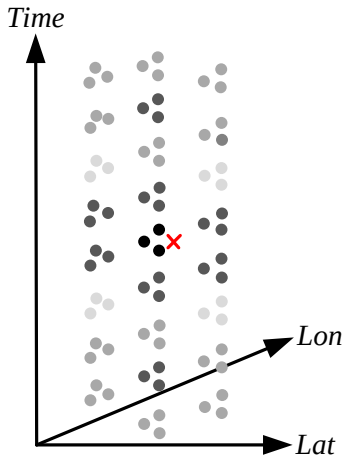


Figure 1.4. Training samples shaded based on their spatial-temporal weight with regard to the irresponsive sample (red cross) in the space-time domain.

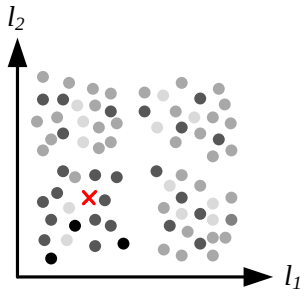


Figure 1.5. Predicting the output of a new sample (red cross) in the feature space where training samples are shaded based on their spatial-temporal weight.

The proposed approach prevents occurrence of the problems associated with considering location and time as features, potentially improves the prediction accuracy by biasing the predictor in favor of more important training samples, and expedites the training process by leaving out training samples with very low spatial-temporal weights. We need to train the predictor separately and independently for each irresponsive sample because the spatial-temporal weight for training samples depend on the location and time of the irresponsive sample.

It is also worth mentioning that our approach is different than active training/learning with support vectors [38; 49; 50]. Active learning trains the predictor using only a subset of training samples which are considered the most uncertain ones. The most uncertain training samples are those lying closest to the classifier's boundary decision. In our approach, we assign a spatial-temporal weight to each training sample, prune training samples with very low spatial-temporal weights, and bias the predictor in favor of more important training samples. The importance of training samples, in our approach, is determined based on the strength of their spatial-temporal autocorrelation with the irresponsive sample.

Our approach is also different than mixture of experts [51]. Mixture of experts is an ensemble method for combining different learners, where the feature space is divided between different learners/experts. The position of an irresponsive sample in the feature space determines which learner should be used for predicting a response. The mixture of experts has two main parts: individual learner/expert networks and the gating network. The gating network decides which learner should be used to predict the irresponsive sample's output and passes the irresponsive sample to the appropriate learner. Then the selected learner would predict the irresponsible sample's output. Training consists of optimizing the parameters of individual learners and the parameters of the gating network. Figure 1.6 shows a mixture of k experts, where θ_1 to θ_k are the gating network's parameters, x is an input feature vector, and y is its output. The gating network decides which learner is appropriate for this input sample and assigns a higher probability (θ) to that learner's output. Mixture of experts is a local machine learning technique, where an irresponsive sample's output is predicted trough a learner that has been trained using training samples close to that irresponsive sample in the feature space. In other words, training samples that are close to the irresponsive sample in the feature space will

contribute more in predicting the irresponsive sample's output than distant training samples. It is different than the proposed approach in this thesis because the contribution of training samples in predicting an irresponsive sample's output, in our work, is proportional to their spatial-temporal autocorrelation with the irresponsive sample in the space-time domain rather than the feature space. Besides, while in mixture of experts the contribution of training samples (the selection of the appropriate learner) is a function of Euclidean distance in the feature space, in our work the contribution of training samples is a function of autocorrelation in space-time domain.

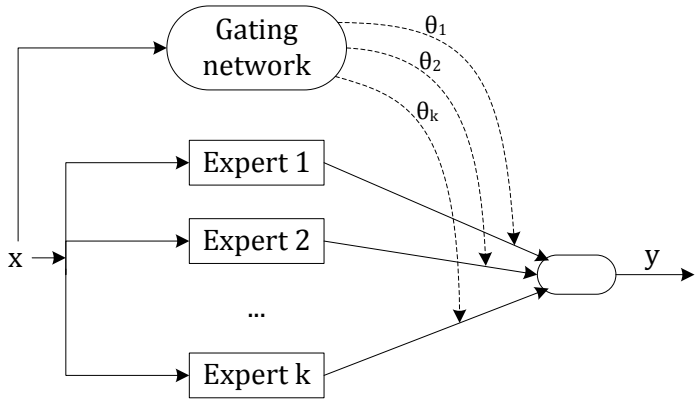


Figure 1.6. Mixture of experts.

At the end, the accuracy and time performance of the proposed approach will be compared against the following approaches:

a) ignoring location and time and using non-weighted machine learning techniques,

b) considering location and time as additional features in non-weighted machine learning techniques,

c) considering location and time as the only features in non-weighted machine learning techniques, and

d) estimating the irresponsive sample's response based on the weighted votes (i.e., spatial-temporal weights) of training samples' responses.

1.4 RESEARCH QUESTIONS AND CONTRIBUTIONS

In this thesis the following research questions are addressed:

- How do we bias different machine learning techniques in favor of training samples with larger weights?
- How do we determine the weight of a training sample for spatial-temporal data?
- Will the proposed weighted machine learning significantly improve the generalization accuracy in comparison with simply ignoring the location and time or considering them as features?

The contributions of this research are:

- Developing weighted machine learning techniques.
- Formulation of spatial-temporal autocorrelations of geographic phenomena and incorporating them as external knowledge in the training process.

1.5 ORGANIZATION OF THE DISSERTATION

This chapter showed the problem addressed in this thesis, highlighted its importance in both spatial-temporal statistics and machine learning, stated the research questions and contributions, and briefly introduced the proposed approach. Chapter 2 reviews select literature on machine learning techniques applied to spatial-temporal data. Chapter 3 discusses the developed weighted versions of select machine learning algorithms. Chapter 4 explains how the training samples' weights are calculated using spatial and temporal semivariograms. Chapter 5 includes experiments with real spatial-temporal datasets to compare the accuracy and time performance of the proposed approach with traditional ones. Chapter 6 concludes this work by providing insight into the proposed approach and future directions.

2.0 BACKGROUND

Li et al. [52] showed that combining machine learning techniques such as RF, RT, or SVM with spatial interpolation methods such as kriging or inverse distance squared (IDS) improves the accuracy in predicting seabed mud content in the southwest Australian margin. For these combined methods, first the machine learning technique is applied to the features, then the spatial interpolation method is applied to the residuals of the machine learning technique, and finally the interpolated residual values are added to the predicted values to produce the final predictions. The input features include bathymetry, distance-to-coast, seabed-slope, latitude, longitude, as well as their second and third powers, multiplication of latitude and longitude, multiplication of latitude to the second power of longitude, and multiplication of longitude to the second power of latitude. Their results showed that RF-OK (random forest combined with ordinary kriging), RF-IDS (random forest combined with inverse distance squared), random forest (RF), and RT-OK (regression tree combined with ordinary kriging) are the most accurate ones, respectively. Combination of SVM (with a linear or Gaussian kernel) with ordinary kriging (OK) or inverse distance squared (IDS) considerably improved its prediction accuracy, although it is still less accurate than OK or IDS. RF [53] was more accurate than regression tree (RT) and RT was more accurate than SVM.

Kanevski et al. [24] applied a similar approach in combining machine learning techniques with geostatistical models with the difference that the spatial coordinates of observations were

the only input features to machine learning techniques. They showed that nonlinear regression models including multilayer perceptron (MLP) and support vector regression (SVR) [54] with Gaussian kernel trained with spatial coordinates as input features could capture the nonlinear global spatial trend in the response variable. However, they leave the local spatial autocorrelation behind which manifests itself as spatially autocorrelated prediction residuals. Sequential Gaussian simulation was then applied to model these local prediction residuals. They reported a better generalization accuracy for the combined approach than either of machine learning or geostatistical models alone in predicting the radioactive soil contamination. In another effort to apply local regression models rather than global ones in predicting the radioactive soil contamination, Kanevski et al. [55] used spatial coordinates as the only input features to train a kNN and a general regression neural network (GRNN). GRNN is a non-parametric regression model based on Parzen windows [56]. They considered two versions of GRNN, one isotropic where the kernel bandwidth is the same in all directions and another anisotropic where the kernel has different bandwidths in different directions. To consider different bandwidths for different directions in a kernel, a matrix can be used as the bandwidth instead of a constant value. They considered two directions in their anisotropic GRNN model and found the optimal directions and bandwidths for those directions using leave-one-out cross-validation. KNN, isotropic and anisotropic GRNN resulted in an RMSE of 22.1, 12.4, and 11.9, respectively. This result indicates that (a) assigning different weights to neighbors based on Parzen windows, as in GRNN, improves the accuracy compared with equal weights for neighbors, as in kNN, and (b) considering different bandwidths in different directions for the kernel, as in anisotropic GRNN, improves the accuracy compared with a single bandwidth, as in isotropic GRNN.

Gilardi and Bengio [25; 33] compared the generalization accuracy of four regression techniques in estimating the rainfall based on spatial coordinates of observations. Their results, in line with Kanevski et al. [24] results, confirmed that global regression models, including MLP and SVR [54] with Gaussian kernel, trained with spatial coordinates as input features capture the nonlinear global spatial trend in the response variable but leave the local spatial autocorrelation behind. On the other hand, local regression techniques, including mixture of experts (ME) [57] and local SVR (which is the standard SVR trained only by responsive samples near the irresponsive sample in the feature space), achieved slightly better generalization accuracies because they were able to partly capture the local spatial autocorrelation. They reported an RMSE of 63.4, 59, 57.1, and 53.2 for SVR, MLP, local SVR, and ME, respectively.

Santibanez et al. [6] compared the accuracy of different machine learning techniques in regressing median rent price per zip code of a two bedroom two bathroom apartment in the Miami-Fort Lauderdale-West Palm Beach metropolitan area in Florida, USA, based on 23 demographic features. Location and time were not among the features. The best accuracy was achieved by MLP combined with PCA, followed by SVM with Gaussian kernel, RF, cubist, partial least squares, MLP, gradient boosting machine, SVM with linear kernel, and general least squares. Santibanez et al. [7] compared the accuracy of the same machine learning techniques with the same input features but with simulated data of varying degrees of spatial autocorrelation. SVM with Gaussian kernel resulted in the highest accuracy for weaker spatial autocorrelations, MLP with PCA achieved the same accuracy as SVM with Gaussian kernel as spatial autocorrelation was increased, and finally cubist performed best when the spatial autocorrelation was very strong.

Cracknell and Reading [58] applied five machine learning techniques, Naïve Bayes (NB), kNN, RF, SVM (using the one-against-one scheme), and MLP, in classifying lithology based on airborne geophysics (containing a digital elevation model, total magnetic intensity, and four gamma-ray spectrometry channels comprising Potassium, Thorium, Uranium, and total count channels) and Landsat ETM+ images. RF achieved the highest accuracy followed by SVM, kNN, MLP, and NB where kNN ran fastest and SVM slowest. They considered different scenarios for spatial distribution of training samples with/without considering location as an input feature. They observed that regardless of including/excluding location as an input feature, substantial higher accuracies are achieved by all machine learning techniques as training samples become more spatially dispersed across the geographic region. This is not surprising as spatial autocorrelation among responses of training samples limits proper training when training samples are not well scattered in the geographic region. Another observation was that higher generalization accuracies are achieved when location is considered as the only feature compared to the other two scenarios that either exclude location or consider it as an additional feature. Although, it is plausible that considering location as an additional feature would improve the generalization accuracy, the better accuracy achieved with using location as the only feature than using it in combination with other features is surprising. This can be true if the spatial distribution of training samples is dense and well engineered and even in that case the trained machine will not perform as well if an irresponsive sample is beyond the autocorrelation range of all training samples.

Table 2.1 provides a list of select scientific articles that do not consider location and time, as features, when applying machine learning techniques for predicting spatial-temporal responses.

Table 2.1. Machine learning methods applied to spatial-temporal data in the literature ignoring location and time.

	Response	Features	Machine learning method	Accuracy
[17]	Classification with two classes: Propensity of forests in southern Cameroon for deforestation	• Distance to the nearest road weighted by the average transportation cost • Distance to the nearest market town weighted by the average transportation cost and the price of the agricultural products at the market town • Soil aptitude for agriculture • Shortest distance to the nearest forest/nonforest edge • Spatial fragmentation of the forest cover in the immediate surroundings of each location	Logistic regression (different models are developed for different time scales)	Precision = 89%
[59]	Classification with 9 classes: Landuse	• Bands 2, 4, 5, and 7 of Landsat TM data • Geology • Hydrology (flow accumulation) • Surface morphology (slope, aspect)	MLP	overall accuracy = 72.61%
[60]	Classification with 6 classes: Landuse	• All four bands of SPOT 6 image (blue, green, red, and near-infrared) • A cluster number assigned to each pixel based on Fuzzy k-means clustering algorithm from all four bands of the image • NDVI calculated for each pixel from the red and near-infrared bands	SVM with Gaussian kernel (using the one-against-all scheme). A 3 × 3 pixel majority filter was applied to all classifications to eliminate the salt and pepper noise.	Overall accuracy = 98%
[61]	Classification with 13 classes: Landuse	Global 8 km resolution AVHRR Pathfinder Land data for 1984 with 24 metrics including, the maximum annual, minimum annual, mean annual, and, amplitude (maximum minus minimum) for • the normalized difference vegetation index (NDVI), • Channel 1 (visible reflectance, 0.58–0.69 μm), • Channel 2 (near-infrared reflectance, 0.725–1.1 μm), • Channel 3 (thermal infrared, 3.55–3.93 μm), • Channel 4 (thermal, 10.3–11.3 μm), and • Channel 5 (thermal, 11.5–12.5 μm)	Decision tree	Overall accuracy = 85%
			Decision tree with bagging	Overall accuracy = 87%
			Decision tree with boosting	Overall accuracy = 89.5%
	Classification with 6 classes: Landuse	Landsat Thematic Mapper scene around Pucallpa, Peru acquired 16 October 1996 including • five bands at 30 m resolution (.45–.53 μm, .52–.60 μm, .63–.69 μm, .76–.90 μm, and 1.55–1.75 μm)	Decision tree	Overall accuracy = 84.5%
			Decision tree with bagging	Overall accuracy = 87%
			Decision tree with boosting	Overall accuracy = 89.5%
[62]	Classification with 5 classes: Grassland type	• SAR data (a total of 12 ENVISAT ASAR images operated at C-band, 15 ERS-2 images operated at C-band, and 12 ALOS PALSAR images operated at L-band), • Ancillary data (soils, sub-soils, elevation, and slope)	SVM with Gaussian kernel (using the one-against-one scheme)	Overall accuracy = 92.5- 97.9%
			RF	Overall accuracy = 92.4-98%
			Extremely randomized trees	Overall accuracy = 94.1- 98.7%

[63]	Classification with 2 classes: Miscanthus presence/absence at the level of the farmer's block	• Agronomical variables (topsoil water capacity, soil texture, and distance to rivers) • Morphological variables (size, shape, and maximum values of elevation and slope for each farmer's block) • Contextual variables (the farmer's block distance to the overall farmland, to the transformation plant, and to the road, proximity to the built-up areas, and length of the parcel boundaries shared with the neighboring woods)	Boosted regression tree	AUC (for training data) = 0.793
[21]	Classification with 2 classes: Presence or absence of harmful algal blooms in the Gulf of Mexico	• The level-2 SeaWiFS sensor data (bands at 443, 490, 510, and 555 nm, and Chlorophyll-a) for the period 1999–2004 with a spatial resolution of 1.1 Km. • The level-2 MODIS-A sensor data (bands at 412, 443, 488, 531, and 551 nm and Chlorophyll-a) for the period 2002–2004 with a spatial resolution of 1 Km. • Ancillary data (meteorological and ozone data) • The feature vector (spectral data) at each sample is the average of features in a cubical window of size 3° of latitudes, 3° of longitudes, and 3 days centered at that sample. • Kernel PCA was used to transform features and only the first 300 components for SeaWiFS features and 72 components for MODIS-A features are used.	SVM with HTRBF kernel (the classifier's parameters are optimized trough cross validation with Genetic Algorithm applied to narrow down the search space)	Kappa coefficient = 0.75
[64]	Classification with 2 classes: Landslide prone/non-prone areas	• Convergence index (characterizes soil and debris erosion and deposition within the landscape) • Compound terrain index (the logarithm of the ratio between upslope contributing area and slope gradient) • Distance from channel base level (the limiting level below which a stream cannot erode its channel) • Distance from faults • Distance from thrust • Downslope distance gradient (how far a given amount of water must travel in the landscape to lose a certain amount of potential energy) • Elevation • Insolation (the amount of radiation reflected by the terrain) • Internal relief (maximum elevation change per unit area) • Morphological protection index (the positive openness which expresses the degree of dominance or enclosure of a location within the landscape) • Slope • Stream power index (expresses the erosive potential of overland flow) • The presence of clay and marl-rich sedimentary formations • Attitude of rock strata • Landuse	ν-SVM with radial basis function Gaussian kernel (the classifier's parameters are optimized trough cross validation)	AUC = 0.83
			Logistic regression	AUC = 0.79
			Linear discriminant analysis	AUC = 0.79
			Naïve Bayes	AUC = 0.76
[65]	Regression: Intra-and-inter species forest aboveground biomass level	• All five bands of 5m RapidEye images (blue, green, red, near infrared, and red-edge) • NDVI and 13 other vegetation indices calculated for each pixel from different bands of RapidEye images	Stochastic gradient boosting regression tree	R^2 = 0.61
			RF	R^2 = 0.37

3.0 WEIGHTED MACHINE LEARNING

In this chapter, machine learning algorithms are modified to take the training samples' weights into account. The weighted machine learning techniques developed in this chapter, not only provide users with the opportunity to give different weights to training samples, but also can be embedded into other algorithms such as AdaBoost [66; 67], where there is a hierarchy of classifiers, each requiring to be trained using a different weighting over training samples. Figure 3.1 shows, schematically, how non-weighted linear predictors become biased when the training samples' weights are taken into account. Figure 3.2 shows the same for nonlinear predictors. There are two classes, one indicated with circles and the other with squares. Darkness of training samples shows their weights and the classifier is represented with a dashed line. The weighted classifier decides in favor of more important samples by keeping more distance from them.

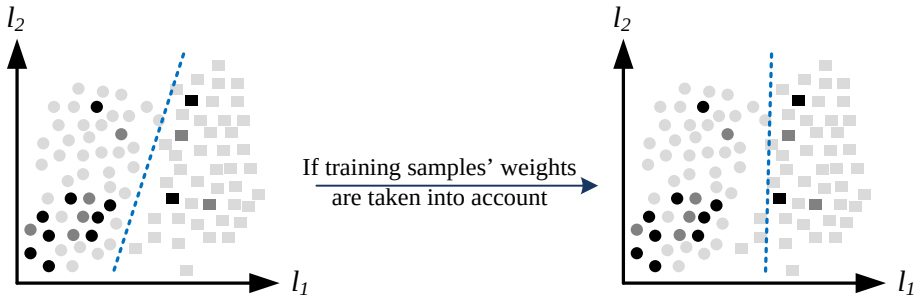


Figure 3.1. A non-weighted linear classifier (left) vs. a weighted linear classifier (right).

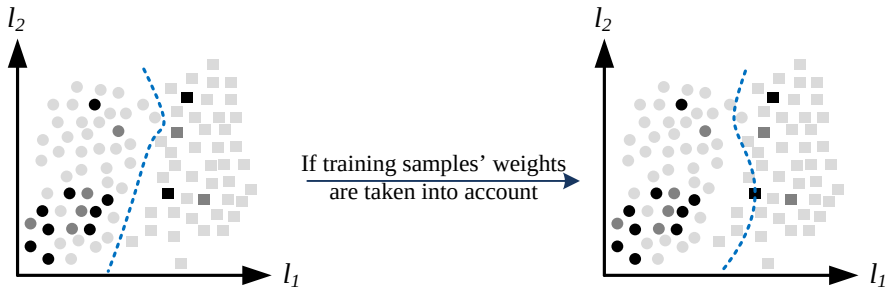


Figure 3.2. A non-weighted nonlinear classifier (left) vs. a weighted nonlinear classifier (right).

To bias the predictor in favor of more important training samples, we embed the training samples' weights into the cost function. This way we regulate the misclassification cost based on the weights during training. In other words, misclassifying more important training samples would be more costly and the predictor will attempt to avoid it. This approach is possible only for machine learning algorithms which are based on minimizing a cost function. For Bayesian predictors, we embed the training samples' weights into the probability distribution functions. This way we increase the likelihood of a class when the irresponsive sample (the sample with an unknown output) is close to training samples with large weights in that class. In short, training the weighted predictor is more concerned about correct prediction of training samples with larger weights than those with smaller weights. As a result, the trained model predicts in favor of training samples with larger weights. This makes the weighted predictor different than its non-weighted counterpart.

In this chapter, we use the training dataset in Table 3.1 to show the difference between the weighted machine learning techniques developed here and their non-weighted counterparts. We consider two classes ω_1 and ω_2, each with 10 samples, and two features l_1 and l_2 to simplify the visualization. Training samples and their weights in this table are chosen carefully to emphasize the difference between weighted and non-weighted predictors. The training dataset is

shown in Figure 3.3. Circles represent class ω_1 and squares represent class ω_2. Darkness of training samples shows their weight.

Table 3.1. Training samples and their weights.

l_1	l_2	Class	Spatial-temporal weight
1	2	ω_1	1
1	3	ω_1	1
2	1	ω_1	1
2	2	ω_1	1
2	3	ω_1	1
2	4	ω_1	1
3	2	ω_1	1
3	3	ω_1	1
4	1	ω_1	2
4	4	ω_1	4
3	1	ω_2	1
3	4	ω_2	1
4	2	ω_2	1
4	3	ω_2	1
5	1	ω_2	1
5	2	ω_2	1
5	3	ω_2	1
5	4	ω_2	1
6	2	ω_2	1
6	3	ω_2	1

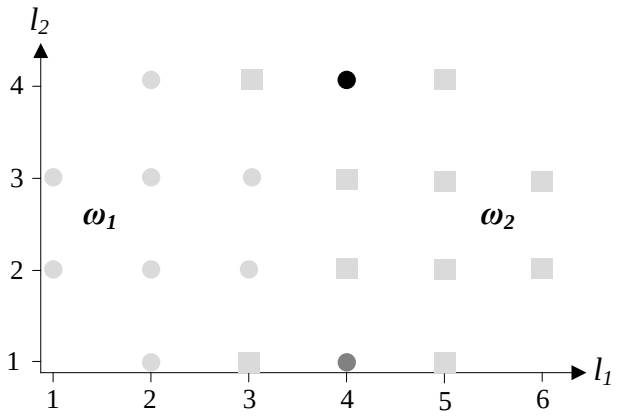


Figure 3.3. Training samples from two classes, circles and squares, shaded based on their weights.

3.1 BAYESIAN PREDICTOR

3.1.1 Classification

The Bayes classifier calculates the probability of different classes given the observed feature vector as $p(\omega_j|x)=p(\omega_j)p(x|\omega_j)/p(x)$ and then assigns x to the class with the highest probability [37; 38]; where $p(\omega_j|x)$ is the posterior probability, $p(\omega_j)$ is the prior probability, and $p(x|\omega_j)$ is the likelihood. The denominator, $p(x)$, is usually ignored in calculations as it is the same for all classes. A simple way to embed weights (g_i) for training samples into the Bayes classifier is to define the prior probability ($p(\omega_j)$) as the sum of weights of training samples belonging to class ω_j divided by the sum of all weights (Equation (3.1)).

$$p(\omega_j) = \sum_{\forall i | x_i \in \omega_j} g_i \Big/ \sum_{\forall i} g_i \qquad (3.1)$$

Regardless of parametric or non-parametric definition of the likelihood ($p(x|\omega_j)$), an important drawback with this simple approach is that it does not consider where the irresponsive sample (x) is situated with respect to more important training samples in each class. For example, the irresponsive sample in Figure 3.4, shown with a cross, is closer to more important samples (darker ones in the figure) in ω_2 and one expects it to be classified in ω_2. However, based on the aforementioned approach, it will be classified in ω_1 because ω_1 has a larger prior ($p(\omega_1)>p(\omega_2)$) and the likelihoods for two classes are equal ($p(x|\omega_1)=p(x|\omega_2)$). Likelihoods, $p(x|\omega_1)$ and $p(x|\omega_2)$, are calculated without considering the weights. To solve this problem, weights need to be considered in likelihoods.

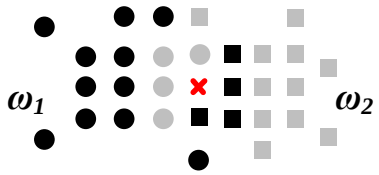


Figure 3.4. Two classes shown with circles and squares where the darkness of samples shows their weight with respect to the irresponsive sample, shown with a cross.

To take into account the position of x with respect to more important training samples in each class, we define the likelihood ($p(x|\omega_j)$) based on non-parametric Parzen windows [56], shown in Equation (3.2), instead of calculating the priors from Equation (3.1).

$$p(x|\omega_j) = \frac{1}{N_j} \sum_{\forall i | x_i \in \omega_j} g_i K(x - x_i, \Sigma_j) \qquad (3.2)$$

In this equation, N_j is the size of the class ω_j, x_i represents the feature vector of the i-th training sample, x represents the irresponsive sample's feature vector, g_i is the i-th training sample' weight, K is the kernel function, and Σ_j is the covariance matrix for class ω_j. The step kernel in Equation (3.3) or the Gaussian kernel in Equation (3.4) can be used in Equation (3.2), where l is the dimension of feature space and the subscript k in x_k and x_{i_k} refers to the k-th feature in the corresponding feature vector. More kernels are available in Hardle [68] and Fan and Gijbels [69]. Instead of choosing the kernel bandwidth to be a constant value, which is the common practice, we choose the covariance matrix for class ω_j (shown by Σ_j) divided by a constant value (shown by σ) as the kernel bandwidth for class j. The constant value (σ) can be tuned using cross-validation.

$$K(x - x_i, \Sigma_j) = \begin{cases} \frac{1}{|\Sigma_j/\sigma|^{1/2}} & |x_k - x_{i_k}| < \frac{1}{2}|\Sigma_j/\sigma|^{\frac{1}{2l}} \\ 0 & otherwise \end{cases} \qquad (3.3)$$

$$K(x - x_i, \Sigma_j) = \frac{1}{(2\pi)^{l/2}\left|\Sigma_j/\sigma\right|^{1/2}} exp\ (-\frac{1}{2}(x - x_i)^T(\Sigma_j/\sigma)^{-1}(x - x_i)) \qquad (3.4)$$

Applying Equation (3.2) to calculate the likelihoods in Figure 3.4 results in $p(x|\omega_1)<p(x|\omega_2)$ and consequently $p(\omega_1|x)<p(\omega_2|x)$ which classifies the irresponsive sample in ω_2.

3.1.2 Regression

In case of regression, Equation (3.5) can be used to estimate the response at the irresponsive sample *x*. This equation estimates the response at *x* as the weighted average of other training samples' responses, where each training sample's weight is the multiplication of its original weight (g_i) by the output of the kernel for that training sample ($K(x\text{-}x_i,\Sigma)$). In other words, a training sample's weight in this equation ($g_iK(x\text{-}x_i,\Sigma)$) is the combination of its importance as well as its distance to the irresponsive sample in the feature space. The latter is what the kernel is concerned about.

$$y(x) = \frac{\sum_{i=1}^{N} y_i g_i K(x - x_i, \Sigma)}{\sum_{i=1}^{N} g_i K(x - x_i, \Sigma)} \qquad (3.5)$$

Since there are no classes in regression, the covariance matrix (Σ) in Equation (3.5) is defined as the covariance matrix over all training samples.

3.1.3 Experiment

Here we use the dataset in Table 3.1 to show the effect of embedding training samples' weights in likelihoods (Equation (3.2)) on the irresponsive sample's classification. Priors are considered equal since the frequencies of the two classes are the same. The Gaussian kernel in Equation (3.4) with a bandwidth of $\Sigma_j/3$ is used as Parzen window, where Σ_j shows the covariance matrix

of class ω_j. Figure 3.5 shows the division of the feature space between the two classes with and without considering the training samples' weights in calculating the likelihoods. It is shown that when the weighted Bayesian classifier is applied, the classification of the irresponsive sample (shown with a cross) is switched from class ω_2 to class ω_1 because of its proximity to some important samples in class ω_1.

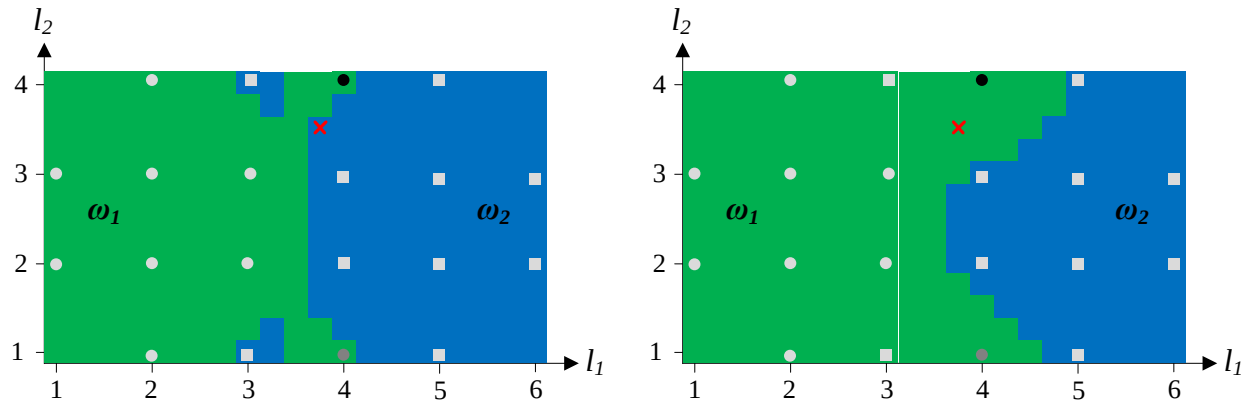


Figure 3.5. Division of the feature space between the two classes, circles and squares, without (left) and with (right) considering the training samples' weights (darkness of samples) in Bayesian classifier.

3.2 LINEAR PREDICTORS

3.2.1 Least squares (LS)

The output of the LS predictor is $\boldsymbol{x}^T\boldsymbol{w}$ where $\boldsymbol{w}$ is the extended weight vector to include the threshold or intercept (w_0) and $\boldsymbol{x}$ is the extended feature vector to include a 1. The desired output is denoted with y_i. The weight vector will be computed so as to minimize the sum of square errors between the desired and true outputs [39], that is:

$$J(w) = \sum_{i=1}^{N} (y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2 \qquad (3.6)$$

where N is the number of training samples. Minimizing the cost function in Equation (3.6) with respect to $\boldsymbol{w}$ results in:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0 \rightarrow \sum_{i=1}^{N} \boldsymbol{x}_i (y_i - \boldsymbol{x}_i^T \boldsymbol{w}) = 0 \rightarrow \left(\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T \right) w = \sum_{i=1}^{N} \boldsymbol{x}_i y_i \qquad (3.7)$$

Let us define:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1l} & 1 \\ x_{21} & x_{22} & \dots & x_{2l} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nl} & 1 \end{bmatrix} \; and \; y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad (3.8)$$

where $\boldsymbol{X}$ is an $N \times (l+1)$ matrix whose rows are the feature vectors with an additional 1, l is the number of features, and y is a vector consisting of the corresponding desired responses. Then:

$$\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T = \boldsymbol{X}^T \boldsymbol{X} \;\; and \;\; \sum_{i=1}^{N} \boldsymbol{x}_i y_i = \boldsymbol{X}^T y \qquad (3.9)$$

By substituting Equation (3.9) in Equation (3.7) we have:

$$(\boldsymbol{X}^T \boldsymbol{X}) \boldsymbol{w} = \boldsymbol{X}^T y \rightarrow \boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T y \qquad (3.10)$$

Matrix $\boldsymbol{X}^+ = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T$ is known as the pseudoinverse of $\boldsymbol{X}$ and is equal to $\boldsymbol{X}^{-1}$ if $\boldsymbol{X}$ is square. To develop the weighted version of LS predictor, we adjust the cost of error based on the weight of training samples (g_i),

$$J(w) = \sum_{i=1}^{N} g_i (y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2 \qquad (3.11)$$

Minimizing the cost function in Equation (3.11) with respect to $\boldsymbol{w}$ results in:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0 \rightarrow \sum_{i=1}^{N} g_i \boldsymbol{x}_i (y_i - \boldsymbol{x}_i^T \boldsymbol{w}) = 0 \rightarrow \left(\sum_{i=1}^{N} g_i \boldsymbol{x}_i \boldsymbol{x}_i^T \right) w = \sum_{i=1}^{N} g_i \boldsymbol{x}_i y_i \tag{3.12}$$

Let us define:

$$G = \begin{bmatrix} g_1 & 0 & 0 & 0 \\ 0 & g_2 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g_N \end{bmatrix} \tag{3.13}$$

Then:

$$\sum_{i=1}^{n} g_i \boldsymbol{x}_i \boldsymbol{x}_i^T = \boldsymbol{X}^T G \boldsymbol{X} \quad and \quad \sum_{i=1}^{n} g_i \boldsymbol{x}_i y_i = \boldsymbol{X}^T G y \tag{3.14}$$

Substituting Equation (3.14) in Equation (3.12) results in:

$$(\boldsymbol{X}^T G \boldsymbol{X}) \boldsymbol{w} = \boldsymbol{X}^T G y \rightarrow \boldsymbol{w} = (\boldsymbol{X}^T G \boldsymbol{X})^{-1} \boldsymbol{X}^T G y \tag{3.15}$$

Equation (3.15) is known as weighted least squares [70]. Let us investigate what happens if the weight of all training samples is equal to a constant c. In this case, $G=c\times I_{N\times N}$ where $I_{N\times N}$ is the $N\times N$ identity matrix. Substituting this in Equation (3.15) results in:

$$\boldsymbol{w} = (\boldsymbol{X}^T c I \boldsymbol{X})^{-1} \boldsymbol{X}^T c I y = (c \boldsymbol{X}^T \boldsymbol{X})^{-1} c \boldsymbol{X}^T y = \frac{1}{c} (\boldsymbol{X}^T \boldsymbol{X})^{-1} c \boldsymbol{X}^T y = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T y \tag{3.16}$$

In other words, the weighted LS is no different than the non-weighted LS if all weights are equal. This is the case with all weighted predictors developed in this work.

3.2.1.1 Experiment Here we use the dataset in Table 3.1 to show the effect of embedding the training samples' weights in LS (Equation (3.15)). Figure 3.6 shows the division of the feature space between the two classes with and without considering the training samples' weights in computing the linear classifier. In the weighted LS classifier, training samples with large weights from ω_1 push the border toward class ω_2.

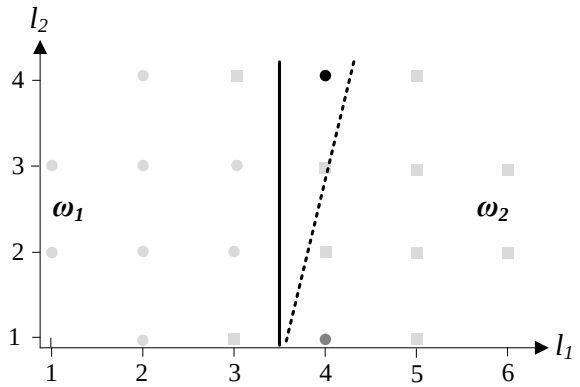


Figure 3.6. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in LS classifier.

3.2.2 Perceptron

The perceptron cost function is defined as [40]:

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} y_i \boldsymbol{w}^T \boldsymbol{x}_i \qquad , y_i = \begin{cases} +1 & if\ w\boldsymbol{x}_i > 0\ but\ \boldsymbol{x}_i \in \omega_2 \\ -1 & if\ w\boldsymbol{x}_i < 0\ but\ \boldsymbol{x}_i \in \omega_1 \\ 0 & if\ w\boldsymbol{x}_i > 0\ and\ \boldsymbol{x}_i \in \omega_1 \\ 0 & if\ w\boldsymbol{x}_i < 0\ and\ \boldsymbol{x}_i \in \omega_2 \end{cases} \tag{3.17}$$

where N is the number of training samples, $\boldsymbol{x}_i$ is the i-th feature vector including an additional 1 as its last element, and $\boldsymbol{w}$ is the weight vector (the perpendicular vector to the hyperplane classifier toward class ω_1) including the threshold (w_0) as its last element. The cost function is minimized if the classifier produces a positive response for samples of class ω_1 and a

negative response for samples of class ω_2. We can iteratively find the weight vector that minimizes the perceptron cost function using the gradient descent scheme [40; 71]:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta\boldsymbol{w}_t = \boldsymbol{w}_t - \alpha \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}|_{\boldsymbol{w}=\boldsymbol{w}_t} = \boldsymbol{w}_t - \alpha \sum_{i=1}^{N} y_i \boldsymbol{x}_i \tag{3.18}$$

where $\boldsymbol{w}_t$ is the weight vector estimate at the t-th iteration and α is the training rate which is a small positive number. Equation (3.18) is called batch mode training [72] where all training samples participate in calculating the gradient at each iteration. We can also apply the pattern or online mode [72] where the gradient at each iteration is calculated based on only one training sample; or the stochastic mode [73] where the gradient at each iteration is calculated based on a small random subset of training samples.

We embed the training samples' weights (g_i) in the perceptron cost function (Equation (3.19)) to punish the classifier more for misclassifying training samples with larger weights and less for training samples with smaller weights. In other words, the training samples' weights enter the cost function to adjust the perceptron cost based on the importance of training samples. The perceptron classifier is no longer equally fair to all training samples.

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} g_i y_i \boldsymbol{w}^T \boldsymbol{x}_i \tag{3.19}$$

With the new cost function, the iterative steps for updating the weight vector through the gradient descent scheme will change to:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta\boldsymbol{w}_t = \boldsymbol{w}_t - \alpha \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}|_{\boldsymbol{w}=\boldsymbol{w}_t} = \boldsymbol{w}_t - \alpha \sum_{i=1}^{N} g_i y_i \boldsymbol{x}_i \tag{3.20}$$

If we define $\alpha_i^* = \alpha g_i$ we obtain:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \tag{3.21}$$

Therefore, the weighted perceptron classifier can be obtained by including the weights in the cost and defining the training rate as $\alpha_i^*=\alpha g_i$ which means a different training rate for each training sample based on its weight. Adjusting the training rate based on the training samples' weights and including the weights in the cost function bias the trained perceptron in favor of training samples with larger weights.

3.2.2.1 Experiment Here we use the dataset in Table 3.1 to show the effect of including training samples' weights in perceptron classifier. Figure 3.7 shows the division of the feature space between the two classes with and without considering the training samples' weights in computing the linear classifier. The high cost of misclassifying important samples from class ω_1 in weighted perceptron classifier pushes the border toward class ω_2.

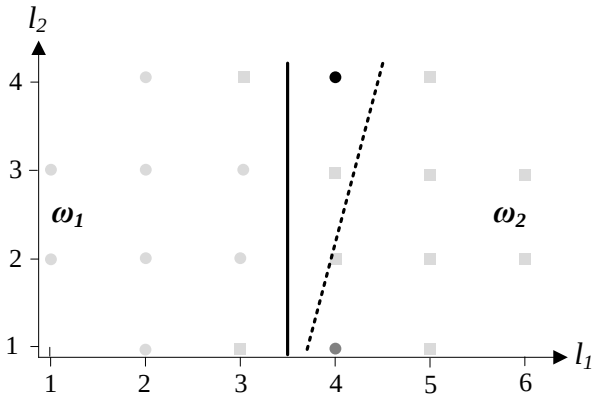


Figure 3.7. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in perceptron classifier (logistic activation function and adaptive training rate with 1000 iterations).

3.2.3 SVM

3.2.3.1 Two linearly separable classes Assume ω_1 and ω_2 are two linearly separable classes shown in Figure 3.8. SVM [41; 42; 43] maximizes the margin around the hyperplane separating the two classes by maximizing the distance to the closest point from either class. We know that the distance between a sample x_i and a hyperplane $f(x)=w^Tx+w_0=0$ is obtained from $|f(x_i)|/||w||$. Assume x_1 is the nearest sample in class ω_1 to the hyperplane $f(x)$ and x_2 is the nearest sample in class ω_2 to the hyperplane $f(x)$. Then x_1 and x_2 are called support vectors. To maximize the margin, the hyperplane $f(x)$ must intersect the line connecting x_1 and x_2 at its midpoint, as shown in Figure 3.8. Therefore, we can scale w and w_0 so that $f(x_1)=1$ and $f(x_2)=-1$. This leads to having a margin of:

$$\frac{|f(x_1)|}{||w||}+\frac{|f(x_2)|}{||w||}=\frac{1}{||w||}+\frac{1}{||w||}=\frac{2}{||w||} \tag{3.22}$$

Since x_1 and x_2 are the closest samples to the hyperplane $f(x)$, the distance of other samples from the hyperplane is greater than $1/||w||$, as shown in Figure 3.8. Therefore, we have:

$$\begin{cases} f(x_i) \geq 1 & \forall x_i \in \omega_1 \\ f(x_i) \leq -1 & \forall x_i \in \omega_2 \end{cases} \tag{3.23}$$

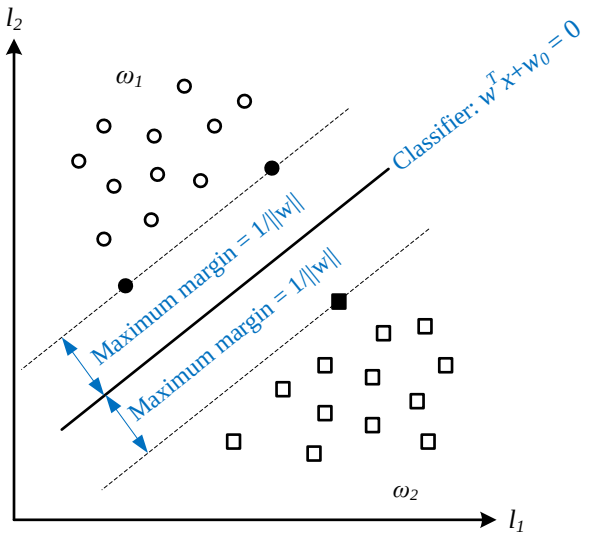


Figure 3.8. SVM classifier for two linearly separable classes; black points show support vectors.

We define:

$$y_i = \begin{cases} +1 & \forall x_i \in \omega_1 \\ -1 & \forall x_i \in \omega_2 \end{cases} \tag{3.24}$$

Substituting Equation (3.24) in Equation (3.23) results in:

$$y_i f(x_i) \geq 1, \quad \forall x_i \tag{3.25}$$

We need to maximize the margin ($2/\|w\|$) in Equation (3.22) which is equivalent to minimizing the norm $\|w\|$. The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$\begin{cases} minimize\ J(w, w_0) = \dfrac{1}{2}\|w\|^2 = \dfrac{1}{2} w^T w & (3.26) \\ subject\ to\ y_i(w^T x_i + w_0) \geq 1 \quad , \quad i = 1,2, \ldots, N & (3.27) \end{cases}$$

where N is the number of training samples. The above cost function is convex and the constraints are linear and define a convex set of feasible solutions. The corresponding Lagrangian function $\mathcal{L}(w,w_0, \lambda)$ for the above convex programming problem is defined as follows [74; 75; 76; 77]:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \lambda_i [y_i (w^T x_i + w_0) - 1] \quad (3.28)$$

where λ_i, i=1,2,…,N are the Lagrangian multipliers associated with the constraint in Equation (3.27). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda \geq 0}\, min_{w,w_0}\, \mathcal{L}(w, w_0, \lambda)$ [74; 75; 76; 77]. The Karush-Kuhn-Tucker conditions that $min_{w,w_0}\, \mathcal{L}(w, w_0, \lambda)$ has to satisfy are [74; 75; 76; 77]:

$$\begin{cases} \dfrac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{N} \lambda_i y_i x_i & (3.29) \\ \dfrac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w_0} = 0 \rightarrow \sum_{i=1}^{N} \lambda_i y_i = 0 & (3.30) \\ \lambda_i \geq 0 \ , \quad i = 1,2, \dots, N & (3.31) \\ \lambda_i [y_i (w^T x_i + w_0) - 1] = 0 \ , i = 1,2, \dots, N \ \ (complementary\ slackness\ conditions) & (3.32) \end{cases}$$

Equations (3.29) and (3.30) depend only on training samples whose $\lambda_i \neq 0$, referred to as support vectors. On the other hand, the conditions in Equation (3.32) state that either λ_i or $y_i(w^T x + w_0)$-1 must be zero. Therefore support vectors are training samples where $|w^T x + w_0| = 1$ ($y_i(w^T x + w_0)$-1=0 and $\lambda_i \neq 0$) which means they are on the boundary of the margin. Therefore, Equations (3.29) and (3.30) depend only on support vectors and consequently the hyperplane classifier is designed only based on support vectors and is independent of other training samples because their λ_i is zero. While, none of the training samples falls inside the margin (by construction), this is not necessarily the case for irresponsive samples. The intuition is that maximizing the margin on the training samples will lead to good separation on the irresponsive samples.

By expanding Equation (3.28), we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} w^T w - w^T \sum_{i=1}^{N} \lambda_i y_i x_i - w_0 \sum_{i=1}^{N} \lambda_i y_i + \sum_{i=1}^{N} \lambda_i$$

By replacing $\sum_{i=1}^{N} \lambda_i y_i = 0$ from Equation (3.30) in the above equation, we get:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} w^T w - w^T \sum_{i=1}^{N} \lambda_i y_i x_i + \sum_{i=1}^{N} \lambda_i$$

By substituting *w* from Equation (3.29) , we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} \left[\sum_{i=1}^{N} \lambda_i y_i x_i \right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i \right] - \left[\sum_{i=1}^{N} \lambda_i y_i x_i \right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i \right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = -\frac{1}{2} \left[\sum_{i=1}^{N} \lambda_i y_i x_i \right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i \right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

Now we maximize the above Lagrangian function with respect to λ:

$$\begin{cases} \max\limits_{\lambda} \left(\sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j \right) & (3.33) \\ subject\ to \sum\limits_{i=1}^{N} \lambda_i y_i = 0 & (3.34) \\ \lambda_i \geq 0 \qquad , i = 1,2, \dots, N & (3.35) \end{cases}$$

Once the optimal Lagrange multipliers (λ_i) have been computed by maximizing Equation (3.33), *w* is obtained by replacing them in Equation (3.29) and w_0 is computed as an average value obtained using complementary slackness conditions in Equation (3.32) for support vectors ($\lambda_i \neq 0$).

The weighted version of SVM needs to be more sensitive to training samples with larger weights. In other words, the distance from training samples to the classifier hyperplane needs to be compromised based on their weights. From a geometric point of view, we develop the weighted SVM by moving training samples toward the classifier hyperplane by a factor

proportional to their weight (g_i), which will change the selection of support vectors and eventually the design of the classifier. We measure the distance of a training sample (x_i) from the classifier hyperplane ($f(x)$) through Equation (3.36), where the actual distance is reduced by a factor of $1/(1+g_i)$. If a training sample's weight is zero, its distance to the classifier hyperplane, in Equation (3.36), remains intact, and if its weight is very large, its distance will become close to zero.

$$\frac{|f(x_i)|}{||w|| \times (1+g_i)} = \frac{|w^T x_i + w_0|}{||w|| \times (1+g_i)} \tag{3.36}$$

Assume x_1 is the nearest sample in class ω_1 to the classifier hyperplane based on the distance calculated from Equation (3.36) and x_2 is the nearest sample in class ω_2 to the classifier hyperplane. We can scale w and w_0 so that $f(x_1)/(1+g_1)=1$ and $f(x_2)/(1+g_2)=-1$. This leads to having a margin of:

$$\frac{|f(x_1)|}{||w|| \times (1+g_1)} + \frac{|f(x_2)|}{||w|| \times (1+g_2)} = \frac{1}{||w||} + \frac{1}{||w||} = \frac{2}{||w||} \tag{3.37}$$

Since x_1 and x_2 are the closest samples to the hyperplane, the distance of other samples from the hyperplane (based on Equation (3.36)) is larger than 1. Therefore, we have:

$$\begin{cases} f(x_i)/(1+g_i) \geq 1 & \forall x_i \in \omega_1 \\ f(x_i)/(1+g_i) \leq -1 & \forall x_i \in \omega_2 \end{cases} \tag{3.38}$$

We define:

$$y_i = \begin{cases} +1 & \forall x_i \in \omega_1 \\ -1 & \forall x_i \in \omega_2 \end{cases} \tag{3.39}$$

Substituting Equation (3.39) in (3.38) results in:

$$y_i f(x_i)/(1+g_i) \geq 1, \ \ \forall x_i \tag{3.40}$$

We need to maximize the margin ($2/\|w\|$) in Equation (3.37) which is equivalent to minimizing the norm $\|w\|$. The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$\begin{cases} minimize\ J(w, w_0) = \frac{1}{2}\|w\|^2 = \frac{1}{2}w^T w & (3.41) \\ subject\ to\ y_i\left(\frac{w^T x_i + w_0}{1 + g_i}\right) \geq 1 \quad , \quad i = 1,2, \dots, N & (3.42) \end{cases}$$

where N is the number of training samples. The corresponding Lagrangian function $\mathcal{L}(w,w_0, \lambda)$ for the above convex programming problem is defined as follows:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - \sum_{i=1}^{N} \lambda_i \left[y_i \left(\frac{w^T x_i + w_0}{1 + g_i} \right) - 1 \right] \quad (3.43)$$

where λ_i, i=1,2,…,N are the Lagrangian multipliers associated with the constraint in Equation (3.42). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda \geq 0}\, min_{w,w_0}\, \mathcal{L}(w, w_0, \lambda)$ [74; 75; 76; 77]. The Karush-Kuhn-Tucker conditions that $min_{w,w_0}\, \mathcal{L}(w, w_0, \lambda)$ has to satisfy are [74; 75; 76; 77]:

$$\begin{cases} \frac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} & (3.44) \\ \frac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w_0} = 0 \rightarrow \sum_{i=1}^{N} \frac{\lambda_i y_i}{1 + g_i} = 0 & (3.45) \\ \lambda_i \geq 0 \ , \quad i = 1,2, \dots, N & (3.46) \\ \lambda_i \left[y_i \left(\frac{w^T x_i + w_0}{1 + g_i} \right) - 1 \right] = 0 \ , i = 1,2, \dots, N \quad (complementary\ slackness\ conditions) & (3.47) \end{cases}$$

The conditions in Equation (3.47) state that either λ_i or $y_i[(w^T x+w_0)/(1+g_i)]$-1 must be zero. Therefore support vectors are training samples where $|w^T x+w_0|/(1+g_i)$=1 ($y_i[(w^T x+w_0)/(1+g_i)]$-1=0 and $\lambda_i \neq 0$). It is now clear how our modified distance function in Equation (3.36) affects the choice of support vectors. Before, support vectors were those

geometrically closest to the hyperplane but now a trade-off between the weight (g_i) and the geometrical distance to the hyperplane determines whether a training sample is a support vector or not.

By expanding Equation (3.43), we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} w^T w - w^T \sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} - w_0 \sum_{i=1}^{N} \frac{\lambda_i y_i}{1 + g_i} + \sum_{i=1}^{N} \lambda_i$$

By replacing $\sum_{i=1}^{N} \frac{\lambda_i y_i}{1+g_i} = 0$, from Equation (3.45) in the above equation, we get:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} w^T w - w^T \sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} + \sum_{i=1}^{N} \lambda_i$$

By substituting *w* from Equation (3.44) , we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right]^T \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right] - \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right]^T \sum_{i=1}^{N} \left[\frac{\lambda_i y_i x_i}{1 + g_i} \right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = -\frac{1}{2} \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right]^T \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\lambda_i \lambda_j y_i y_j x_i^T x_j}{(1 + g_i)(1 + g_j)}$$

Now we maximize the above Lagrangian function with respect to λ:

$$\begin{cases} \max\limits_{\lambda} \left(\sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \frac{\lambda_i \lambda_j y_i y_j x_i^T x_j}{(1 + g_i)(1 + g_j)} \right) & (3.48) \\ subject\ to \sum\limits_{i=1}^{N} \frac{\lambda_i y_i}{1 + g_i} = 0 & (3.49) \\ \lambda_i \geq 0 \qquad , i = 1,2, \dots, N & (3.50) \end{cases}$$

Once the optimal Lagrange multipliers (λ_i) have been computed by maximizing Equation (3.48), *w* is obtained by replacing them in Equation (3.44) and w_0 is computed as an average

value obtained using complementary slackness conditions in Equation (3.47) for support vectors ($\lambda_i \neq 0$). Labeling a new sample is no different here; if $f(x)=w^T x+w_0>0$, x is classified in ω_1, and otherwise in ω_2.

An interesting observation is that the term $(1+g_i)$ appears everywhere in the computations as a denominator of y_i. It means the weighted SVM can be obtained by replacing y_i with $y_i/(1+g_i)$ in non-weighted SVM computations.

3.2.3.2 Two linearly nonseparable classes If the two classes are not linearly separable which is usually the case in real-world problems, e.g., Figure 3.9, then it is not possible to find an empty band separating them. Each training sample will have one of the following constraints, as shown in Figure 3.9:

- it falls outside the band and is correctly classified, i.e., $y_i(w^T x_i+w_0)>1$,
- it falls inside the band and is correctly classified, i.e., $0 \leq y_i(w^T x_i+w_0) \leq 1$, or
- it is misclassified, i.e., $y_i(w^T x_i+w_0)<0$.

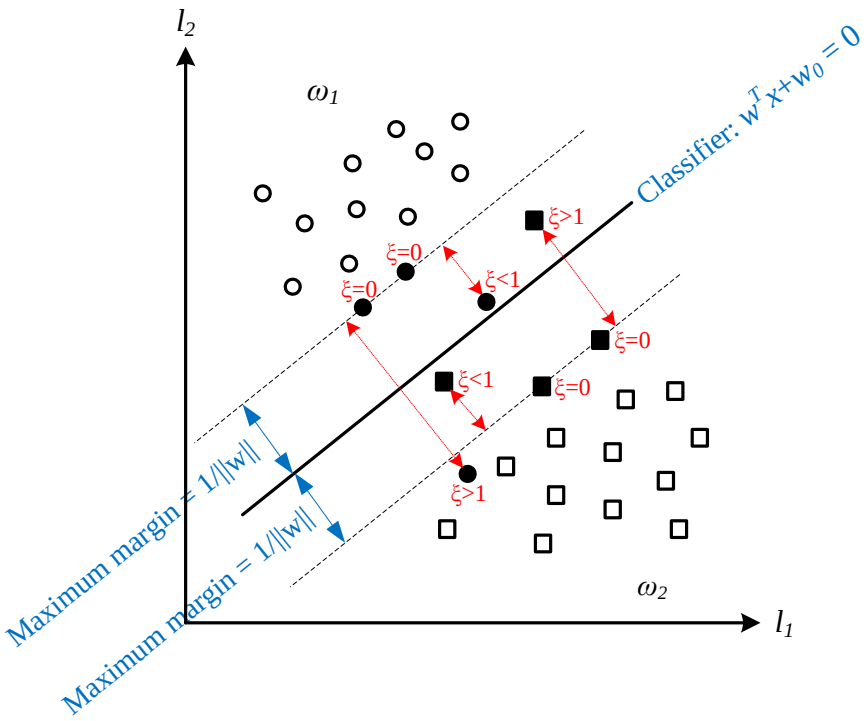


Figure 3.9. SVM classifier for two linearly nonseparable classes; black points show support vectors.

We can summarize the three above constraints in one by introducing the slack variable (ξ_i) [41]:

$$y_i(w^T x_i + w_0) \geq 1 - \xi_i \,, \begin{cases} \xi_i = 0 & if\ x_i\ is\ outside\ the\ band\ and\ correctly\ classified \\ 0 < \xi_i \leq 1 & if\ x_i\ is\ inside\ the\ band\ and\ correctly\ classified \\ \xi_i > 1 & if\ x_i\ is\ misclassified \end{cases} \quad (3.51)$$

The optimization task is now to maximize the margin (minimize the norm) while minimizing the slack variables [41]. The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$\begin{cases} minimize\ J(w, w_0, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i & (3.52) \\ subject\ to\ y_i(w^T x_i + w_0) \geq 1 - \xi_i \quad , i = 1,2,\dots,N & (3.53) \\ \xi_i \geq 0 \quad , i = 1,2,\dots,N & (3.54) \end{cases}$$

The smoothing parameter C is a positive user-defined constant that controls the trade-off between the two competing terms in the cost function. The two terms are against each other because minimizing the norm (i.e., maximizing the margin) increases the slack variables by increasing the number of training samples inside the band. On the other hand, decreasing the number of samples inside the band is equivalent to decreasing the margin. Therefore, by choosing a very large $C \to \infty$, the width of the margin disappears, $2/\|w\| \to 0$, because we allow the norm to grow much faster than slack variables (ξ_i). The corresponding Lagrangian function $\mathcal{L}(w, w_0, \xi, \lambda, \mu)$ for the above convex programming problem is defined as follows [74; 75; 76; 77]:

$$\mathcal{L}(w, w_0, \xi, \lambda, \mu) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i - \sum_{i=1}^{N} \lambda_i [y_i (w^T x_i + w_0) - 1 + \xi_i] \qquad (3.55)$$

where λ_i, $i=1,2,\ldots,N$ are the Lagrangian multipliers associated with the constraint in Equation (3.53) and μ_i, $i=1,2,\ldots,N$ are the Lagrangian multipliers associated with the constraint in Equation (3.54). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda \geq 0} \, min_{w, w_0, \xi} \, \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ [74; 75; 76; 77]. The Karush-Kuhn-Tucker conditions that $min_{w, w_0, \xi} \, \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ has to satisfy are [74; 75; 76; 77]:

$$\begin{cases} \dfrac{\partial \mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial w} = 0 \to w = \displaystyle\sum_{i=1}^{N} \lambda_i y_i x_i & (3.56) \\ \dfrac{\partial \mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial w_0} = 0 \to \displaystyle\sum_{i=1}^{N} \lambda_i y_i = 0 & (3.57) \\ \dfrac{\partial \mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial \xi_i} = 0 \to C - \mu_i - \lambda_i = 0 \;\; , \;\; i = 1,2, \ldots, N & (3.58) \\ \mu_i \xi_i = 0 \;\; , \;\; i = 1,2, \ldots, N & (3.59) \\ \mu_i \geq 0 \;\; , \;\; \lambda_i \geq 0 \;\; , \;\; i = 1,2, \ldots, N & (3.60) \\ \lambda_i [y_i (w^T x_i + w_0) - 1 + \xi_i] = 0 \;\; , i = 1,2, \ldots, N & (3.61) \\ (complementary\ slackness\ conditions) & \end{cases}$$

Equations (3.56) and (3.57) depend only on training samples whose $\lambda_i \neq 0$, referred to as support vectors. On the other hand, the conditions in Equation (3.61) state that either λ_i or $y_i(w^T x+w_0)-1+\xi_i$ must be zero. Therefore support vectors are training samples where $y_i(w^T x+w_0)=1-\xi_i$ ($y_i(w^T x+w_0)-1+\xi_i=0$ and $\lambda_i \neq 0$). Therefore, correctly classified training samples outside the margin are not support vectors because we have $y_i(w^T x+w_0)>1$ and $y_i(w^T x+w_0)-1+\xi_i$ cannot be zero considering $\xi_i \geq 0$. It means that support vectors are those on the edge of the margin ($\xi_i=0$), correctly classified inside the margin ($0<\xi_i<1$), or misclassified ($\xi_i \geq 1$), as shown in Figure 3.9. From Equations (3.58) and (3.59), we can see that $\lambda_i=C$ for support vectors falling inside the margin ($\xi_i>0$) and $0<\lambda_i<C$ for support vectors falling on the edge of the margin ($\xi_i=0$). Therefore, Equations (3.56) and (3.57) depend only on support vectors and consequently the hyperplane classifier is designed only based on support vectors and is independent of other training samples because their λ_i is zero.

By expanding Equation (3.55), we have:

$$\mathcal{L}(w, w_0, \xi, \lambda, \mu) = \frac{1}{2} w^T w + \sum_{i=1}^{N} C\xi_i - \sum_{i=1}^{N} \mu_i \xi_i - w^T \sum_{i=1}^{N} \lambda_i y_i x_i - w_0 \sum_{i=1}^{N} \lambda_i y_i + \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \lambda_i \xi_i$$

$$\mathcal{L}(w, w_0, \xi, \lambda, \mu) = \frac{1}{2} w^T w + \sum_{i=1}^{N} (C - \mu_i - \lambda_i)\xi_i - w^T \sum_{i=1}^{N} \lambda_i y_i x_i - w_0 \sum_{i=1}^{N} \lambda_i y_i + \sum_{i=1}^{N} \lambda_i$$

By replacing $\sum_{i=1}^{N} \lambda_i y_i = 0$ from Equation (3.57) and $C-\mu_i-\lambda_i=0$ from Equation (3.58), we get:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} w^T w - w^T \sum_{i=1}^{N} \lambda_i y_i x_i + \sum_{i=1}^{N} \lambda_i$$

By substituting w from Equation (3.56), we end up with:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] - \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = -\frac{1}{2}\left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

Now we maximize the above Lagrangian function with respect to λ:

$$\begin{cases} \max_{\lambda}\left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j\right) & (3.62) \\ subject\ to \sum_{i=1}^{N} \lambda_i y_i = 0 & (3.63) \\ 0 \le \lambda_i \le C\ (due\ to\ Equation\ 3.20)\ \ , i = 1,2,\dots,N & (3.64) \end{cases}$$

Once the optimal Lagrange multipliers (λ_i) have been computed, by maximizing (3.62), w is obtained by replacing them in Equation (3.56) and w_0 is computed as an average value obtained using complementary slackness conditions in Equation (3.61) for support vectors ($\lambda_i \neq 0$). However, ξ_i is also unknown in Equation (3.61). We know from Equations (3.58) and (3.59) that ξ_i is zero for training samples whose $\lambda_i < C$. Therefore, if we only use the training samples whose $0 < \lambda_i < C$ (support vectors falling on the edge of the margin) to find w_0 via Equation (3.61), we can consider $\xi_i = 0$.

In the linearly nonseparable case the Lagrangian multipliers (λ_i) are bounded above by C, which is the only difference between the linearly separable and nonseparable cases. The slack variables, ξ_i, and their associated Lagrangian multipliers, μ_i, are not involved in finding the classifier hyperplane but their effect is indirectly felt through C [38].

The weighted version of SVM needs to be more sensitive to training samples with larger weights (g_i). In other words, the distance from training samples to the classifier hyperplane needs to be compromised based on their weights. However, the modified distance function in case of

two nonseparable classes is different than separable classes. When the two classes are separable, we always move training samples toward the classifier hyperplane by a factor proportional to their weight because training samples are always on the correct side of the classifier hyperplane. On the other hand, in case of two nonseparable classes, a training sample might lie on the wrong side of the classifier hyperplane. Therefore, if a training sample lies on the correct side of the classifier hyperplane, we should move it toward the hyperplane and otherwise away from it by a factor proportional to its weight. This way we increase the sensitivity of the classifier to training samples with large weights and raise their chances to be selected as support vectors. We introduce the modified distance function for weighted SVM, in case of two nonseparable classes, as:

$$\begin{cases} \frac{|f(x_i)|}{\|w\|} \times \left(\frac{1}{1+g_i}\right) = \frac{|f(x_i)|}{\|w\|} - \left(1 - \frac{1}{1+g_i}\right)\frac{|f(x_i)|}{\|w\|} & if\ x_i\ is\ correctly\ classified \\ \frac{|f(x_i)|}{\|w\|} \times \left(1 + \frac{g_i}{1+g_i}\right) = \frac{|f(x_i)|}{\|w\|} + \left(1 - \frac{1}{1+g_i}\right)\frac{|f(x_i)|}{\|w\|} & if\ x_i\ is\ not\ correctly\ classified \end{cases} \quad (3.65)$$

We define:

$$y_i = \begin{cases} +1 & \forall x \in \omega_1 \\ -1 & \forall x \in \omega_2 \end{cases} \rightarrow y_i \frac{f(x_i)}{|f(x_i)|} = \begin{cases} +1 & if\ x_i\ is\ correctly\ classified \\ -1 & if\ x_i\ is\ not\ correctly\ classified \end{cases} \quad (3.66)$$

Using Equation (3.66), we can combine the two distance functions in (3.65) in one:

$$\frac{|f(x_i)|}{\|w\|} - \left(y_i \frac{f(x_i)}{|f(x_i)|}\right)\left(1 - \frac{1}{1+g_i}\right)\frac{|f(x_i)|}{\|w\|} = \frac{|f(x_i)| - y_i\left(1 - \frac{1}{1+g_i}\right)f(x_i)}{\|w\|} \quad , \forall x_i \quad (3.67)$$

By scaling w and w_0, and introducing the slack variable (ξ_i) we can define the following constraint for training samples:

$$|f(x_i)| - y_i\left(1 - \frac{1}{1+g_i}\right)f(x_i) \geq 1 - \xi_i \ , \begin{cases} \xi_i = 0 & if\ x_i\ is\ outside\ the\ band\ and\ correctly\ classified \\ 0 < \xi_i \leq 1 & if\ x_i\ is\ inside\ the\ band\ and\ correctly\ classified \\ \xi_i > 1 & if\ x_i\ is\ misclassified \end{cases} \quad (3.68)$$

The optimization task is now to maximize the margin (minimize the norm) while minimizing the slack variables (ξ_i). The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$\begin{cases} minimize\ J(w, w_0, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i & (3.69) \\ subject\ to\ |w^T x_i + w_0| - y_i\left(1 - \frac{1}{1+g_i}\right)(w^T x_i + w_0) \geq 1 - \xi_i \quad , i = 1,2,\dots,N & (3.70) \\ \xi_i \geq 0 \quad , i = 1,2,\dots,N & (3.71) \end{cases}$$

The corresponding Lagrangian function $\mathcal{L}(w,w_0,\xi,\lambda,\mu)$ for the above convex programming problem is defined as follows:

$$\mathcal{L}(w, w_0, \xi, \lambda, \mu) = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\lambda_i\left[|w^T x_i + w_0| - y_i\left(1 - \frac{1}{1+g_i}\right)(w^T x_i + w_0) - 1 + \xi_i\right] \quad (3.72)$$

where λ_i, i=1,2,…,N are the Lagrangian multipliers associated with the constraint in (3.70) and μ_i, i=1,2,…,N are the Lagrangian multipliers associated with the constraint in (3.71). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda\geq 0}\, min_{w,w_0,\xi}\, \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ [74; 75; 76; 77]. The Karush-Kuhn-Tucker conditions that $min_{w,w_0,\xi}\, \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ has to satisfy are [74; 75; 76; 77]:

$$\begin{cases} \frac{\partial\mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^T x_i + w_0|}{w^T x_i + w_0} - y_i\left(1 - \frac{1}{1+g_i}\right)\right) & (3.73) \\ \frac{\partial\mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial w_0} = 0 \rightarrow \sum_{i=1}^{N}\lambda_i\left(\frac{|w^T x_i + w_0|}{w^T x_i + w_0} - y_i\left(1 - \frac{1}{1+g_i}\right)\right) = 0 & (3.74) \\ \frac{\partial\mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial \xi_i} = 0 \rightarrow C - \mu_i - \lambda_i = 0 \quad , \quad i = 1,2,\dots,N & (3.75) \\ \mu_i\xi_i = 0 \quad , \quad i = 1,2,\dots,N & (3.76) \\ \mu_i \geq 0 \quad , \quad \lambda_i \geq 0 \quad , \quad i = 1,2,\dots,N & (3.77) \\ \lambda_i\left[|w^T x_i + w_0| - y_i\left(1 - \frac{1}{1+g_i}\right)(w^T x_i + w_0) - 1 + \xi_i\right] = 0 \ , i = 1,2,\dots,N & (3.78) \\ (complementary\ slackness\ conditions) & \end{cases}$$

By expanding Equation (3.72), we have:

$$\mathcal{L}(w,w_0,\xi,\lambda,\mu)=\frac{1}{2}w^Tw+\sum_{i=1}^{N}C\xi_i-\sum_{i=1}^{N}\mu_i\xi_i-\sum_{i=1}^{N}\lambda_i\left[|w^Tx_i+w_0|-y_i\left(1-\frac{1}{1+g_i}\right)(w^Tx_i+w_0)\right]+\sum_{i=1}^{N}\lambda_i-\sum_{i=1}^{N}\lambda_i\xi_i$$

$$\mathcal{L}(w,w_0,\xi,\lambda,\mu)=\frac{1}{2}w^Tw+\sum_{i=1}^{N}(C-\mu_i-\lambda_i)\xi_i-\sum_{i=1}^{N}(w^Tx_i+w_0)\lambda_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

By replacing C-μ_i-λ_i=0 from Equation (3.75), we get:

$$\mathcal{L}(w,w_0,\lambda)=\frac{1}{2}w^Tw-\sum_{i=1}^{N}(w^Tx_i+w_0)\lambda_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

$$\mathcal{L}(w,w_0,\lambda)=\frac{1}{2}w^Tw-w^T\sum_{i=1}^{N}\lambda_ix_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+w_0\sum_{i=1}^{N}\lambda_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

By replacing $\sum_{i=1}^{N}\lambda_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)=0$ from Equation (3.74), we have:

$$\mathcal{L}(w,w_0,\lambda)=\frac{1}{2}w^Tw-w^T\sum_{i=1}^{N}\lambda_ix_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

$$\mathcal{L}(w,w_0,\lambda)=\sum_{i=1}^{N}\lambda_i+w^T\left(\frac{1}{2}w-\sum_{i=1}^{N}\lambda_ix_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]\right)$$

By substituting w from Equation (3.73), we end up with:

$$\mathcal{L}(w,w_0,\lambda)=\sum_{i=1}^{N}\lambda_i+\left[\sum_{i=1}^{N}\lambda_ix_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]^T\left(\frac{1}{2}\left[\sum_{i=1}^{N}\lambda_ix_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]-\left[\sum_{i=1}^{N}\lambda_ix_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]\right)$$

$$\mathcal{L}(w,w_0,\lambda)=\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\left[\sum_{i=1}^{N}\lambda_ix_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]^T\left[\sum_{i=1}^{N}\lambda_ix_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]$$

Now we maximize the above Lagrangian function with respect to λ:

$$\begin{cases}\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\left[\sum_{i=1}^{N}\lambda_ix_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]^T\left[\sum_{i=1}^{N}\lambda_ix_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]\right) & (3.79)\\ subject\ to\ \sum_{i=1}^{N}\lambda_i\left(\frac{|w^tx_i+w_0|}{w^tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)=0 & (3.80)\\ 0\le\lambda_i\le C\ \ (due\ to\ Equation\ 3.34)\quad,\quad i=1,2,\dots,N & (3.81)\end{cases}$$

Once the optimal Lagrangian multipliers (λ_i) have been computed by maximizing Equation (3.79), w is obtained by replacing them in Equation (3.73) and w_0 is computed as an average value obtained using complementary slackness conditions in Equation (3.78) for support vectors whose $0<\lambda_i<C$ and considering $\xi_i=0$. Labeling a new sample is no different here; if $f(x)=w^Tx+w_0>0$, x is classified in ω_1, and otherwise in ω_2.

Maximizing the above Lagrangian function with respect to λ is not as easy as Equation (3.62) because this time w and w_0 are involved in the process of finding the Lagrangian multipliers (λ_i) in Equation (3.79) while they are unknown. The appearance of w and w_0 in Equation (3.79) originates from the dichotomy in the distance function in Equation (3.65). Therefore, an iterative optimization technique must be adopted:

- *w and w_0 are initialized for a non-weighted SVM,*
- *Loop: repeat until convergence*
    - *λ_i are calculated using Equation (*3.79)
    - *w and w_0 are calculated using Equations (*3.73) *and (*3.78)

The time complexity of the above algorithm is k times more than the time complexity of finding the non-weighted SVM classifier hyperplane ($O(N^3)$ with a naïve implementation of a quadratic programming solver [38]), where k is the number of iterations in the loop. Since w and w_0 are initialized using a non-weighted SVM, the convergence is expected to happen in a few iterations.

From a geometric point of view, the loop in the above algorithm is updating the classifier hyperplane by redefining the distance of training samples to the hyperplane in each iteration. This is equivalent to relocating the training samples after each iteration with respect to the hyperplane classifier based on their weights and updating the classifier hyperplane based on the relocated training samples. Therefore, the following algorithm offers an alternative but

geometrically equivalent approach to the above algorithm with the same time complexity. The following algorithm can take advantage of existing software and libraries for non-weighted SVM to develop the weighted SVM.

- *w and w_0 are initialized for a non-weighted SVM,*
- *Loop: repeat until convergence*

$$\hat{X}_t = X - \left(1 - \frac{1}{1+g}\right).\left(\frac{\left|Xw_{t-1} + w_{0_{t-1}}\right|}{\|w_{t-1}\|}\right).y\frac{w_{t-1}^T}{\|w_{t-1}\|} \qquad (3.82)$$

  - *find w_t and w_{0t} for the non-weighted SVM classifier hyperplane based on $\hat{X}_t$*

where the subscript *t* stands for the iterator inside the loop. At the first step in the loop, *X* is the input feature matrix (each row representing one training sample), *y* is a column vector containing the responses, *w* is a column vector representing the norm of the classifier hyperplane, w_0 is the intercept of the classifier hyperplane, and g is a column vector containing the training samples' weights. The dot shows array (or element-wise) operations versus matrix operations shown with a cross. In Equation (3.82), $\left(1 - \frac{1}{1+g}\right).\left(\frac{|Xw+w_0|}{\|w\|}\right)$ is the magnitude we have to move the training samples, and $\left(-y\frac{w^T}{\|w\|}\right)$ is the movement direction. The movement magnitude is proportional to the training sample's weight. The movement is in the direction of the classifier's vector (*w*) for training samples in class ω_2 (*y*=-1) and in the opposite direction of *w* for training samples in class ω_1 (*y*=1). In other words, we have to update the position of a training sample by moving it $\left(1 - \frac{1}{1+g_i}\right)\left(\frac{|x_i w+w_0|}{\|w\|}\right)$ toward the classifier hyperplane if it is correctly classified or the same amount away from the hyperplane if it is wrongly classified. Therefore, training samples with large weights which were not normally selected as support vectors, now have a higher chance of being selected as support vectors if the aforementioned shift has dropped them inside the margin.

3.2.3.3 Experiment Due to SVM's stability to changes in a small part of the training data, the dataset in Table 3.1 cannot differentiate between the non-weighted and weighted SVM. In other words, the two classifiers are the same for that dataset. Instead, we use the dataset in Table 3.2 to show the effect of embedding training samples' weights in SVM. Figure 3.10 shows the division of the feature space between the two classes with and without considering the training samples' weights in computing the linear classifier. In weighted SVM, the important samples in class ω_1 will move toward class ω_2, through Equation (3.82), and repel the border toward class ω_2.

Table 3.2. Training samples and their weights for SVM.

l_1	l_2	Class	Spatial-temporal weight
1	1	ω_1	4
1	2	ω_1	2
1.4	1.5	ω_1	2
2	1	ω_2	1
2	2	ω_2	1

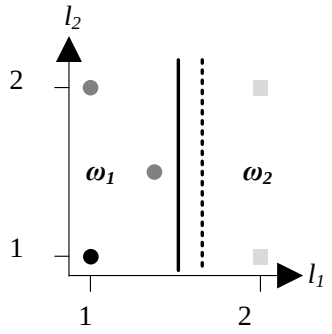


Figure 3.10. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in SVM classifier (C=1).

3.3 NONLINEAR PREDICTORS

If two classes are linearly separable in case of classification, or if responses have a linear relationship with features in case of regression, the linear models optimally find the regression or classification hyperplane. Otherwise, they find the hyperplane minimizing the prediction error based on some cost function. Nonlinear predictors, on the other hand, bend and curve themselves to get closer to the training samples in case of regression or to put more training samples on the correct side of the hypersurface in case of classification. However, some training samples might still remain far from the regressor or on the wrong side of the classifier depending on how flexible the nonlinear predictor is allowed to be.

One question which needs to be answered before designing a predictor is whether the relationship between responses and features is linear in case of regression, or whether the classes are linearly separable or not in case of classification. In other words, whether a linear or nonlinear predictor must be used. If training data show an approximately linear relationship between responses and features in case of regression, or if training samples are almost linearly separable in case of classification, assuming there is no more knowledge about the nature of the problem in hand, choosing a linear predictor is more sensible. One approach to find out whether a linear predictor is sufficient or not is to train a LS predictor and see how accurately it predicts the training samples' responses. Noise in training data might cause slight nonlinearity among responses. Therefore, if the LS predictor makes few mistakes in predicting the response of training samples, linear predictors are a cautious and conservative choice. However, if it turns out that the linear regression is far from being accurate or the classes are far from being linearly separable, nonlinear predictors are unavoidable. Yet, one needs to be careful with choosing a nonlinear predictor over a linear one to avoid capturing the particularities of small training

datasets in the predictor. In this section, we develop the weighted version of some nonlinear predictors.

3.3.1 Decision trees

Ordinary binary decision trees (OBDTs) split the feature space into hyperrectangles with sides parallel to the axes [44]. Nodes in an OBDT, shown in Figure 3.11, are binary questions whose answers are either yes or no and the answer to these questions determines the path to a leaf which is equivalent to a response (nominal label in classification or numerical estimate in regression). Questions at nodes are of the form "is $x_k \leq \alpha$?" where x_k is the k-th feature and α is a threshold. To predict the response of an irresponsive sample, one needs to answer the question at each node and traverse to the left or right node based on the answer until a leaf (response) is reached.

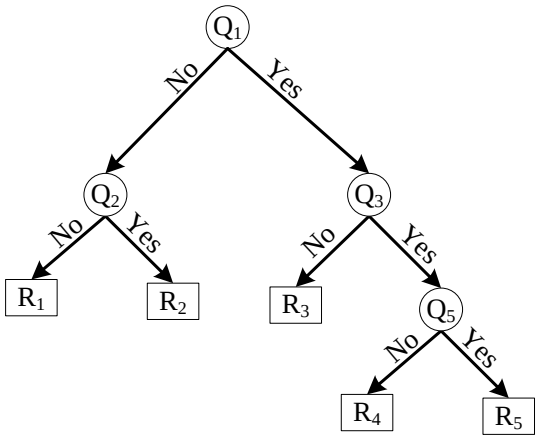


Figure 3.11. Ordinary binary decision trees; Q stands for question and R stands for response.

The training process involves designing the questions, structuring the tree, and associating each leaf with a response. Each node splits the training dataset into two disjoint groups, each corresponding to one of the answers: yes or no. Many questions can be asked at one node based on what feature (x_k) to choose and what threshold (α) to use. Different thresholds that can be considered for a specific feature at a node are determined based on the training samples at that node. For example, if there are N samples at a node, there could be N-1 different thresholds, each taken halfway between consecutive distinct values of x_k in the training samples at that node. Therefore, if there are l features and N training samples at a node, (N-1)×l different questions can be asked. The best question to ask at a node is the one which maximizes the impurity decrease (ΔI). The impurity decrease is calculated through Equation (3.83) [44]:

$$\Delta I = I - \frac{N_Y}{N} I_Y - \frac{N_N}{N} I_N \tag{3.83}$$

where I is the impurity of the ancestor node, N is the number of training samples in the ancestor node, N_Y is the number of training samples in the descendant node corresponding with the answer "yes" to the question, N_N is the number of training samples in the descendant node corresponding with the answer "no" to the question, and I_Y and I_N are the impurities of the descendent nodes. Entropy of training samples at a node, in Equation (3.84), is a common definition of node impurity in classification tasks ($I_{classification}$) [44], where N is the number of training samples at this node, M is the number of classes, and $N(\omega_i)$ is the number of training samples from class ω_i at this node. Therefore, in classification, impurity at a node is proportional to the heterogeneity of classes among training samples at that node. The largest impurity ($log_2 M$) happens when training samples are equally distributed among classes and the least impurity (0) happens when all training samples belong to the same class.

The impurity of a node in regression tasks ($I_{regression}$) is commonly calculated as the variance, in Equation (3.85), where y_i is the response of the i-th training sample at this node and $\bar{y}$ is the average of responses at this node.

$$I_{classification} = -\sum_{i=1}^{M} \frac{N(\omega_i)}{N} log_2 \frac{N(\omega_i)}{N} \tag{3.84}$$

$$I_{regression} = \frac{\sum_{i=1}^{N}(y_i - \bar{y})^2}{N} \tag{3.85}$$

A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than a user-defined threshold, although other alternative conditions have been used in the literature [44; 78]. The majority rule in case of classification or the average rule in case of regression are commonly used to determine the response at that leaf [44].

In the weighted version of OBDT, the impurity decrease (ΔI) and impurity (I) are calculated through the following equations:

$$\Delta I = I - \frac{\sum g_Y}{\sum g} I_Y - \frac{\sum g_N}{\sum g} I_N \tag{3.86}$$

$$I_{classification} = -\sum_{i=1}^{M} \frac{g(\omega_i)}{\sum g} \log_2 \frac{g(\omega_i)}{\sum g} \tag{3.87}$$

$$I_{regression} = \frac{\sum_{i=1}^{N} g_i(y_i - \bar{y})^2}{\sum_{i=1}^{N} g_i} \tag{3.88}$$

where Σg_Y and Σg_N are the sum of the weight of training samples corresponding to the answers “yes” and “no”, respectively, Σg is the sum of the weight of all training samples at the ancestor node, $g(\omega_i)$ is the sum of the weight of training samples belonging to class ω_i, and g_i is the i-th training sample’s weight.

A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than a user-defined threshold. In case of classification, the class with the largest total weight

($argmax_{\omega_j} \sum_{i \in \omega_j} g_i$) is associated with that leaf. In case of regression, the weighted average of the responses ($\sum_{i \in leaf} g_i y_i / \sum_{i \in leaf} g_i$) is associated with that leaf.

In the weighted decision tree, samples with larger weights play a more important role in deciding what question to ask at a node (by playing a more significant role in calculating impurity and impurity decrease), when to stop splitting the nodes, and what response to associate with a leaf.

3.3.1.1 Experiment Here we use the dataset in Table 3.1 to show the effect of embedding training samples' weights in decision tree. Figure 3.12 shows the division of the feature space between the two classes with and without considering the training samples' weights in developing the decision tree. The important samples from class ω_1 change the way the weighted decision tree divides the feature space between the two classes in comparison with non-weighted decision tree.

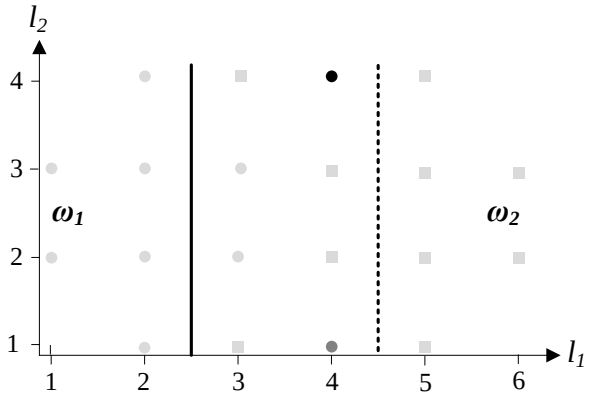


Figure 3.12. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in decision tree classifier (minimum impurity decrease for splitting a node is considered 0.1).

3.3.2 Multilayer perceptron (MLP)

In the backpropagation algorithm [45; 46; 47], the architecture of the network is fixed and its synaptic weights are computed so as to minimize a cost function defined as:

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \varepsilon(i) \tag{3.89}$$

where N is the number of training samples and $\varepsilon(i)$ is a function of the network's output ($\hat{y}(i)$) and the desired output ($y(i)$) for the i-th training sample. A common choice for $\varepsilon(i)$ is the sum of squared errors in the output nodes [79; 80; 46; 47]:

$$\varepsilon(i) = \frac{1}{2}\sum_{j=1}^{k_L} \left(\hat{y}_j^L(i) - y_j^L(i)\right)^2 \quad , \qquad i = 1,2,\dots,N \tag{3.90}$$

where L refers to the output layer, k_L represents the number of nodes in the output layer, $\hat{y}_j^L(i)$ represents the output of the j-th node in the output layer, and $y_j^L(i)$ represents its corresponding desired value. We also have the following equation for calculating the output of the j-th node at the r-th layer for the i-th training sample ($\hat{y}_j^r(i)$):

$$\hat{y}_j^r(i) = f_j^r\left(v_j^r(i)\right) \tag{3.91}$$

$$v_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r \hat{y}_k^{r-1}(i) \tag{3.92}$$

where f_j^r is the activation function at the j-th node of the r-th layer, k_{r-1} is the number of nodes at the (r-1)-th layer, $\hat{y}_k^{r-1}(i)$ is the output of the k-th node in the (r-1)-th layer, and w_{jk}^r is the synaptic weight from the k-th node at the (r-1)-th layer to the j-th node at the r-th layer.

We can iteratively find the synaptic weight vectors that minimize the perceptron cost function using the gradient descent scheme [45; 46; 47]. In each iteration, the weight vector

(including the threshold) of the *j*-th node in the *r*-th layer ($\boldsymbol{w}_j^r$) is modified through Equation (3.93):

$$\boldsymbol{w}_j^r(new) = \boldsymbol{w}_j^r(old) + \Delta\boldsymbol{w}_j^r \tag{3.93}$$

The modification term in Equation (3.93) ($\Delta\boldsymbol{w}_j^r$) is computed through Equation (3.94) according to the gradient descent scheme:

$$\Delta\boldsymbol{w}_j^r = -\alpha\frac{\partial J(\boldsymbol{w})}{\partial\boldsymbol{w}_j^r} \tag{3.94}$$

By substituting the cost function from Equation (3.89) in Equation (3.94) and applying the chain rule in differentiation, we obtain:

$$\Delta\boldsymbol{w}_j^r = -\alpha\frac{\partial\sum_{i=1}^{N}\varepsilon(i)}{\partial\boldsymbol{w}_j^r} = -\alpha\sum_{i=1}^{N}\frac{\partial\varepsilon(i)}{\partial\boldsymbol{w}_j^r} = -\alpha\sum_{i=1}^{N}\frac{\partial\varepsilon(i)}{\partial v_j^r(i)}\frac{\partial v_j^r(i)}{\partial\boldsymbol{w}_j^r} \tag{3.95}$$

By defining $\delta_j^r(i) = \frac{\partial\varepsilon(i)}{\partial v_j^r(i)}$ in the above equation, we obtain:

$$\Delta\boldsymbol{w}_j^r = -\alpha\sum_{i=1}^{N}\delta_j^r(i)\frac{\partial v_j^r(i)}{\partial\boldsymbol{w}_j^r} \tag{3.96}$$

We can calculate $\frac{\partial v_j^r(i)}{\partial\boldsymbol{w}_j^r}$ using Equation (3.92) as follows:

$$\frac{\partial v_j^r(i)}{\partial\boldsymbol{w}_j^r} = \begin{bmatrix}\frac{\partial v_j^r(i)}{\partial w_{j1}^r}\\ \vdots \\ \frac{\partial v_j^r(i)}{\partial w_{jk_{r-1}}^r}\end{bmatrix} = \hat{y}^{r-1}(i) \tag{3.97}$$

where k_{r-1} is the number of nodes in the (*r*-1)-th layer and $\hat{y}^{r-1}(i)$ is the output vector of the (*r*-1)-th layer for the *i*-th training sample. By substituting Equation (3.97) in Equation (3.96) we obtain:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} \delta_j^r(i)\hat{y}^{r-1}(i) \tag{3.98}$$

The above equation obtains the correction term for batch mode [72]. In online or pattern mode, instead of summing up the corrections over all training samples and updating the weights at once, the weights are updated once for each individual training sample before moving on to the next [72]. In stochastic mode, the gradient at each iteration is calculated based on a random subset of training samples [73].

Now we have to compute $\delta_j^r(i)$ based on the definition of the cost function given in Equation (3.90). First we calculate this term for the output layer (r=L):

$$\delta_j^L(i) = \frac{\partial \varepsilon(i)}{\partial v_j^L(i)} \tag{3.99}$$

By substituting Equations (3.90) and (3.91) in the above equation we get:

$$\delta_j^L(i) = \frac{\partial}{\partial v_j^L(i)}\left[\frac{1}{2}\sum_{m=1}^{k_L}\left(f_m^L\left(v_m^L(i)\right) - y_m^L(i)\right)^2\right] \tag{3.100}$$

By keeping only the terms that are dependent on $v_j^L(i)$ we get:

$$\delta_j^L(i) = \frac{\partial}{\partial v_j^L(i)}\left[\frac{1}{2}\left(f_j^L\left(v_j^L(i)\right) - y_j^L(i)\right)^2\right] = \left(\hat{y}_j^L(i) - y_j^L(i)\right)\frac{\partial f_j^L\left(v_j^L(i)\right)}{\partial v_j^L(i)} \tag{3.101}$$

where $\hat{y}_j^L(i)$ is the output of the j-th node in the output layer for the i-th training sample, $y_j^L(i)$ is its corresponding desired value, and f_j^L is the activation function of the j-th node in the output layer which takes $v_j^L(i)$ as input.

Now we compute $\delta_j^r(i)$ for hidden layers (r<L):

$$\delta_j^r(i) = \frac{\partial \varepsilon(i)}{\partial v_j^r(i)} = \sum_{k=1}^{k_{r+1}} \frac{\partial \varepsilon(i)}{\partial v_k^{r+1}(i)}\frac{\partial v_k^{r+1}(i)}{\partial v_j^r(i)} = \sum_{k=1}^{k_{r+1}} \delta_k^{r+1}(i)\frac{\partial v_k^{r+1}(i)}{\partial v_j^r(i)} \tag{3.102}$$

We use Equation (3.92) to calculate $\frac{\partial v_k^{r+1}(i)}{\partial v_j^r(i)}$:

$$\frac{\partial v_k^{r+1}(i)}{\partial v_j^r(i)} = \frac{\partial}{\partial v_j^r(i)}\left[\sum_{m=0}^{k_r} w_{km}^{r+1}\hat{y}_m^r(i)\right] \tag{3.103}$$

Replacing $\hat{y}_m^r(i)$ with $f_m^r\left(v_m^r(i)\right)$ based on Equation (3.91), we get:

$$\frac{\partial v_k^{r+1}(i)}{\partial v_j^r(i)} = \frac{\partial}{\partial v_j^r(i)}\left[\sum_{m=1}^{k_r} w_{km}^{r+1} f_m^r\left(v_m^r(i)\right)\right] \tag{3.104}$$

By keeping only the terms that are dependent on $v_j^r(i)$ we get:

$$\frac{\partial v_k^{r+1}(i)}{\partial v_j^r(i)} = \frac{\partial}{\partial v_j^r(i)}\left[\boldsymbol{w}_{kj}^{r+1} f_j^r\left(v_j^r(i)\right)\right] = \boldsymbol{w}_{kj}^{r+1}\frac{\partial f_j^r\left(v_j^r(i)\right)}{\partial v_j^r(i)} \tag{3.105}$$

Replacing the above equation in Equation (3.102), we obtain:

$$\delta_j^r(i) = \sum_{k=1}^{k_{r+1}} \delta_k^{r+1}(i)\boldsymbol{w}_{kj}^{r+1}\frac{\partial f_j^r\left(v_j^r(i)\right)}{\partial v_j^r(i)} \tag{3.106}$$

where k_{r+1} is the number of nodes in the (r+1)-th layer, $\boldsymbol{w}_{kj}^{r+1}$ is the synaptic weight from the j-th node in the r-th layer to the k-th node in the (r+1)-th layer, and f_j^r is the activation function of the j-th node in the r-th layer which takes $v_j^r(i)$ as input.

Now, we develop the weighted version of MLP. We include each training sample's weight ($g(i)$) in the cost function to adjust the MLP cost based on the importance of training samples. The MLP classifier will no longer be equally fair to all training samples. We modify the MLP cost function as in Equation (3.107) to punish the classifier more for misclassifying training samples with larger weights and less for training samples with smaller weights.

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} g_i\varepsilon(i) \tag{3.107}$$

We can compute the modification term ($\Delta \boldsymbol{w}_j^r$) through gradient descent scheme as follows:

$$\Delta \boldsymbol{w}_j^r = -\alpha \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}_j^r} = -\alpha \frac{\partial \sum_{i=1}^{N} g_i \varepsilon(i)}{\partial \boldsymbol{w}_j^r} = -\alpha \sum_{i=1}^{N} g_i \frac{\partial \varepsilon(i)}{\partial \boldsymbol{w}_j^r} \tag{3.108}$$

By applying the chain rule in differentiation:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} g_i \frac{\partial \varepsilon(i)}{\partial v_j^r(i)} \frac{\partial v_j^r(i)}{\partial \boldsymbol{w}_j^r} \tag{3.109}$$

By replacing $\frac{\partial \varepsilon(i)}{\partial v_j^r(i)}$ with $\delta_j^r(i)$, we obtain:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} g_i \delta_j^r(i) \frac{\partial v_j^r(i)}{\partial \boldsymbol{w}_j^r} \tag{3.110}$$

By applying Equation (3.97), we obtain:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} g_i \delta_j^r(i) \hat{y}^{r-1}(i) \tag{3.111}$$

Therefore, the only difference between the above equation for computing the correction term and Equation (3.98) is the presence of the training samples' weights (g_i) in the summand. If we define $\alpha^*(i)=\alpha g_i$, we obtain:

$$\Delta \boldsymbol{w}_j^r = -\sum_{i=1}^{N} \alpha^*(i) \delta_j^r(i) \hat{y}^{r-1}(i) \tag{3.112}$$

The above equation shows that the weighted version of the backpropagation algorithm for MLP is obtained by defining the training rate as $\alpha^*(i)=\alpha g_i$, which means a different training rate for each training sample based on its weight; remembering that the cost must also be calculated through Equation (3.107) which includes different weights for training samples. Adjusting the

training rate based on the weight of training samples and including the weights in the cost function bias the trained MLP in favor of training samples with larger weights.

3.3.2.1 Experiment Here we use the dataset in Table 3.1 to show the effect of embedding training samples' weights in the cost function (Equation (3.107)) and backpropagation algorithm (Equation (3.111)). The MLP is designed with one hidden layer including 2 nodes. Including more hidden nodes will result in all training samples being correctly classified in both non-weighted and weighted MLP, a zero classification cost for both non-weighted (Equation (3.89)) and weighted (Equation (3.107)) MLP, and consequently similar classifiers. With 2 hidden nodes some training samples cannot be correctly classified, so we can see the difference between non-weighted and weighted MLP classifiers. Figure 3.13 shows the division of the feature space between the two classes with and without considering the training samples' weights in the cost function and the backpropagation algorithm. Despite both weighted and non-weighted MLP misclassify the same four training samples, the weighted MLP classifier provides a better fit (a lower error) for the two more important samples from class ω_1.

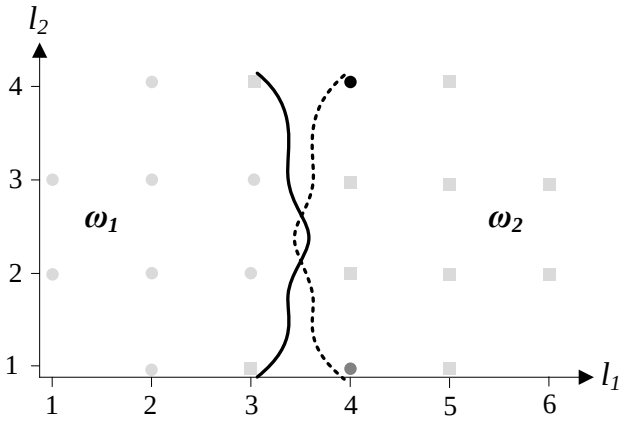


Figure 3.13. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in MLP classifier (logistic activation function and adaptive training rate with 2000 iterations).

3.3.3 Nonlinear SVM

In nonlinear SVM [48], training samples are nonlinearly mapped from their original *l*-dimensional space (where they cannot be linearly separated) into a *k*-dimensional space (*k*>>*l*) where they are more likely to be linearly separable [81; 38]. However, there is no guarantee that training samples will be linearly separable in the new *k*-dimensional space. Therefore, linear SVM with slack variables is used to find the hyperplane separating the two classes in the *k*-dimensional space. Although the classifier is a hyperplane in the *k*-dimensional space, it is a hypersurface in the *l*-dimensional space due to the nonlinear mapping, hence the name nonlinear SVM. The next step is to find the dimensionality of the new space (*k*) and the mapping function. We use the following equations, obtained in Section 3.2.3.2, to find the SVM classifier hyperplane $f(\acute{x})=w^T\acute{x}+w_0$ in the *k*-dimensional space, where $\acute{x}_i$ is the *i*-th feature vector (x_i) mapped into the *k*-dimensional space.

$$\begin{cases} \max\limits_{\lambda} \left(\sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \lambda_i \lambda_j y_i y_j \acute{x}_i^T \acute{x}_j \right) & (3.113) \\ subject\ to \sum\limits_{i=1}^{N} \lambda_i y_i = 0 & (3.114) \\ 0 \le \lambda_i \le C\ (due\ to\ Equation\ 3.20)\ \ , i = 1,2, \dots, N & (3.115) \end{cases}$$

$$\begin{cases} w = \sum\limits_{i=1}^{N} \lambda_i y_i \acute{x}_i & (3.116) \\ \lambda_i [y_i (w^T \acute{x}_i + w_0) - 1 + \xi_i] = 0\ \ , i = 1,2, \dots, N & (3.117) \\ (complementary\ slackness\ conditions) \end{cases}$$

By substituting *w* from Equation (3.116) in Equation (3.117) as well as in the hyperplane $f(\acute{x})=w^T\acute{x}+w_0$ we end up with:

$$\begin{cases} f(\acute{x}) = \left(\sum\limits_{i=1}^{N} \lambda_i y_i \acute{x}_i \right)^T \acute{x} + w_0 = \sum\limits_{i=1}^{N} \lambda_i y_i (\acute{x}_i^T \acute{x}) + w_0 & (3.118) \\ \lambda_i \left[y_i \left(\sum\limits_{j=1}^{N} \lambda_j y_j \left(\acute{x}_j^T \acute{x}_i \right) + w_0 \right) - 1 + \xi_i \right] = 0\ , & (3.119) \\ i = 1,2, \dots, N (complementary\ slackness\ conditions) \end{cases}$$

An elegant property of the SVM helps implicitly map the training samples into the *k*-dimensional space without knowing the mapping function and *k*. Notice that training samples enter into Equations (3.113), (3.118), and (3.119) in pairs, in the form of inner products ($\acute{x}_i^T \acute{x}_j$) in the *k*-dimensional space. Therefore, for finding *w* and w_0 of the hyperplane in the *k*-dimensional space and even for classifying a new sample using Equation (3.118), only the inner product of pairs of feature vectors in the *k*-dimensional space is required. Knowing the mapping function and the dimensionality of the new space (*k*) is not necessary. We can use the kernel trick to find the inner product of two feature vectors in the *k*-dimensional space without actually mapping them from the *l*-dimensional space into the *k*-dimensional space. According to Mercer's theorem, for any kernel (*K*), there exists a space in which $K(x_i,x_j)=\acute{x}_i^T \acute{x}_j$ [82; 83; 84]. Equations (3.120),

(3.121), and (3.122) are examples of kernel functions [83] called polynomial, radial basis function, and hyperbolic tangent, respectively, where σ is the kernel's bandwidth.

$$K(x_i, x_j) = (x_i^T x_j + 1)^q \quad , q > 0 \tag{3.120}$$

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right) \tag{3.121}$$

$$K(x_i, x_j) = tanh(\beta x_i^T x_j + \gamma) \quad \text{, for appropriate values of } \beta \text{ and } \gamma\text{, e.g., } \beta=2 \text{ and } \gamma=1 \tag{3.122}$$

Therefore, to convert the linear SVM to nonlinear SVM we just need to replace the inner product of the mapped feature vectors ($\acute{x}_i^T \acute{x}_j$) by a kernel function of the original feature vectors $K(x_i, x_j)$:

$$\begin{cases} \max\limits_{\lambda} \left(\sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \right) & (3.123) \\ subject\ to \sum\limits_{i=1}^{N} \lambda_i y_i = 0 & (3.124) \\ 0 \le \lambda_i \le C \ (due\ to\ Equation\ 3.20) \quad , i = 1,2, \dots, N & (3.125) \end{cases}$$

$$\begin{cases} f(x) = \sum\limits_{i=1}^{N} \lambda_i y_i K(x_i, x) + w_0 & (3.126) \\ \lambda_i \left[y_i \left(\sum\limits_{j=1}^{N} \lambda_j y_j K(x_i, x) + w_0 \right) - 1 + \xi_i \right] = 0 \ , i = 1,2, \dots, N & (3.127) \\ (complementary\ slackness\ conditions) \end{cases}$$

Although $f(x)$ is linear in the k-dimensional space, it is nonlinear in the l-dimensional space due to the nonlinearity of the kernel function.

Here we explain why we cannot develop the weighted version of nonlinear SVM. We use the following equations, obtained in Section 3.2.3.2, to find the weighted SVM classifier hyperplane $f(\acute{x})=w^T\acute{x}+w_0$ in the k-dimensional space, where $\acute{x}_i$ is the i-th feature vector (x_i) mapped into the k-dimensional space.

$$\begin{cases} \max\limits_{\lambda}\left(\sum\limits_{i=1}^{N}\lambda_i-\frac{1}{2}\left[\sum\limits_{i=1}^{N}\lambda_i\acute{x}_i\left(\frac{|w^T\acute{x}_i+w_0|}{w^T\acute{x}_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]^T\left[\sum\limits_{i=1}^{N}\lambda_i\acute{x}_i\left(\frac{|w^T\acute{x}_i+w_0|}{w^T\acute{x}_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]\right) & (3.128)\\ subject\ to\ \sum\limits_{i=1}^{N}\lambda_i\left(\frac{|w^t\acute{x}_i+w_0|}{w^t\acute{x}_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)=0 & (3.129)\\ 0\le\lambda_i\le C\ \ (due\ to\ Equation\ 3.34)\quad ,i=1,2,\dots,N & (3.130)\end{cases}$$

$$\begin{cases} w=\sum\limits_{i=1}^{N}\lambda_i\acute{x}_i\left(\frac{|w^T\acute{x}_i+w_0|}{w^T\acute{x}_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right) & (3.131)\\ \lambda_i\left[|w^T\acute{x}_i+w_0|-y_i\left(1-\frac{1}{1+g_i}\right)(w^T\acute{x}_i+w_0)-1+\xi_i\right]=0\ \ ,i=1,2,\dots,N & (3.132)\\ (complementary\ slackness\ conditions)\end{cases}$$

Training samples do not enter into Equations (3.128), (3.131), and (3.132) in the form of their inner products ($\acute{x}_i^T\acute{x}_j$) in the *k*-dimensional space, thus the kernel trick cannot be used here.

A plausible approach for developing the weighted nonlinear SVM is to develop the weighted linear SVM in the *k*-dimensional space using the iterative algorithm at the end of Section 3.2.3.2:

- *w and w_0 are initialized for a non-weighted SVM in the k-dimensional space,*
- *Loop: repeat until convergence*

$$\hat{\acute{X}}_t=\acute{X}-\left(1-\frac{1}{1+g}\right).\left(\frac{\left|\acute{X}w_{t-1}+w_{0_{t-1}}\right|}{\|w_{t-1}\|}\right).y\frac{w_{t-1}^T}{\|w_{t-1}\|} \qquad (3.133)$$

  - *find w_t and w_{0t} for the non-weighted SVM classifier hyperplane based on $\hat{\acute{X}}_t$*

The above algorithm attempts to develop the non-weighted linear SVM classifier, *f(ẋ)*, in the *k*-dimensional space, iteratively relocate the mapped feature vectors ($\hat{\acute{x}}_i$) with respect to this hyperplane based on their weights (Equation (3.133)), and find the new hyperplane *f($\hat{\acute{x}}_i$)* until convergence. However, the Mercer's theorem provides neither the dimensionality of the new space (*k*) nor the mapping function [85]. Therefore, it is not possible to map the feature vectors into the *k*-dimensional space and we do not know $\acute{X}$ in Equation (3.133).

To bypass the lack of knowledge about $\acute{X}$ in the k-dimensional space, one might consider updating the position of feature vectors in the original l-dimensional space with respect to the nonlinear classifier hypersurface based on their weights iteratively until convergence. However, the hypersurface in the l-dimensional space is not known. It is worth noting that even if the hypersurface was known, moving the training samples perpendicularly toward/away from a hypersurface is a challenging mathematical problem.

3.4 EXPERIMENT WITH BREAST CANCER DATA

The University of Wisconsin hospital breast cancer dataset was obtained from UCI machine learning repository [86]. This dataset has 683 samples, after samples with missing data are removed. This is a classification problem with 9 input features and 2 classes. The features include: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitoses. All input features are numerical values between 1 and 10. The two classes, that need to be predicted, are benign (444 samples) and malignant (239 samples). For weighted machine learning, we need a weight associated with each training sample, reflecting its reliability or accuracy. Due to the lack of such weights in this dataset (and to our knowledge in other machine learning datasets), we apply the following procedure to produce artificial weights for training samples. A function runs through each training sample, switches its output class from what it is to the other one with a random probability of $0<k<1$, and assigns a weight of $1-k$ to that training sample. This way, a training sample's weight shows how reliable that training sample is and there are no changes imposed on test samples. Weighted machine learning techniques take advantage of these weights

to take the training samples' reliability into account but non-weighted machine learning techniques ignore these weights. Leave-one-out or one-fold cross validation is used to estimate the accuracy of weighted and non-weighted machine learning techniques, reported in Table 3.3, where hyperparameters are optimized using cross-validation.

Table 3.3. Accuracy of different classification techniques for breast cancer prediction.

Technique	Overall accuracy (%)		Settings
	Non-weighted version	Weighted version	
Bayesian	46.71	76.72	• Non-parametric Parzen windows with Gaussian kernel (Equation 10) where σ=1 • Priors are based on class frequencies
LS	45.39	92.39	
SVM	51.24	64.86	• Smoothing parameter (C)=18
Decision tree	48.17	89.90	• A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than 0.04
Perceptron	51.10	93.12	• Logistic activation function • Cost function: Sum of squared errors • Maximum number of iterations: 1000 • Not updating the weights after those iterations resulting in an increase in the total cost • Multiply all learning rates by 1.1 or 0.8 after each step based on whether the total cost decreases or increases • Adaptive learning rate: multiply the learning rate for a parameter by 1.2 if the partial derivative of the loss, with respect to that parameter, remains the same sign in successive steps and multiply it by 0.7 otherwise [87]
MLP	55.64	89.02	In addition to the settings for the perceptron: • Maximum number of iterations: 2000 • Number of hidden nodes for MLP: 2

The higher accuracy of weighted machine learning techniques (see Table 3) comes as no surprise since they take advantage of weights while non-weighted machine learning techniques do not. Nevertheless, it proves the proposed weighted machine learning techniques' efficiency in

appropriately taking the training samples' weights into account, whenever such weights are available, in order to improve the prediction accuracy. Clearly, as mentioned earlier, the weighted SVM classifier shows the least difference with its non-weighted counterpart, underlying its relative reluctance to react to weights in comparison with other weighted predictors. The weighted least squares and weighted perceptron achieve the highest accuracy, shedding light on the linear separability of the two classes in this specific application. On the other hand, the weighted Bayesian classifier makes more dramatic changes in the border, resulting in a lower accuracy than other weighted non-linear classifiers (decision tree and MLP), for the same reason, i.e., the linear separability of the two classes in this specific application.

3.5 CONCLUSIONS

The weighted machine learning techniques developed in this chapter provide developers with the opportunity to give different weights to training samples. These weights are used to adjust the classifier/regressor in favor of more importance samples, and thereby giving a higher significance to more important samples. It is worth noting that the weighted linear and nonlinear classifiers change the division of the feature space only around the border (the most uncertain area) and areas far from the border are less likely to change their label. The weighted SVM classifier showed the least difference with its non-weighted counterpart when the training samples' weights are not much variant. The reason for this is that SVM classifier is designed only based on support vectors not all training samples. If the weights of training samples are not much different, their relocation based on their weights might not be large enough to change the selection of support vectors. In other words, the weighted SVM classifier would be different than

its non-weighted version only if relocating training samples based on weights would result in a rearrangement of training samples significant enough to change the selection of support vectors. In other words, the weighted SVM has the highest stability with respect to weight changes in a small subset of training samples. The weighted MLP also takes the training samples' weights into account through smallest adjustments in its nonlinear border. However, the MLP's behavior is highly dependent on the network's size. In other words, a larger number of hidden nodes will result in a more significant difference between the weighted and non-weighted MLP. On the other hand, the weighted decision tree and weighted Bayesian classifiers showed the most dramatic changes in how the feature space is divided between the two classes in comparison with their non-weighted counterparts. The reason is that these two models are highly local, especially the non-parametric Bayesian classifier. Therefore, even small changes in the training samples' weights would result in a different classification of the feature space. The weighted LS and perceptron showed slight and similar changes in how they divide the feature space between the two classes in comparison with their non-weighted counterparts. The similar behavior is because both models minimize the sum of square errors, although in different ways. The difference between weighted and non-weighted versions being slight in these two cases originates from their linear nature and consequently their rather restricted flexibility in modifying their shape. How the weighted machine learning techniques developed in this work will contribute in improving the prediction accuracy in different real-world applications is yet to be seen.

4.0 SPATIAL-TEMPORAL WEIGHT FOR TRAINING SAMPLES

In this chapter, we identify and quantify those characteristics of spatial-temporal data that make them different than other types of data and also affect the training process. It is the autocorrelation among spatial-temporal data which makes them special. If this spatial-temporal autocorrelation is quantitatively captured, it can be used as an external knowledge to enrich the training process. This external knowledge is entered in the training process as spatial-temporal weights assigned to training samples. Higher the spatial-temporal weight, more effective the training sample is and more biased the training process must be in its favor. Since machine learning techniques which take the training samples' weights into account were developed in the previous chapter, here we focus on developing a quantitative approach to assign a spatial-temporal weight to each training sample.

Semivariogram is used as the basis in calculating both spatial and temporal semivariances and the spatial-temporal weight is proportional to the inverse of the overall semivariance at specific spatial and temporal distances. To develop the spatial and temporal semivariograms and calculate the spatial-temporal weights, we only need the location, time, and response of training samples. Feature vectors are not needed to calculate the spatial-temporal weights.

4.1 SPATIAL SEMIVARIOGRAM

Autocorrelation or self-correlation assesses the similarity of characteristics at geographic locations relative to their spatial distance [2; 3]. In other words, a metric that relates the changes in responses to spatial distance is used. This metric will help us determine the level of similarity between the responses at two geographic locations, knowing their spatial distance.

A measure of spatial autocorrelation among training samples is semivariance (γ). Semivariance for the lag d is calculated through Equation (4.1) [2; 3; 4; 23; 11; 13; 25] where Δ is the lag interval, n_d is the number of observation pairs with a distance (d_{ij}) between d-$\Delta/2$ and d+$\Delta/2$, and y_i and y_j are the responses of the observations i and j, respectively. If the lags (d) are, for example, 10 m, 20 m, and 30 m, etc., then the lag interval (Δ) is 10 m. Only pairs of observations with a distance between d-$\Delta/2$ and d+$\Delta/2$ participate in the Σ. The hat (^) over the semivariance in this equation is to emphasize that the calculated value is the mean over all pairs with a distance of $d \mp \Delta/2$.

$$\hat{\gamma}(d) = \frac{1}{2n_d} \sum_{d_{ij}=d-\Delta/2}^{d+\Delta/2} \left(y_i - y_j\right)^2 \qquad (4.1)$$

The geographical fact that near things are more related than distant things means that the response of the observations which are geographically closer to each other is more likely to be similar than distant observations. For example, the landuse, weather, or population of people or animals in one region tends to be more similar to those of its surrounding regions than very far regions. This fact manifests itself as an increase in semivariance ($\hat{\gamma}$; see Equation (4.1)) with spatial lag (d) [2; 3; 4; 23; 11; 13; 24; 25] shown schematically in Figure 4.1.

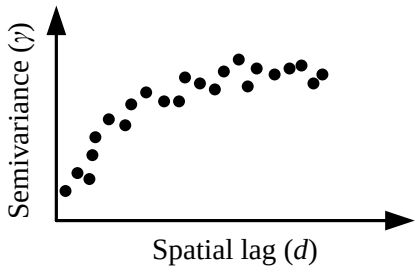


Figure 4.1. Semivariance versus spatial distance.

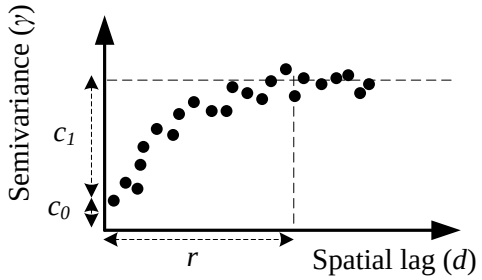


Figure 4.2. Nugget (c_0), partial sill (c_1), and range (r) in a semivariance versus spatial distance plot.

The increase in semivariance with spatial distance is not perpetual and the semivariance levels off and remains constant after some spatial lag [2; 3; 4; 23; 11; 13; 24; 25]. Sill (c_0+c_1) is the semivariance upper bound (see Figure 4.2). Partial sill (c_1) is the sill minus the nugget and is defined in Equation (4.2) where σ^2 is the variance of responses [2; 25]:

$$c_1 = \lim_{d_s \to \infty} \hat{\gamma}(d_s) \approx \sigma^2 \qquad (4.2)$$

The range (r) is the lag at which the semivariance reaches the sill and flattens out (see Figure 4.2). Beyond the range, there is no particular spatial autocorrelation structure among observations [2; 3; 25]. The nugget effect (c_0) presents a discontinuity in the semivariance at the origin (see Figure 4.2). The semivariance is always *0* at d=0. The nugget is the jump in the semivariance as soon as d>0 [88]. In other words, nugget is the semivariance at an infinitesimally small lag. The nugget is attributed to microscale variations (spatial variations at distances smaller than the shortest sampling interval) and measurement errors [89]. It is estimated as:

$$c_0 \approx \hat{\gamma}(0) \qquad (4.3)$$

Many empirical spatial semivariograms approximate to a spherical model [2; 3; 32; 24] shown in Equation (4.4) and visualized in Figure 4.3. The spherical model is the most frequently used model and is the default in many GIS software [90; 2; 3; 32; 24].

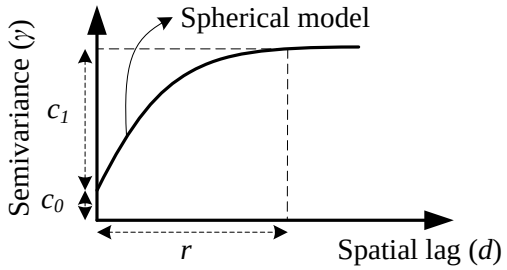


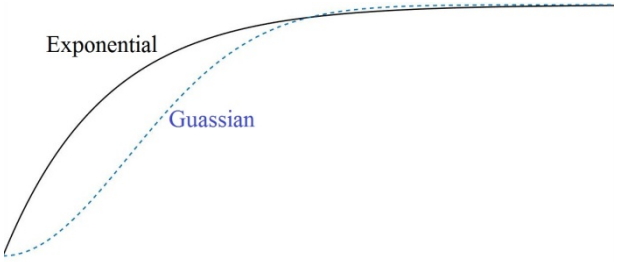
Figure 4.3. A spherical semivariogram model fitted to the semivariances in Figure 4.1.

$$\hat{\hat{\gamma}}(d_s) = \begin{cases} c_0 + c_1\left[\dfrac{3d_s}{2r} - 0.5\left(\dfrac{d_s}{r}\right)^3\right] & d_s \le r \\ c_0 + c_1 & d_s > r \end{cases} \tag{4.4}$$

The second hat (^) over the semivariance in Equation (4.4) is to emphasize that the calculated value is from the fitted semivariogram model and the subscript *s* in d_s is to emphasize that the distance is in the space domain (not the time domain).

Two other common semivariogram models [90] are also listed in Table 4.1, where both of them depend only on nugget (c_0), partial sill (c_1), and range (r).

Table 4.1. Common semivariogram models.

$\hat{\hat{\gamma}}(d_s) = c_0 + c_1\left[1 - \exp\left(\dfrac{3d_s}{r}\right)\right]$	Exponential model	
$\hat{\hat{\gamma}}(d_s) = c_0 + c_1\left[1 - \exp\left(\dfrac{3d_s^2}{r^2}\right)\right]$	Gaussian model	

The spatial semivariogram model in Equation (4.4), after c_0, c_1, and r are replaced with their values obtained from training samples, is used to show how strong is the correlation between each training sample and the irresponsive sample based on their spatial distance (d_s).

4.2 TEMPORAL SEMIVARIOGRAM

The semivariogram is also used to model the autocorrelation among the responses of observations over time rather than space. Equation (4.1) can be used to estimate the temporal semivariance, where d refers to the temporal distance between pairs of training samples rather than their spatial distance. The shape of temporal semivariogram might not necessarily be the same as spatial semivariogram. The autocorrelation among the responses of observations is more complicated over time than space because not only temporally closer observations are more likely to have similar responses than temporally farther observations [4; 15; 16], but also responses might exhibit a periodic behavior [17; 18; 19; 20; 21] over time as shown in Figure 4.4. For example, the temperature or weather today is more correlated with the temperature or weather yesterday than a month ago and it is more correlated with the temperature or weather a year ago than four months ago. In other words, the temporal semivariogram, shown in Figure 4.4, might never level off but rather show a periodic behavior. Another important point is that the responses might have more than one periodic behavior with different frequencies and amplitudes as exemplified in Figure 4.4. For example, there might be weekly, monthly, and yearly cycles with different amplitudes. Therefore, the temporal semivariogram, if stationary, is approximately the result of the random superposition of periodic components oscillating at different frequencies. It is important to mention that the nugget (c_0) in Figure 4.2 and Figure 4.4 are the

same because they are both referring to the semivariance at zero spatial and temporal lags (d_s=d_t=0).

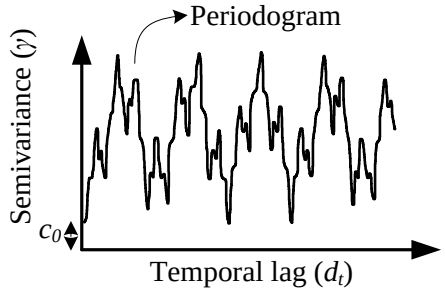


Figure 4.4. Semivariance versus temporal distance.

The sinusoid model $Acos(2\pi\omega d_t+\phi)$ captures the periodic behavior of the data at frequency ω where A is the amplitude and ϕ is the phase shift, determining the start point of the cosine function [91]. Applying the trigonometric identity ($\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$)) and defining $\beta_1=Acos\phi$ and $\beta_2=-Asin\phi$, we obtain the sinusoid in Equation (4.5) which is easier to regress on a dataset because it has two coefficients (β_1 and β_2) and one unknown parameter (ω) inside the cosine instead of one coefficient (A) and two unknown parameters inside the cosine (ω and ϕ) [91].

$$\hat{\hat{\gamma}}(d_t) = \beta_1 \cos(2\pi\omega d_t) + \beta_2 \sin(2\pi\omega d_t) \tag{4.5}$$

The two coefficients (β_1 and β_2) can be estimated through a linear regression via Equations (4.6) and (4.7) [91].

$$\hat{\beta}_1 = \frac{\sum_{d_t=1}^{n} \hat{\gamma}(d_t) \cos(2\pi\omega d_t)}{\sum_{d_t=1}^{n} \cos^2(2\pi\omega d_t)} = \frac{2}{n} \sum_{d_t=1}^{n} \hat{\gamma}(d_t) \cos(2\pi\omega d_t) \tag{4.6}$$

$$\hat{\beta}_2 = \frac{\sum_{d_t=1}^{n} \hat{\gamma}(d_t) \sin(2\pi\omega d_t)}{\sum_{d_t=1}^{n} \sin^2(2\pi\omega d_t)} = \frac{2}{n} \sum_{d_t=1}^{n} \hat{\gamma}(d_t) \sin(2\pi\omega d_t) \quad (4.7)$$

However, Equation (4.5) captures the periodic behavior at only one frequency (ω), while the data might oscillate at different frequencies (ω_i) with different amplitudes ($A=\sqrt{(\beta_1^2 + \beta_2^2)}$). Equation (4.8) shows different frequencies which need to be considered, where *n* is the number of training samples and *i* is the number of cycles through the life time of the data [91].

$$\omega_i = \frac{i}{n} \quad , i = 1, 2, \dots, \left\lceil \frac{n}{2} - 1 \right\rceil \quad (4.8)$$

To consider all frequencies, Equation (4.9) is used to fit a periodic regression on the semivariances based on temporal lag (d_t).

$$\hat{\hat{\gamma}}(d_t) = \sum_{i=1}^{\left\lceil \frac{n}{2}-1 \right\rceil} \beta_{i1} \cos(2\pi\omega_i d_t) + \beta_{i2} \sin(2\pi\omega_i d_t), \qquad \omega_i = \frac{i}{n} \quad (4.9)$$

The coefficients (β_{i1} and β_{i2}) in Equation (4.9) are calculated through Equations (4.10) and (4.11) for different frequencies (ω_i). We add frequencies associated with the largest amplitudes to the periodic regression in Equation (4.9) one by one as long as it produces a closer fit to data points.

$$\hat{\beta}_{i1} = \frac{2}{n} \sum_{d_t=1}^{n} \hat{\gamma}(d_t) \cos(2\pi\omega_i d_t), \qquad \omega_i = \frac{i}{n} \quad , i = 1, 2, \dots, \left\lceil \frac{n}{2} - 1 \right\rceil \quad (4.10)$$

$$\hat{\beta}_{i2} = \frac{2}{n} \sum_{d_t=1}^{n} \hat{\gamma}(d_t) \sin(2\pi\omega_i d_t), \qquad \omega_i = \frac{i}{n} \quad , i = 1, 2, \dots, \left\lceil \frac{n}{2} - 1 \right\rceil \quad (4.11)$$

Equation (4.9) models the semivariance of responses based on the time interval between observations and it can be used to determine how significant is the role of each training sample when we want to predict the response of a new sample.

4.3 SPATIAL-TEMPORAL WEIGHTS

Equation (4.12) calculates the spatial-temporal weight of the i-th training sample (g_i), where $d_s(ip)$ is the spatial distance between the i-th training sample and the irresponsive sample p, $d_t(ip)$ is the temporal distance between the i-th training sample and the irresponsive sample p, $\hat{\hat{\gamma}}(d_s(ip))$ is the spatial semivariance at $d_s(ip)$ calculated from Equation (4.4), $\hat{\hat{\gamma}}(d_t(ip))$ is the temporal semivariance at $d_t(ip)$ calculated from Equation (4.9), and $\hat{\hat{\gamma}}(ip)$ is the overall semivariance between the i-th training sample and the irresponsive sample p.

$$g_i = \frac{1}{\hat{\hat{\gamma}}(ip)} \qquad where, \quad \hat{\hat{\gamma}}(ip) = \frac{\hat{\hat{\gamma}}(d_s(ip)) + \hat{\hat{\gamma}}(d_t(ip))}{2} \tag{4.12}$$

4.4 DATA CONSTRAINTS

4.4.1 Fixed location or time

When calculating the spatial semivariance, the time must be the same for each pair of y_i and y_j in Equation (4.1). The same time does not mean the exact same instant and depends on the dataset. For example, if one set of observations are observed in one day (e.g., a series of satellite images taken on Jan 1st, 2001) and another set are observed in another day (e.g., another set of satellite images for the same area taken on Feb 1st, 2002), then the same time means observed on the same day. On the other hand, when calculating the temporal semivariance, the location must be the same for each pair of y_i and y_j in Equation (4.1). Again, the same location does not necessarily mean the same exact coordinates (x,y) but depends on the nature of the dataset. For

example, if each observation belongs to a separate tree or neighborhood (e.g., the height of the tree or the population of the neighborhood), then the same location means the same tree or the same neighborhood. For a satellite image, the same location means, e.g., the same pixel in the image.

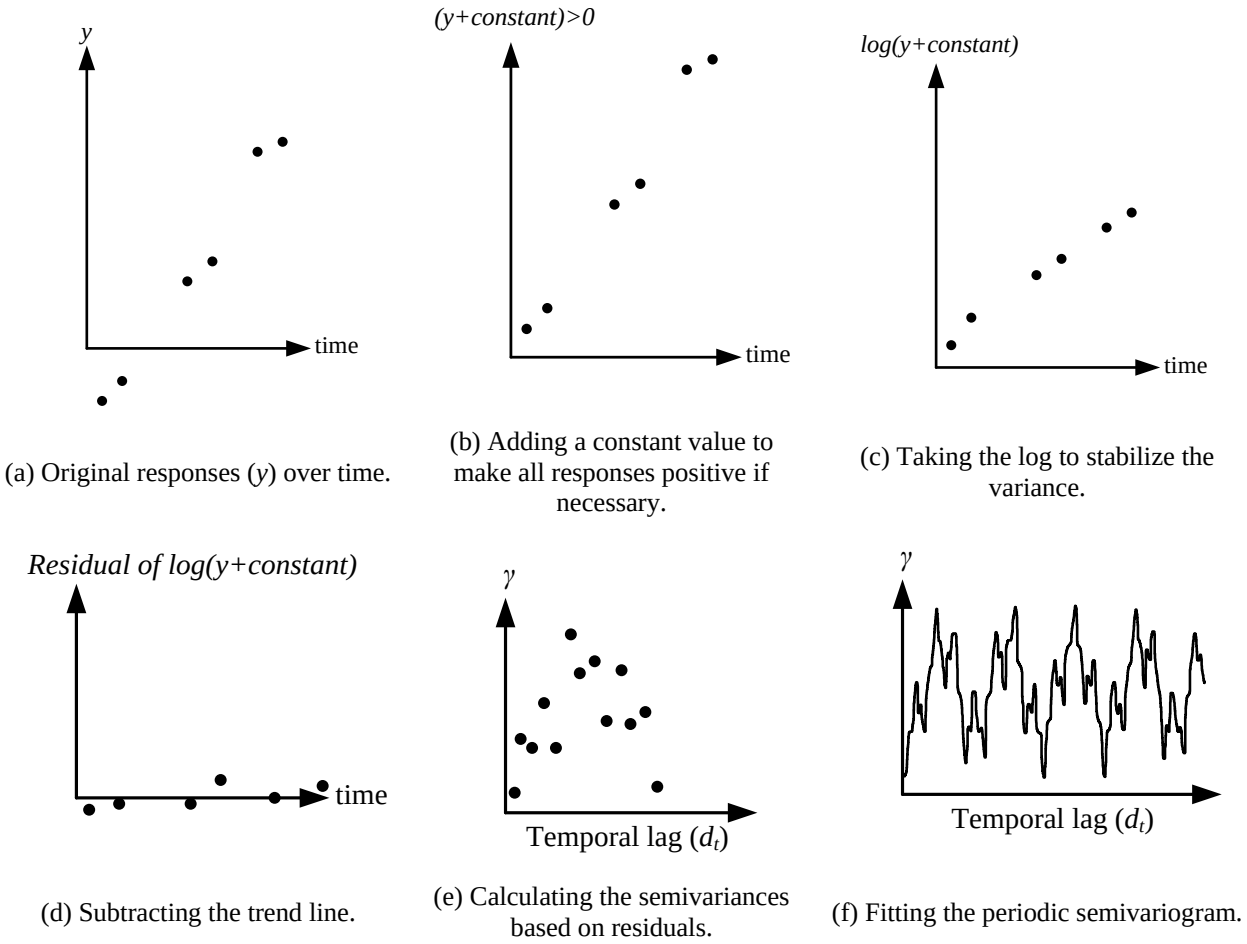
4.4.2 Categorical responses

As mentioned before, y_i in Equation (4.1) is the response of the observation *i*. If the responses are already numerical (interval- or ratio-scaled), they are used for *y*. Categorical (or non-numerical) responses are either ordinal or nominal [92]. If the responses are ordinal such as the agricultural potential of different lands or purity of different water bodies (e.g., good, average, and bad), they can be replaced by numbers (e.g., 1, 2, and 3) somehow that the interval between numbers approximates the implicit interval between responses (although ordinal data do not suggest any quantitative interval between levels). Although converting ordinal variables to interval variables in this way is not precise, it is legitimate here as weighted machine learning techniques, developed in this thesis, are not much sensitive to small changes in training samples' weights. If the responses are nominal, where the responses cannot be ordered (e.g., different landuses or building uses: shop, residential, etc.), assigning a number to each response incorrectly implies that some classes are closer to each other than others. In this case, we consider y_i-y_j=0 if samples *i* and *j* have the same response and 1 (or any desired constant value) otherwise [93; 94]. An alternative approach is to code responses via dummy variables [35]. In this approach responses are represented using vectors so as the distance between classes remains constant. The number of elements in the vector is equal to the number of classes. For each class, one element of the vector is one and the rest are zero (e.g., (1,0,0), (0,1,0), (0,0,1)). Hamming distance, L^1 or L^2 norms can

then be used to calculate the distance between response vectors. This approach has been common to code nominal features.

4.4.3 Stationarity of data

4.4.3.1 Temporal stationarity Fitting the periodogram to the temporal semivariances is only meaningful if the data are temporally stationary [91]. A stochastic process is strictly stationary if the joint statistical distribution of $\boldsymbol{x_{t_1}}, \dots, \boldsymbol{x_{t_l}}$ is the same as the joint statistical distribution of $\boldsymbol{x_{t_1+\tau}}, \dots, \boldsymbol{x_{t_l+\tau}}$ for all l and τ [95], where t represents the time. This means that statistical properties of all degrees (expectations, variances, third order, and higher) of the process, any where are the same. Since, strict stationary is too unrealistic for real-world processes, weak or second-order stationarity is defined as a process whose mean and variance do not vary with time and the autocovariance between x_t and $x_{t+\tau}$ (shown as $\text{cov}(x_t , x_{t+\tau})$) only depends on the lag τ [95]. We attempt to transform the data closer to a weakly stationary one by first stabilizing the variance and then stabilizing the mean. To stabilize the temporal variance, y is replaced by $log(y)$ for training samples. If there are negative values among responses, we can add a constant value to make them all positive and then take the *log*. This constant value will be removed in the next step. To stabilize the temporal mean, after stabilizing the temporal variance, a line, called the trend line, is fitted to all $log(y_i)$ based on time. Then, the value on the trend line is subtracted from $log(y_i)$. Temporal semivariances are calculated based on these residuals. Figure 4.5 summarizes these steps.



(a) Original responses (y) over time.

(b) Adding a constant value to make all responses positive if necessary.

(c) Taking the log to stabilize the variance.

(d) Subtracting the trend line.

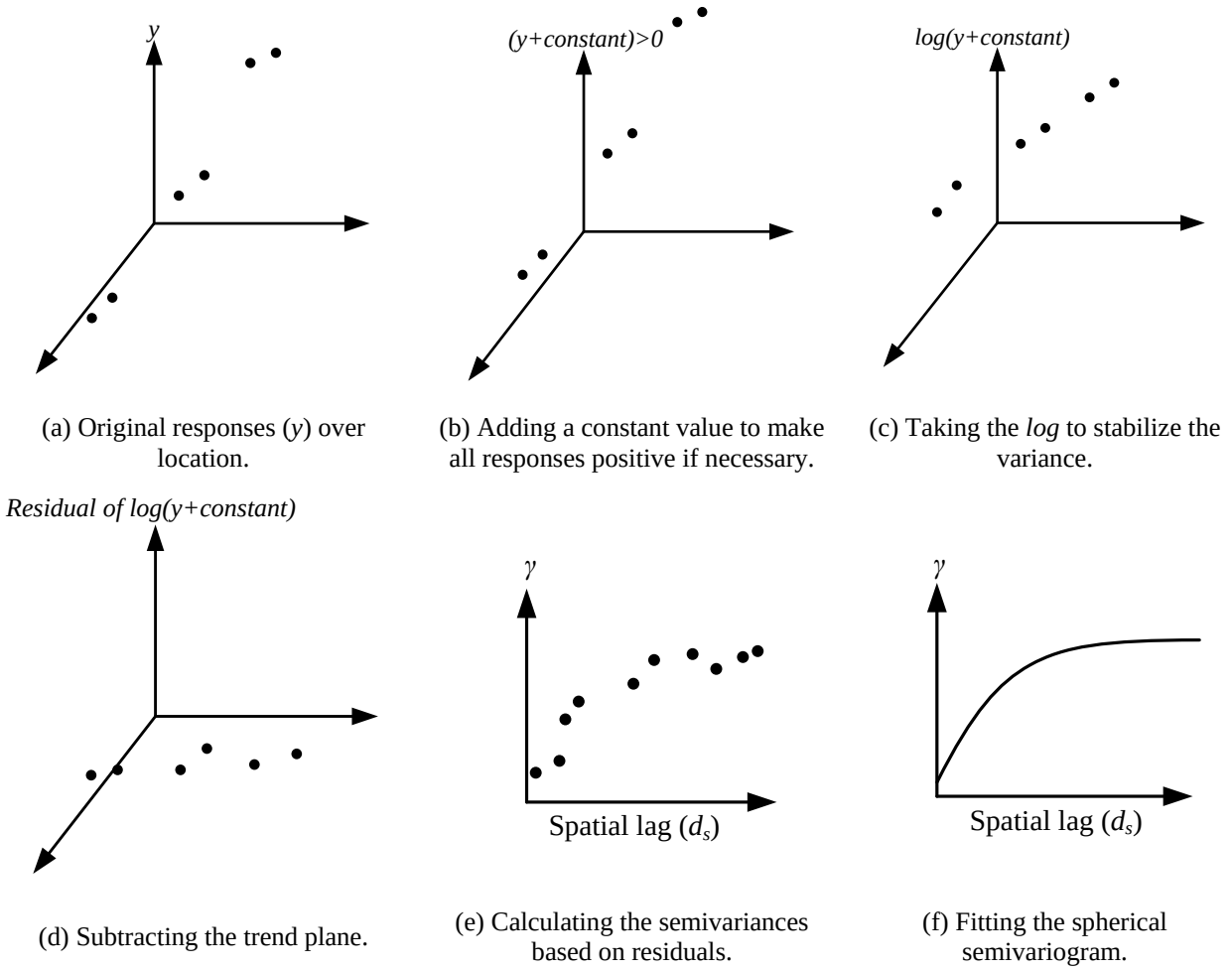
(e) Calculating the semivariances based on residuals.

(f) Fitting the periodic semivariogram.

Figure 4.5. Stabilizing the temporal variance and mean before developing the temporal semivariogram.

4.4.3.2 Spatial stationarity Fitting the spherical semivariogram to the spatial semivariances is only meaningful if the data are spatially stationary [24; 25]. If the spatial mean is not stable or, in other words, if there is a trend among the responses of training samples over space, the spatial semivariances will show an exponential behavior over lags and never flatten out. On the other hand, if the spatial variance is not stable or, in other words, if the range of changes in responses varies dramatically over space, the spatial semivariances will be dramatically scattered around the spherical model. Again, because real-world processes are far from being strictly stationary, we resort to weak stationarity. Thus, before calculating the spatial semivariances and fitting the

spherical semivariogram model, the spatial variance and mean must be stabilized. The steps are very similar to temporal stabilization and are illustrated in Figure 4.6. It is noteworthy that because space has two dimensions, in comparison to time with one dimension, here we have a trend plane instead of a trend line.



(a) Original responses (y) over location.

(b) Adding a constant value to make all responses positive if necessary.

(c) Taking the *log* to stabilize the variance.

(d) Subtracting the trend plane.

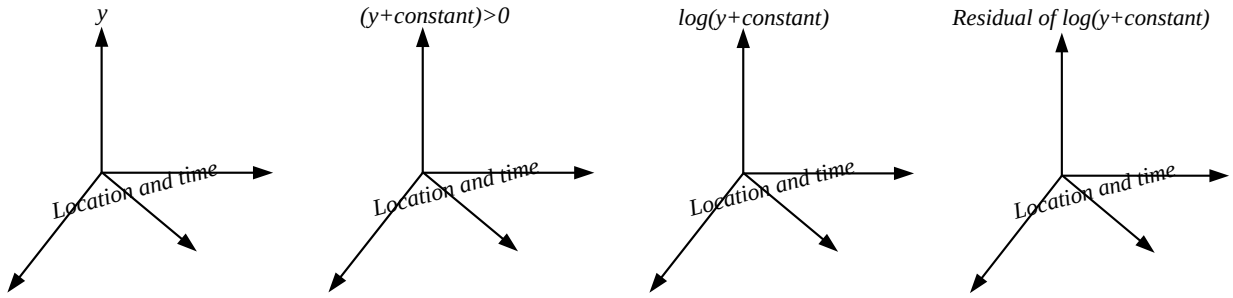
(e) Calculating the semivariances based on residuals.

(f) Fitting the spherical semivariogram.

Figure 4.6. Stabilizing the spatial variance and mean before developing the spatial semivariogram.

4.4.3.3 Combined spatial-temporal stationarity If our dataset is only spatial or temporal, we use either of the previous sections to stabilize the dataset. However, if our dataset is spatial and

temporal, we do not need to transform the dataset twice, once over location and once over time. Instead, the two processes are combined as shown in Figure 4.7. First, we add a constant value to make all responses positive. Then we take the *log* to stabilize the variance. To stabilize the mean, a 3D hyperplane is regressed over all responses based on both location and time and subtracted from all responses. These residuals are both spatially and temporally stabilized and we can proceed with developing semivariograms.



(a) Original responses (*y*) over location and time.

(b) Adding a constant value to make all responses positive if necessary.

(c) Taking the *log* to stabilize the variance.

(d) Subtracting the trend plane.

Figure 4.7. Stabilizing the variance and mean before developing the spatial and temporal semivariograms.

4.5 CONCLUSIONS AND FUTURE DIRECTIONS

Figure 4.8 shows the entire procedure of calculating the spatial-temporal weight for training samples. Spatial-temporal weights are visualized using color saturation. Darker samples have larger spatial-temporal weights. The red cross is the irresponsive sample and plays the central role in determining the spatial-temporal weight for training samples. Training samples are weighted based on their spatial and temporal autocorrelation with the irresponsive sample. In the

spatial-temporal domain, the irresponsive sample is at the origin of the coordinate system and spheres delineate the training samples with larger spatial-temporal weights.

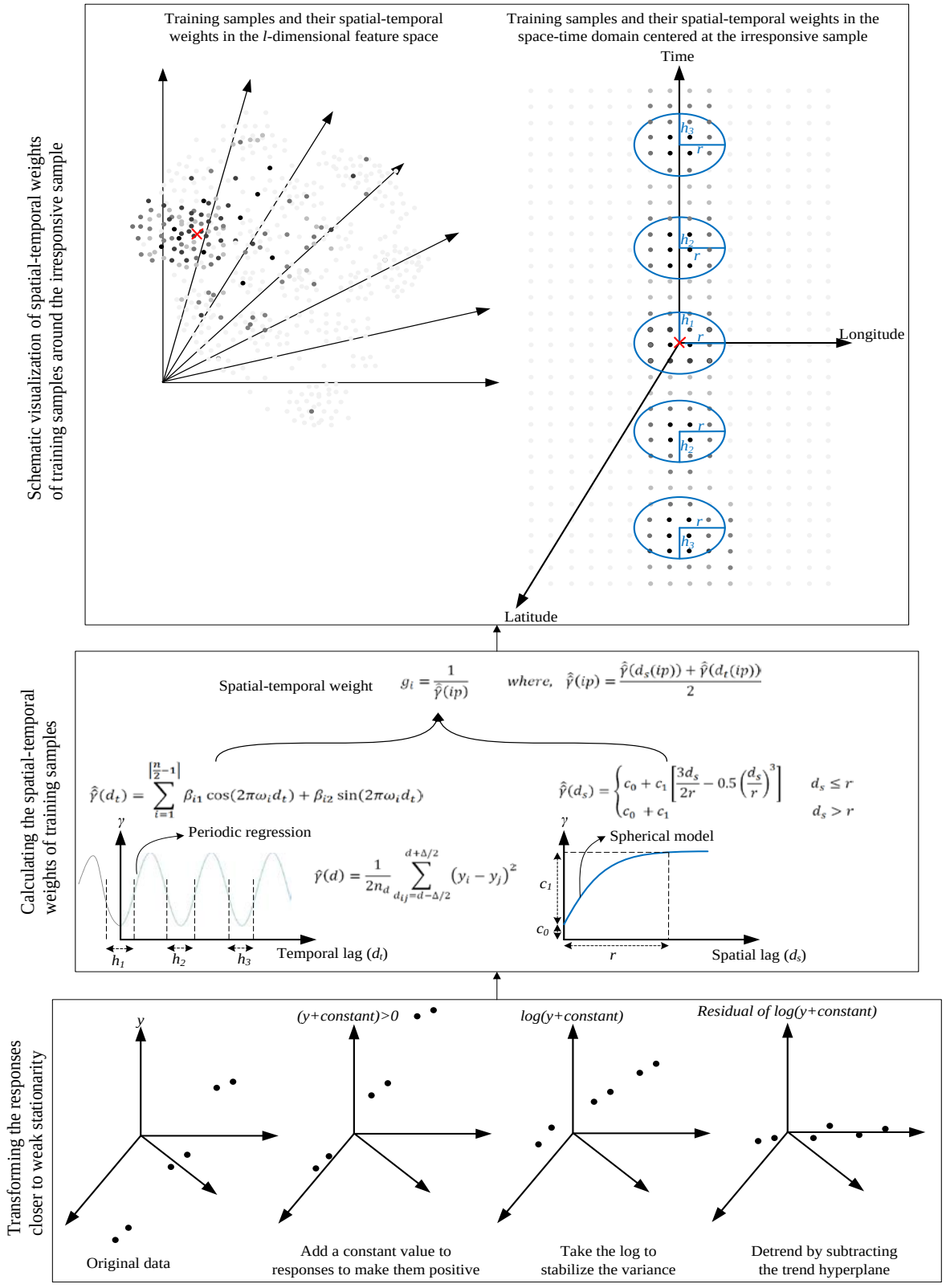


Figure 4.8. Calculating spatial-temporal weight for training samples based on their autocorrelation with the irresponsive sample.

It is worth noting that spatial-temporal weights assigned to training samples are dynamic with respect to the location and time of the irresponsive sample. In other words, spatial-temporal weights are relative and need to be recalculated every time a new observation is to be predicted. As a consequence, the weighted machine learning techniques need to be trained separately for each irresponsive sample, as the spatial-temporal weight of training samples depends on the location and time of the irresponsive sample.

The logic behind deciding in favor of spatial-temporally more important samples is that the output of the irresponsive sample is more likely to be similar to the output of training samples with larger spatial-temporal weights. This is justified because we calculate the spatial-temporal weights based on similarity among the responses of training samples across space and time. Therefore, we expect the spatial-temporally weighted predictor to achieve a better out-of-sample (test) accuracy than its non-weighted counterpart, despite it is not difficult to see that the non-weighted predictor achieves a better in-sample (training) accuracy since it is equally fair to all training samples. The scope of the proposed approach for calculating spatial-temporal weights is limited to spatial-temporal phenomena whose spatial semivariances approximately follow the spherical model and whose temporal semivariances approximately follow the periodic semivariogram proposed here. Additionally, the calculated spatial-temporal weights would be optimal when the data are stationary in the space-time domain.

Our approach of calculating spatial weights for training samples assumes that the underlying phenomenon is isotropic by considering only the distance between pairs of observations and ignoring the direction of the vector connecting them. A future research venue is to increase the accuracy of spatial weights by taking into account the anisotropy or directional effects in the spatial variation of responses. This can be done by developing two (or more) spatial

semivariograms, each modeling the spatial similarity in either north-south or east-west direction. To calculate the semivariance in a specific direction, only pairs of samples aligned in that direction (at least approximately) are used. Therefore, the direction of the vector connecting the irresponsive sample to each training sample determines which spatial semivariogram must be used to calculate that training sample's spatial weight. This way training samples aligned in a specific direction with respect to the irresponsive sample might gain a higher spatial weight.

We developed spatial and temporal semivariograms separately because the former is best modeled with a spherical model and the latter with a periodogram. Another future research venue is to find a way to develop a single spatial-temporal semivariogram.

5.0 EVALUATION

In Chapter 3, we developed weighted machine learning techniques which can use the training samples' weights to bias the predictor in favor of more important samples. In Chapter 4, we calculated a spatial-temporal weight for each training sample which determines its importance in classifying/regressing the irresponsive observation. In this chapter, the weighted machine learning techniques are applied to real spatial-temporal data. The spatial-temporal weight of training samples are estimated using the methodology proposed in Chapter 4 and then different weighted machine learning techniques developed in Chapter 3 are compared with their non-weighted versions and other baseline methodologies in terms of accuracy and time performance. The MATLAB software on a 64-bit platform with 8 GB RAM, a Core i7 CPU and a 2.00GHz processor was used for the validation of the proposed techniques. Two applications are presented in this chapter, regression of air temperature based on meteorological features and classification of land cover based on morphological and remote sensing features. A classification problem with simulated data is also presented at the end.

5.1 REGRESSION OF AIR TEMPERATURE

Oceanographic and surface meteorological readings, taken from a series of buoys positioned throughout the equatorial Pacific, from 1980 to 1998, are available to public in [96]. Each

reading includes location, date, zonal winds, meridional winds, humidity, and temperature. The dataset has 178,000 records. Removing records with missing features leaves the dataset with 94,000 records. In this experiment, we will predict the air temperature based on zonal winds, meridional winds, and humidity. Table 5.1 shows the variances and correlation coefficients between different variables in this dataset.

Table 5.1. Non-diagonal elements show the correlation coefficient between different variables and diagonal elements show the variances.

	Zonal winds	Meridional winds	Humidity	Temperature
Zonal winds	11.72	0.08	0.06	0.23
Meridional winds	0.08	9.13	0.08	-0.34
Humidity	0.06	0.08	27.83	-0.39
Temperature	0.23	-0.34	-0.39	2.80

According to the above table, air temperature is fairly correlated with all other variables while other variables are not much correlated with each other. The spatial and temporal semivariances for the response variable (air temperature), and the spatial and temporal semivariograms fitted to them are shown in Figure 5.1 and Figure 5.2. These figures are produced using the proposed approach in Chapter 4. As required in Section 4.4.1, we consider two observations to be at the same time as long as their time difference is less than one day and we consider two observations to be at the same location as long as their distance is less than 10 m.

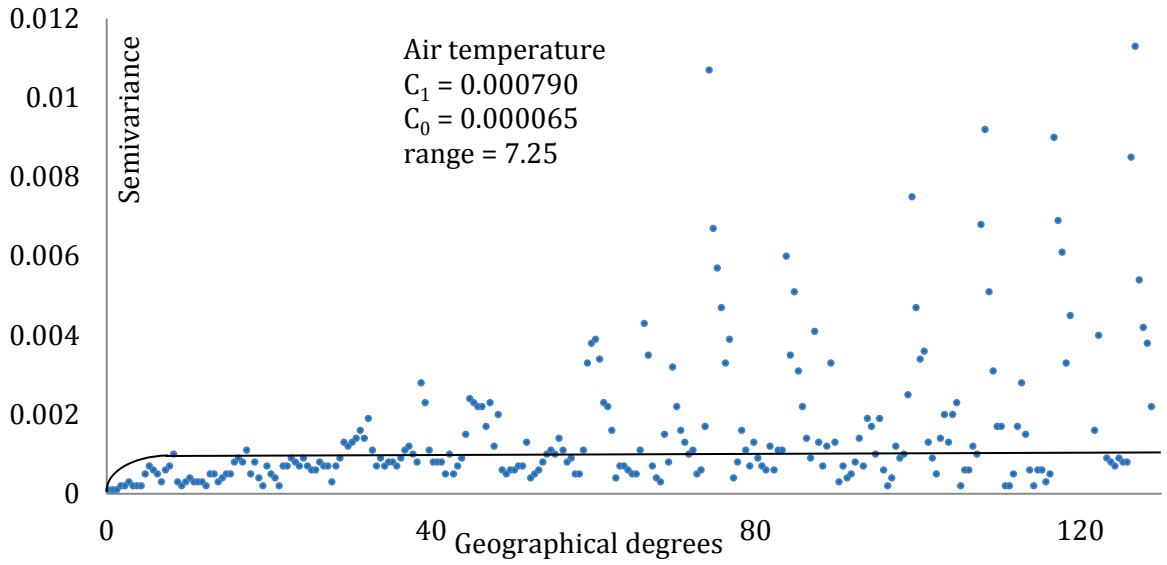


Figure 5.1. Spatial semivariogram for temperature.

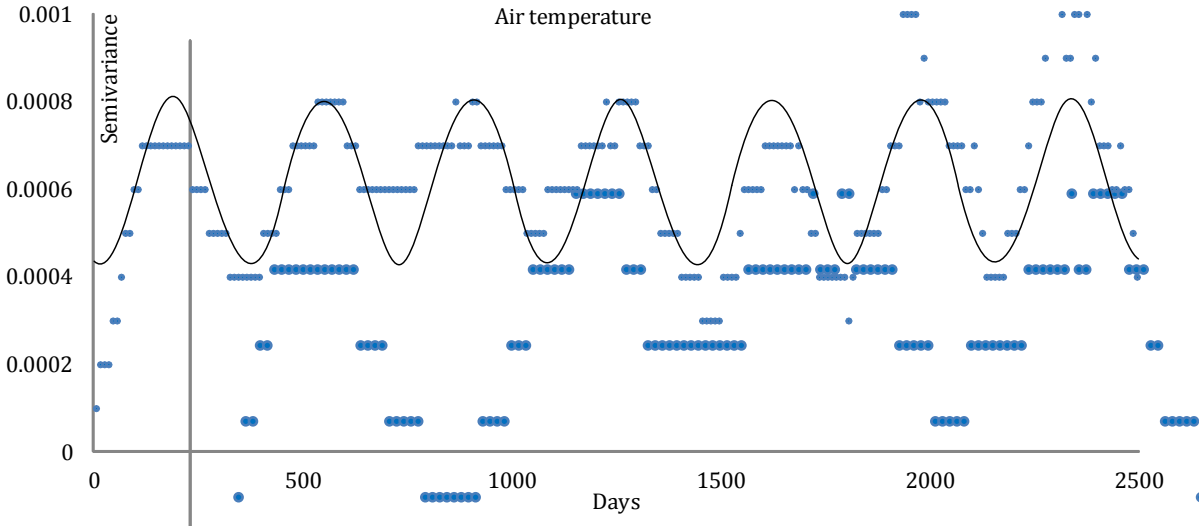


Figure 5.2. Temporal semivariogram for temperature with a period of one year.

Spatial and temporal semivariograms in the above figures can be used to determine the spatial-temporal weight of training samples. Knowing the spatial distance of a training sample to the irresponsive sample, we can use the spatial semivariogram in Figure 5.1 (represented by

Equation (4.4)) to calculate its spatial semivariance. Knowing the temporal distance of a training sample to the irresponsive sample, we can use the temporal semivariogram in Figure 5.2 (represented by Equation (4.9)) to calculate its temporal semivariance. Knowing both spatial and temporal semivariances for that training sample, we can use Equation (4.12) to calculate its spatial-temporal weight.

We use the leave-one-out or one-fold cross validation to evaluate the performance of weighted regression versus non-weighted regression. Each time, one of the training samples is excluded from the dataset and considered as the irresponsive sample. The spatial and temporal semivariance of each training sample is determined using Figure 5.1 and Figure 5.2 based on its spatial and temporal distances from the irresponsive sample. The spatial and temporal semivariances of training samples are converted to their spatial-temporal weight using Equation (4.12). Finally, the weighted Bayesian regressor, weighted LS, and weighted decision tree are applied to predict the response of the irresponsive sample. Following this, another training sample is chosen as the irresponsive sample and the whole process is repeated. All input features are normalized to have a zero mean and unit variance. Table 5.2 shows the accuracy and experimental time performance of different weighted regressors. Root mean square error (RMSE) and coefficient of determination (R^2) are reported for each regressor. The time performance includes the time spent to calculate the spatial-temporal weights in addition to the training and test time.

For comparison purposes, we also used only the top 30%, 10%, and 1% of training samples with largest spatial-temporal weights for training. The resulted accuracy and time performance are reported in Table 5.2. This table also reports the accuracy and time performance of non-weighted regressors trained with non-spatial features (zonal winds, meridional winds, and

relative humidity), non-weighted regressors trained with all features including location and time, non-weighted regressors trained with location and time as the only features, and finally estimating the response as the weighted average (i.e., spatial-temporal weights) of training samples' responses.

Table 5.2. The accuracy and time performance of different regression techniques for predicting the air temperature.

Technique	RMSE	R^2	Time (min)	Settings
Weighted average of training samples' responses	1.65	0.03	82	• Spatial-temporal weights of training samples are used as weights
Weighted average of the top 30% of training samples' responses	1.53	0.16	89	
Weighted average of the top 10% of training samples' responses	1.31	0.39	89	
Weighted average of the top 1% of training samples' responses	**1.12**	**0.55**	**89**	
LS with location and time as the only features	1.62	0.07	20	
LS without location and time as additional features	1.37	0.33	19	
LS with location and time as additional features	1.35	0.35	36	
Weighted LS	1.35	0.35	134	
Weighted LS using only the top 30% of training samples	1.26	0.43	123	
Weighted LS using only the top 10% of training samples	1.13	0.54	110	
Weighted LS using only the top 1% of training samples	**0.96**	**0.67**	**105**	
Bayesian regressor with location and time as the only features	1.40	0.30	4150	• Non-parametric Parzen windows with Gaussian kernel (Equation (3.4)) where σ=3
Bayesian regressor without location and time as additional features	1.31	0.39	4180	
Bayesian regressor with location and time as additional features	1.07	0.59	4522	
Weighted Bayesian regressor	1.27	0.42	4956	
Weighted Bayesian regressor using only the top 30% of training samples	1.16	0.52	1149	
Weighted Bayesian regressor using only the top 10% of training samples	1.04	0.62	579	
Weighted Bayesian regressor using only the top 1% of training samples	**0.92**	**0.70**	**154**	
Weighted decision tree using only the top 10% of training samples	1.11	0.56	41395	• A node is considered a leaf if ΔI_{max} is less than 0.2
Weighted decision tree using only the top 1% of training samples	**1.04**	**0.62**	**649**	

One apparent irregularity in the above table is that for the regressor that simply estimates the irresponsive sample's response as the weighted average of training samples' responses, the performance time does not reduce when training samples with small weights are excluded. It is because if all training samples participate in taking the weighted average, there is no need to sort the weights of training samples but if one decides to exclude the training samples with very small weights, the weight vector needs to be sorted. The sorting function which is invoked only if a subset of training samples needs to be applied, has a greater time complexity than the function that takes the weighted average of training samples' responses.

Non-parametric regressors, in our case the Bayesian regressor with Parzen windows, need to be trained separately for each irresponsive sample, regardless of the regressor being non-weighted or weighted. For parametric regressors, the weighted version needs to be trained separately for each irresponsive sample but the non-weighted version needs to be trained once for all irresponsive samples. In other words, the weighted regressors need to be trained as many times as the number of irresponsive samples because the spatial-temporal weights for training samples depend upon the location and time of the irresponsive sample. However, the leave-one-out approach for evaluation eliminates this effect, because each time there is only one test sample and consequently both weighted and non-weighted regressors are trained equal number of times.

The weighted and non-weighted Bayesian regressors have almost identical performance times with only 10% difference. The weighted LS takes 3.7 times longer than non-weighted LS which is due to the presence of training samples' weights in Equation (3.15).

Among the same versions of four different regressors (weighted average, LS, Bayes, and decision tree) in Table 5.2, the weighted average is the fastest approach, followed by LS, Bayesian regressor, and decision tree. Cross-validation of the weighted decision tree, trained

with only 10% of training samples, took 29 days. Cross-validation of a decision tree, trained with 30% of training samples (or more), takes months and the results are not reported in Table 5.2.

Despite its simplicity and lower accuracy in comparison with other more complicated techniques, the regressor which estimates the irresponsive sample's output as the weighted average of training samples' responses achieves a very acceptable accuracy in a very short time. Considering that this technique only needs location and time (to calculate the spatial-temporal weights), its high accuracy underscores the efficiency of spatial-temporal weights in depicting the spatial and temporal autocorrelation between training samples and the irresponsive sample.

Rows with a bold font in the above table show the settings leading to the best accuracy for each of the four regression techniques (weighted average, LS, Bayes, and decision tree). In all cases, the regressor trained using location and time as the only features results in the worst accuracy. This highlights the importance of non-spatial/temporal features in proper training. On the other hand, in all cases, the regressor that ignores the location and time altogether results in a lower accuracy than regressors which somehow take account of location and time, either as input features or as weights for training samples. This highlights the importance of location and time in proper training. The difference between (a) the accuracy of the non-weighted regressor trained using all features including location and time and (b) the accuracy of its weighted counterpart, is very small in all cases. However, when only a subset of training samples with largest spatial-temporal weights are applied to train the weighted regressor, the accuracy is considerably improved, revealing the best accuracy and performance time. This also uncovers the fact that training samples with very small spatial-temporal weights are not much constructive in training the regressor. This is in line with our expectations that weighted regression captures the spatial-

temporal autocorrelation best and excluding training samples with very small spatial-temporal weights not only reduces the performance time but also improves the accuracy.

Among the same versions of four different regressors (weighted average, LS, Bayes, and decision tree) in Table 5.2, the Bayesian regressor achieves the highest accuracy, followed by LS, decision tree, and weighted average. However, weighted decision tree seems to precede weighted LS, in terms of accuracy, when more training samples are included.

5.2 CLASSIFICATION OF LAND COVER

This is a classification problem with 8 input features and 8 classes. In this application we use bands 2, 3, 5, and 7 of Landsat TM images, two categorical features including geology—with six categories: (a) Quaternary alluvium, (b) Termeil essexite Permian, (c) Snapper point Permian, (d) Pebbly beach Permian, (e) Sedimentary Permian, and (f) Ordovician—and aspect—with five categories: east, north, west, south, and no aspect indicating zero slope—and two quantitative hydrological features including flow accumulation and flow length to predict the land covers shown in Table 5.3. Geology and aspect are coded in dummy variables [35]. All features are in 30×30 m grid format and the land covers represent the dominant vegetation cover by field observation for 1121 sites in Kioloa, New South Wales, Australia. These sites are not contiguous regions but are instead isolated samples as shown in Figure 5.3. The TM images are from Nov 8, 1994 and other layers are from 1992 [97; 98; 99; 100; 101; 59] and the entire dataset is publically available in [102].

This dataset is available only for one point in time. Therefore, the spatial weight is used instead of spatial-temporal weight during training. Figure 5.4 shows the spatial semivariances for

land covers and the spatial semivariogram fitted to them. This figure is produced using the proposed approach in Chapter 4.

Table 5.3. Different land covers and their relative frequency.

Class	dry sclerophyl	E. botryoides	lower slope wet	wet E. maculata	dry E. maculata	rainforest ecotone	rainforest	cleared land
Relative freq.	0.2569	0.0401	0.0401	0.2239	0.1588	0.0839	0.0705	0.1258

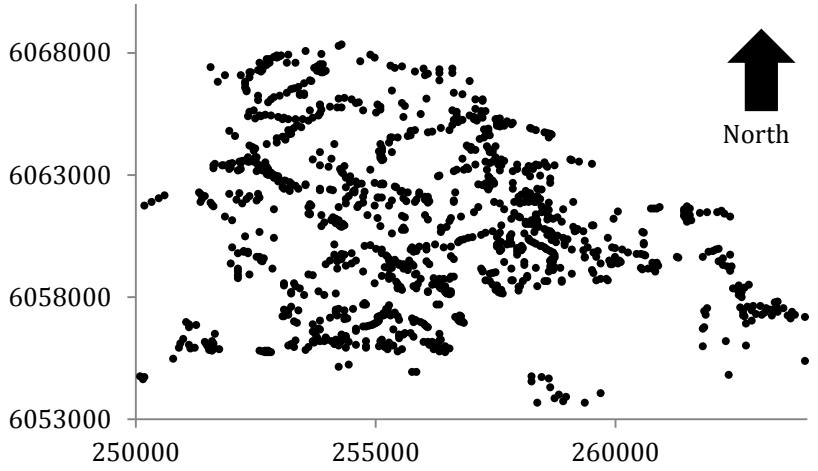


Figure 5.3. Distribution of land cover samples across space, in UTM zone 56S.

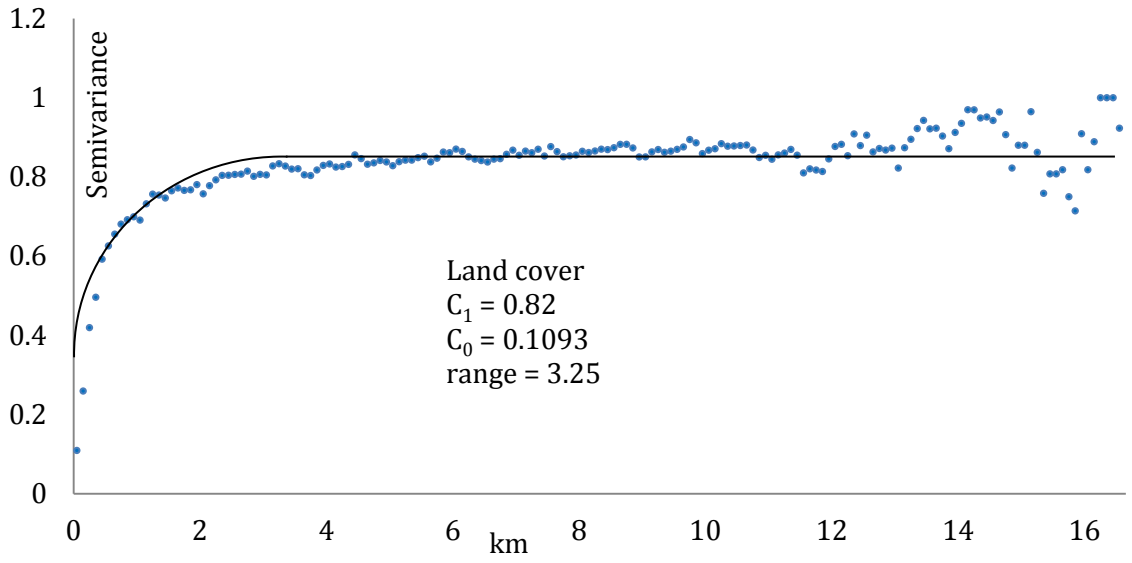


Figure 5.4. Spatial semivariogram for land covers.

The above spatial semivariogram can be used to determine the spatial weight of training samples. Knowing the spatial distance of a training sample to the irresponsive sample, we can use the spatial semivariogram in Figure 5.4 (represented by Equation (4.4)) to calculate its spatial semivariance. The spatial weight is the inverse of spatial semivariance.

We use the leave-one-out or one-fold cross validation to evaluate the performance of weighted Bayesian classifier, weighted LS, weighted perceptron, Weighted SVM, weighted decision tree, and weighted MLP. All input features are normalized to have a zero mean and unit variance. Table 5.4 shows the overall accuracy and experimental time performance of different weighted classifiers. The time performance includes the time spent to calculate the spatial weights in addition to the training and test time.

For comparison purposes, we also used only the top 30%, 10%, and 1% of training samples with largest spatial weights for training. The resulted accuracy and time performance are reported in Table 5.4. This table also reports the accuracy and time performance of non-weighted

classifiers trained with non-spatial features, non-weighted classifiers trained with all features including location, non-weighted classifiers trained with location as the only feature, and finally picking the class with the largest collective spatial weight among training samples.

Table 5.4. The accuracy and time performance of different classification techniques for predicting the land cover.

Technique	Overal l Acc.	Time (sec)	Settings
Choosing the class with largest collective weight among training samples	34.61	1	• Spatial-temporal weights of training samples are used as weights
Choosing the class with largest collective weight among the top 30% of training samples	31.85	1	
Choosing the class with largest collective weight among the top 10% of training samples	57.09	1	
Choosing the class with largest collective weight among the top 1% of training samples	**72.61**	**1**	
LS with location as the only feature	36.04	4	• Using the one-against-one scheme since there are more than two classes
LS without location as additional feature	14.99	12	
LS with location as additional feature	16.68	12	
Weighted LS	**52.72**	**33**	
Weighted LS using only the top 30% of training samples	42.64	13	
Weighted LS using only the top 10% of training samples	31.13	8	
Weighted LS using only the top 1% of training samples	22.03	2	
Bayesian with location as the only feature	**65.30**	**45**	• Non-parametric Parzen windows with Gaussian kernel (Equation (3.4)) where σ=85 • Priors are based on class frequencies
Bayesian without location as additional feature	14.09	293	
Bayesian with location as additional feature	11.41	304	
Weighted Bayesian	14.09	293	
Weighted Bayesian using only the top 30% of training samples	16.77	86	
Weighted Bayesian using only the top 10% of training samples	24.44	28	
Weighted Bayesian using only the top 1% of training samples	25.78	4	
Decision tree with location as the only feature	**74.22**	**2955**	• A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than 0.1.
Decision tree without location as additional feature	37.29	4399	
Decision tree with location as additional feature	67.35	15435	
Weighted decision tree	51.47	7317	

Weighted decision tree using only the top 30% of training samples	49.33	2620	
Weighted decision tree using only the top 10% of training samples	32.83	666	
Weighted decision tree using only the top 1% of training samples	22.75	20	
SVM with location as the only feature	19.98	2907	• Smoothing parameter (C)=5 • Using the one-against-one scheme since there are more than two classes
SVM without location as additional feature	28.99	14925	
SVM with location as additional feature	30.95	18174	
Weighted SVM using only the top 30% of training samples	**39.52**	**1184961**	
Weighted SVM using only the top 10% of training samples	29.97	297564	
Weighted SVM using only the top 1% of training samples	23.64	911	
Perceptron with location as the only feature	37.56	1777	• Logistic activation function • Cost function: Sum of squared errors • Maximum number of iterations: 1000 for perceptron and 3000 for MLP • Not updating the weights after those iterations resulting in an increase in the total cost • Multiply all learning rates by 1.1 or 0.8 after each step based on whether the total cost decreases or increases • Adaptive learning rate: multiply the learning rate for a parameter by 1.2 if the partial derivative of the loss, with respect to that parameter, remains the same sign in successive steps and multiply it by 0.7 otherwise [87] • Number of hidden nodes for MLP: 3
Perceptron without location as additional feature	45.85	16392	
Perceptron with location as additional feature	47.90	17291	
Weighted perceptron	43.35	24786	
Weighted perceptron using only the top 30% of training samples	**54.95**	**12472**	
Weighted perceptron using only the top 10% of training samples	36.66	3397	
Weighted perceptron using only the top 1% of training samples	22.57	365	
MLP with location as the only feature	45.41	35219	
MLP without location as additional feature	44.87	37224	
MLP with location as additional feature	48.26	41811	
Weighted MLP	**57.72**	**127349**	
Weighted MLP using only the top 30% of training samples	51.03	34972	
Weighted MLP using only the top 10% of training samples	35.95	13613	
Weighted MLP using only the top 1% of training samples	23.46	1483	

One apparent irregularity in the above table is that for the classifier that simply assigns the class with the largest collective spatial weight among the training samples to the irresponsive sample, the performance time does not reduce when training samples with small weights are excluded. It is because if all training samples participate in finding the class with the largest collective weight, there is no need to sort the weights of training samples but if one decides to exclude the training samples with very small weights, the weight vector needs to be sorted. The

sorting function which is invoked only if a subset of training samples needs to be applied, has a greater time complexity than the function that sums up the training samples' weights.

Non-parametric classifiers, in our case the Bayesian classifier with Parzen windows, need to be trained separately for each irresponsive sample, regardless of the classifier being non-weighted or weighted. For parametric classifiers, the weighted version needs to be trained separately for each irresponsive sample but the non-weighted version needs to be trained once for all irresponsive samples. In other words, the weighted classifiers need to be trained as many times as the number of irresponsive samples because the spatial weights for training samples depend upon the irresponsive sample's location. However, the leave-one-out approach for evaluation eliminates this effect, because each time there is only one test sample and consequently both weighted and non-weighted classifiers are trained equal number of times.

The weighted and non-weighted Bayesian classifiers have almost identical performance times with only 4% difference. The weighted SVM takes 65 times longer to be trained than non-weighted SVM in Table 5.4. The reason is that finding the weighted SVM classifier includes a loop where in each iteration the original training samples are shifted with respect to the non-weighted SVM classifier proportional to their weight (Equation (3.82)) and the non-weighted SVM classifier is recalculated for the shifted training samples. The weighted MLP takes 3 times longer than non-weighted MLP. Upon detailed analysis of the results it became clear that the longer training time for weighted MLP is due to the presence of training samples' weights in computing the correction term (Equation (3.111)). Similarly, the 1.4 times longer performance time of weighted Perceptron in comparison with non-weighted Perceptron is attributed to the presence of training samples' weights in computing the correction term (Equation (3.20). The 2.8 times longer performance time of the weighted LS than non-weighted LS is attributed to the

presence of training samples' weights in Equation (3.15). The weighted decision tree halves the performance time in comparison with non-weighted decision tree which is because the former is developed in a feature space with one less dimension (i.e., location) than the latter and dimensionality plays a crucial role in both training and height of decision trees.

Among the same versions of seven different classifiers in Table 5.4, the collective weight is the fastest approach, followed by LS, Bayesian classifier, decision tree, perceptron, SVM, and MLP in most cases. Weighted SVM is the slowest approach. Cross-validation of the weighted SVM, trained with only 30% of training samples, took 14 days. Cross-validation of a weighted SVM, trained with all training samples, takes months and the results are not reported in Table 5.4.

Despite its simplicity and short time performance, the classifier which assigns the irresponsive sample to the class with the largest collective weight among the top 1% of training samples achieves an accuracy which is surpassed only slightly (1.61%) by decision tree with location as the only feature. Considering that this technique only needs location (to calculate the spatial weights), its high accuracy underscores the efficiency of spatial weights in depicting the spatial autocorrelation between training samples and the irresponsive sample. It is worth noting that this classifier owes its high accuracy, in part, to the almost uniform and dense distribution of samples across space, shown in Figure 5.3, which manifests itself in a large value for partial sill (c_1 =0.82) in the spatial semivariogram (Figure 5.4). The absence of such density and uniformity in distribution of samples across space will result in a small value for partial sill (c_1) in the spatial semivariogram (Figure 4.2), meaning no or weak autocorrelation. This, in turn, would make the geographical proximity of much less help in classifying irresponsive samples and could

significantly degrade the accuracy of this classifier. In such circumstances, non-spatial features can help improve the classification accuracy.

Highly non-linear classifiers (decision tree, collective weight, and Bayes) with location as the only input feature achieve the highest accuracies, as shown in Table 5.4. This implies two facts in classifying land covers: (a) the distribution of classes in space are very non-linear and (b) location plays the most important role in identifying the land cover. The first fact explains why the classification accuracy drops for linear and slightly non-linear classifiers even when location is their only input feature. The second fact explains the low accuracy of any classifier which ignores the location altogether. None of these facts explain why the weighted version of these highly non-linear classifiers (decision tree and Bayesian) cannot outperform their non-weighted version which uses location as the only input feature. This can be explained by the low distinctive power (noisy behavior) of non-spatial features in identifying the classes, which is proven by the very low accuracy of all non-weighted classifiers with only non-spatial input features (second row in each group in Table 5.4). The lower sensitivity of perceptron to noisy training samples in comparison with LS comes as a major advantage in such circumstances, creating a considerable difference between their classification accuracies when there are non-spatial features among inputs. The low distinctive power of non-spatial features in identifying the land covers is also accountable for the decreasing accuracy of weighted classifiers as the training samples are shrunk to those with largest spatial weights.

Rows with a bold font in the above table show the settings leading to the best accuracy for each of the seven classification techniques (collective weight, LS, Bayes, decision tree, SVM, perceptron, and MLP). The accuracy of weighted classifiers is always higher than the accuracy of their non-weighted counterpart trained using all features including location, with two

exceptions: decision tree and perceptron. In case of perceptron, using only the top 30% of training samples with largest weights puts the weighted perceptron above the non-weighted perceptron, in terms of accuracy.

Table 5.5 ranks the same version of the seven different classifiers (collective weight, LS, Bayes, decision tree, SVM, perceptron, and MLP) based on their accuracy in Table 5.4.

Table 5.5. The same version of different classifiers ranked based on their accuracy for land cover data.

Classifier with location as the only feature	Classifier without location as additional feature	Classifier with location as additional feature	Weighted classifier	Weighted classifier using only the top 30% of training samples	Weighted classifier using only the top 10% of training samples	Weighted classifier using only the top 1% of training samples
Decision tree	Perceptron	Decision tree	MLP	Perceptron	Collective weight	Collective weight
Bayes	MLP	MLP	LS	MLP	Perceptron	Bayes
MLP	Decision tree	Perceptron	Decision tree	Decision tree	MLP	SVM
Perceptron	SVM	SVM	Perceptron	LS	Decision tree	MLP
LS	LS	LS	Collective weight	SVM	LS	Decision tree
SVM	Bayes	Bayes	Bayes	Collective weight	SVM	Perceptron
				Bayes	Bayes	LS

Accuracy ↑

5.3 CLASSIFICATION OF SIMULATED DATA

We generated 500 random samples from each of the two classes A and B. Samples are pulled from two five dimensional normal probability distribution functions with the following mean vectors (μ_A and μ_B) and symmetric positive-definite covariance matrixes (Σ_A and Σ_B):

$$\mu_A = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \quad \Sigma_A = \begin{bmatrix} 10 & 0 & 1 & 1 & 2 \\ 0 & 10 & 3 & 4 & 1 \\ 1 & 3 & 10 & 2 & 3 \\ 1 & 4 & 2 & 10 & 4 \\ 2 & 1 & 3 & 4 & 10 \end{bmatrix} \qquad \mu_B = \begin{bmatrix} 101 \\ 101 \\ 101 \\ 101 \\ 101 \end{bmatrix} \quad \Sigma_B = \begin{bmatrix} 10 & 1 & 2 & 1 & 4 \\ 1 & 11 & 4 & 1 & 3 \\ 2 & 4 & 10 & 2 & 2 \\ 1 & 1 & 2 & 11 & 4 \\ 4 & 3 & 2 & 4 & 10 \end{bmatrix}$$

Location is defined for samples to satisfy a spherical semivariogram with the following characteristics: c_1=0.3, c_0=0, and r=25. Time is defined for samples to satisfy a single-frequency periodogram with the following characteristics: amplitude=0.05, period=20, and c_0=0. A white noise was also added to location and time of training samples, resulting in spatial and temporal semivariograms with the following characteristics: c_1=0.25, c_0=0.04, r=22.5 for the spatial semivariogram and β_1=-0.0142, β_2=-0.0394, amplitude=0.0419, period=22.5, and c_0=0.04 for the temporal semivariogram.

Of all the above samples, 80% are chosen randomly to serve as training samples and the remaining 20% are used to evaluate the performance of weighted Bayesian classifier, weighted LS, weighted perceptron, weighted SVM, weighted decision tree, and weighted MLP. Table 5.6 shows the overall accuracy of different weighted classifiers.

For comparison purposes, we also used only the top 30%, 10%, and 1% of training samples with largest spatial-temporal weights for training. The resulted accuracies are reported in Table 5.6. This table also reports the accuracy of non-weighted classifiers trained with non-spatial and non-temporal features, non-weighted classifiers trained with all features including location and time, non-weighted classifiers trained with location and time as the only features, and finally picking the class with the largest collective spatial-temporal weight among training samples.

Table 5.6. The accuracy of different classification techniques for the simulated data.

Technique	Overall Acc.	Settings
Choosing the class with largest collective weight among training samples	63.5	• Spatial-temporal weights of training samples are used as weights
Choosing the class with largest collective weight among the top 30% of training samples	41	
Choosing the class with largest collective weight among the top 10% of training samples	**96**	
Choosing the class with largest collective weight among the top 1% of training samples	51	
LS with location and time as the only features	57.0	• Using the one-against-one scheme since there are more than two classes
LS without location and time as additional features	56	
LS with location and time as additional features	56.5	
Weighted LS	60	
Weighted LS using only the top 30% of training samples	51.5	
Weighted LS using only the top 10% of training samples	**97**	
Weighted LS using only the top 1% of training samples	48	
Bayesian with location and time as the only features	53.5	• Non-parametric Parzen windows with Gaussian kernel (Equation (3.4)) where σ=5 • Priors are based on class frequencies
Bayesian without location and time as additional features	46.5	
Bayesian with location and time as additional features	57	
Weighted Bayesian	48	
Weighted Bayesian using only the top 30% of training samples	60.5	
Weighted Bayesian using only the top 10% of training samples	**70**	
Weighted Bayesian using only the top 1% of training samples	51.0	
Decision tree with location and time as the only features	45	• A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than 2.0.
Decision tree without location and time as additional features	45	
Decision tree with location and time as additional features	45	
Weighted decision tree	63.5	
Weighted decision tree using only the top 30% of training samples	41	
Weighted decision tree using only the top 10% of training samples	**96**	
Weighted decision tree using only the top 1% of training samples	51	
SVM with location and time as the only features	49.5	• Smoothing parameter (C)=10 • Using the one-against-one scheme since there are more than two classes
SVM without location and time as additional features	45	
SVM with location and time as additional features	45	
Weighted SVM	45	

Weighted SVM using only the top 30% of training samples	50.5	
Weighted SVM using only the top 10% of training samples	**62.5**	
Weighted SVM using only the top 1% of training samples	50	
Perceptron with location and time as the only features	50	• Maximum number of iterations: 1000 for perceptron and 5000 for MLP • Not updating the weights after those iterations resulting in an increase in the total cost • Multiply all learning rates by 1.1 or 0.8 after each step based on whether the total cost decreases or increases • Adaptive learning rate: multiply the learning rate for a parameter by 1.2 if the partial derivative of the loss, with respect to that parameter, remains the same sign in successive steps and multiply it by 0.7 otherwise [87] • Cost function for MLP: Sum of squared errors • Logistic activation function for MLP • Number of hidden nodes for MLP: 5
Perceptron without location and time as additional features	43.5	
Perceptron with location and time as additional features	50	
Weighted perceptron	48	
Weighted perceptron using only the top 30% of training samples	43.5	
Weighted perceptron using only the top 10% of training samples	**92**	
Weighted perceptron using only the top 1% of training samples	50.5	
MLP with location and time as the only features	50.5	
MLP without location and time as additional features	45	
MLP with location and time as additional features	44.5	
Weighted MLP	65	
Weighted MLP using only the top 30% of training samples	41	
Weighted MLP using only the top 10% of training samples	**96.5**	
Weighted MLP using only the top 1% of training samples	51	

Despite its simplicity, the classifier which assigns the irresponsive sample to the class with the largest collective weight among the top 10% of training samples achieves an accuracy which is surpassed only slightly (1%) by weighted least squares and weighted MLP using only the top 10% of training samples. Considering that this technique only needs location and time (to calculate the spatial-temporal weights), its high accuracy underscores the effectiveness of spatial-temporal weights in depicting the spatial-temporal autocorrelation between training samples and the irresponsive sample. It is worth noting that this classifier owes its high accuracy, in part, to the almost uniform and dense distribution of samples across space and time.

Rows with a bold font in Table 5.6 show the settings leading to the best accuracy for each of the seven classification techniques (collective weight, LS, Bayes, decision tree, SVM, perceptron, and MLP). Ignoring location and time altogether resulted in a lower accuracy in

comparison with considering them as the only or additional features. However, this difference in accuracy is small (0~10%). On the other hand, in all cases, the weighted classifier using only the top 10% of training samples achieves the highest accuracy. The accuracy improvement is 13% to 51%. This shows the superiority of the weighted machine learning over other alternatives for considering the location and time to improve the accuracy. This also underscores the important role both spatial-temporal and other features play in improving the prediction accuracy.

Between the two approaches, i.e., the non-weighted classifier trained using all features including location and time and its weighted counterpart, the latter achieves a slightly better accuracy (except for Bayesian and Perceptron). However, when only 10% of training samples with largest spatial-temporal weights are applied to train the weighted classifier, the accuracy is dramatically improved. This also uncovers the fact that training samples with very small spatial-temporal weights are not much constructive in training the classifier. This is in line with our expectations that weighted classification captures the spatial-temporal autocorrelation best and excluding training samples with very small spatial-temporal weights not only reduces the time performance but also improves the accuracy.

Table 5.7 ranks the same version of the seven different classifiers (collective weight, LS, Bayes, decision tree, SVM, perceptron, and MLP) based on their accuracy in Table 5.6.

Table 5.7. The same version of different classifiers ranked based on their accuracy for simulated data.

Classifier with location and time as the only features	Classifier without location and time as additional features	Classifier with location and time as additional features	Weighted classifier	Weighted classifier using only the top 30% of training samples	Weighted classifier using only the top 10% of training samples	Weighted classifier using only the top 1% of training samples

LS	LS	Bayes	MLP	Bayes	LS	MLP
Bayes	Bayes	LS	Decision tree	LS	MLP	Bayes
MLP	MLP	Perceptron	Collective weight	SVM	Decision tree	Decision tree
Perceptron	SVM	SVM	LS	Perceptron	Collective weight	Collective weight
SVM	Decision tree	Decision tree	Bayes	MLP	Perceptron	Perceptron
Decision tree	Perceptron	MLP	Perceptron	Decision tree	Bayes	SVM
			SVM	Collective weight	SVM	LS

Accuracy

5.4 CONCLUSIONS AND FUTURE DIRECTIONS

The question posed in this dissertation, how the recorded location and time for training samples should contribute to the training and testing process, can be answered more precisely now. We compared three general approaches: (a) ignoring location and time, (b) considering location and time as input features, and (c) using the spatial-temporal autocorrelation between each training sample and the irresponsive sample as that training sample's weight in weighted machine learning techniques. The third approach resulted in a better prediction accuracy since it captures the autocorrelation in spatial-temporal phenomena more properly. However, because the spatial-temporal weight of training samples depends on the irresponsive sample's location and time, the machine needs to be trained separately for each irresponsive sample. We showed that using only a subset of training samples with largest spatial-temporal weights is an effective way to mitigate the training time without compromising the prediction accuracy. Applying different feature selection/generation methods and investigating their effect on the prediction accuracy is a future research direction.

6.0 CONCLUSIONS AND FUTURE DIRECTIONS

Figure 6.1. Shows the overall scheme of the proposed weighted machine learning for spatial-temporal data. It starts with calculating the spatial-temporal weight for training samples and ends with using them to bias the predictor in favor of training samples with larger weights. Spatial-temporal weights are visualized using color saturation. Darker samples have larger spatial-temporal weights. The red cross is the irresponsive sample and plays the central role in determining the spatial-temporal weight for training samples. Training samples are weighted based on their spatial and temporal autocorrelation with the irresponsive sample. In the spatial-temporal domain, the irresponsive sample is at the origin of the coordinate system and spheres delineate the training samples with larger spatial-temporal weights. The predictor is trained to be more concerned about the correct prediction of training samples with larger spatial-temporal weights.

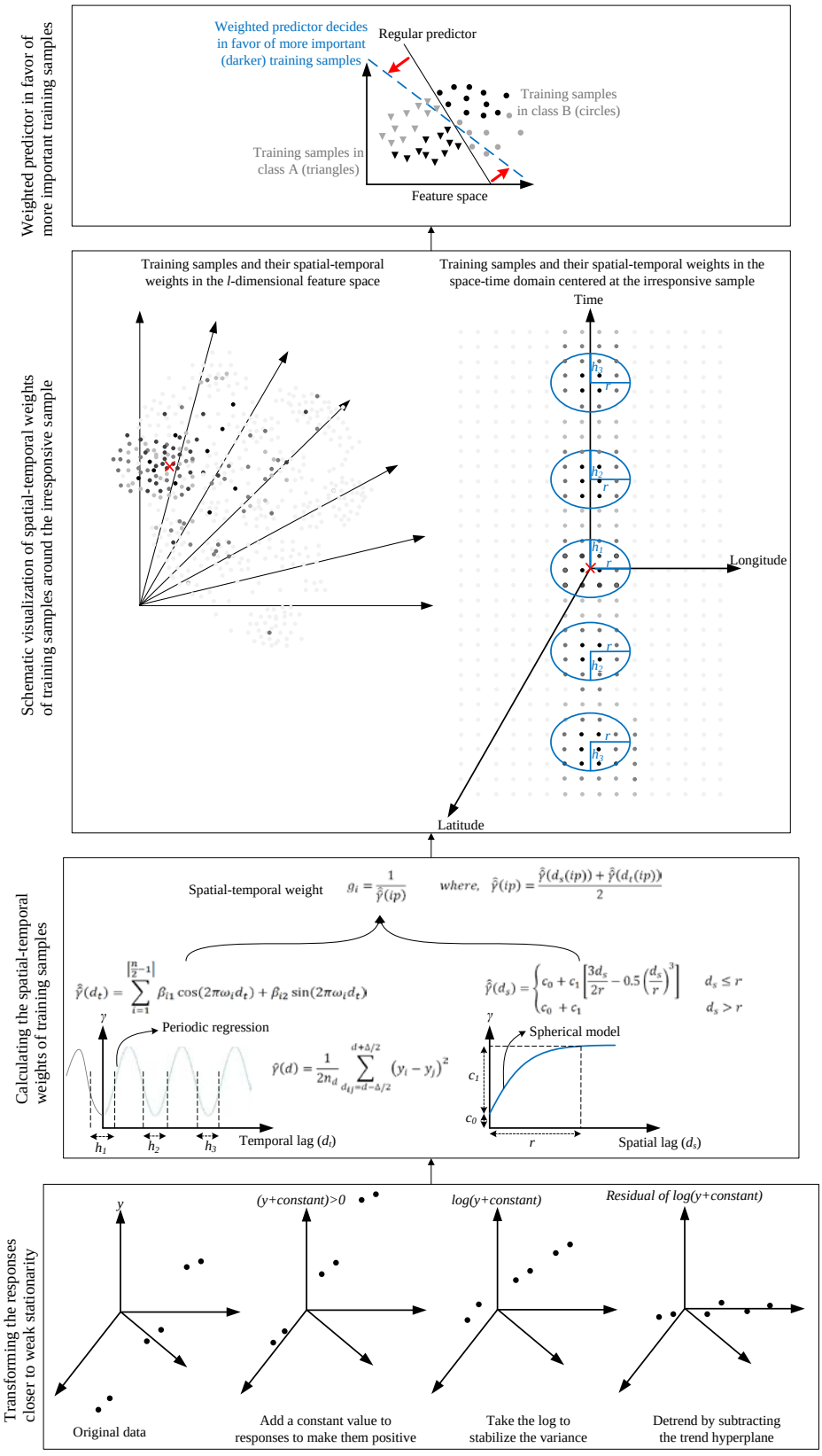


Figure 6.1. Weighted machine learning for spatial-temporal data.

6.1 CONCLUSIONS

The weighted machine learning techniques developed in Chapter 3, not only provide users with the opportunity to give different weights to training samples, but also can be embedded into other algorithms such as AdaBoost [66; 67], where there is a hierarchy of classifiers, each requiring to be trained using a different weighting over training samples. These weights are used to adjust the classifier/regressor in favor of more importance samples, and thereby giving a higher significance to more important samples. It is worth noting that the weighted linear and nonlinear classifiers change the division of the feature space only around the border (the most uncertain area) and areas far from the border are less likely to change their label. The weighted SVM classifier showed the least difference with its non-weighted counterpart when the training samples' weights are not much variant. The reason for this is that SVM classifier is designed only based on support vectors not all training samples. If the weights of training samples are not much different, their relocation based on their weights might not be large enough to change the selection of support vectors. In other words, the weighted SVM classifier would be different than its non-weighted version only if relocating training samples based on weights would result in a rearrangement of training samples significant enough to change the selection of support vectors. In other words, the weighted SVM has the highest stability with respect to weight changes in a small subset of training samples. The weighted MLP also takes the training samples' weights into account through smallest adjustments in its nonlinear border. However, the MLP's behavior is highly dependent on the network's size. In other words, a larger number of hidden nodes will result in a more significant difference between the weighted and non-weighted MLP. On the other hand, the weighted decision tree and weighted Bayesian classifiers showed the most dramatic changes in how the feature space is divided between the two classes in comparison with

their non-weighted counterparts. The reason is that these two models are highly local, especially the non-parametric Bayesian classifier. Therefore, even small changes in the training samples' weights would result in a different classification of the feature space. The weighted LS and perceptron showed slight and similar changes in how they divide the feature space between the two classes in comparison with their non-weighted counterparts. The similar behavior is because both models minimize the sum of square errors, although in different ways. The difference between weighted and non-weighted versions being slight in these two cases originates from their linear nature and consequently their rather restricted flexibility in modifying their shape.

The logic behind deciding in favor of spatial-temporally more important samples is that the output of the irresponsive sample is more likely to be similar to the output of training samples with larger spatial-temporal weights. This is justified because, in Chapter 4, we calculate the spatial-temporal weights based on similarity among the responses of training samples across space and time. Therefore, we expect the spatial-temporally weighted predictor to achieve a better out-of-sample (test) accuracy than its non-weighted counterpart, despite it is not difficult to see that the non-weighted predictor achieves a better in-sample (training) accuracy (simply because it is equally fair to all training samples).

We posed the following question in this dissertation: how the recorded location and time for training samples should contribute to the training and testing process. We compared three general approaches: (a) ignoring location and time, (b) considering location and time as input features, and (c) using the spatial-temporal autocorrelation between each training sample and the irresponsive sample as that training sample's weight in weighted machine learning techniques. The third approach resulted in a better prediction accuracy in two real-world applications since it captures the autocorrelation in spatial-temporal phenomena more properly. However, because the

spatial-temporal weight of training samples depends on the irresponsive sample's location and time, the machine needs to be trained separately for each irresponsive sample. We showed that using only a subset of training samples with largest spatial-temporal weights is an effective way to mitigate the training time without compromising the prediction accuracy.

6.2 FUTURE DIRECTIONS

Our approach of calculating spatial weights for training samples assumes that the underlying phenomenon is isotropic by considering only the distance between pairs of observations and ignoring the direction of the vector connecting them. A future research venue is to increase the accuracy of spatial weights by taking into account the anisotropy or directional effects in the spatial variation of responses. This can be done by developing two (or more) spatial semivariograms, each modeling the spatial similarity in either north-south or east-west direction. To calculate the semivariance in a specific direction, only pairs of samples aligned in that direction (at least approximately) are used. Therefore, the direction of the vector connecting the irresponsive sample to each training sample determines which spatial semivariogram must be used to calculate that training sample's spatial weight. This way training samples aligned in a specific direction with respect to the irresponsive sample might gain a higher spatial weight.

We developed spatial and temporal semivariograms separately because the former is best modeled with a spherical model and the latter with a periodogram. Another future research venue is to find a way to develop a single spatial-temporal semivariogram. Applying different feature selection/generation methods and investigating their effect on the prediction accuracy is also a future research direction.

BIBLIOGRAPHY

[1] Deng, L., & Yu, D. (2014). Deep learning: Methods and applications. *Foundations and Trends in Signal Processing , 7* (3-4), 197-387.

[2] O'Sullivan, D., & Unwin, D. (2010). *Geographic information analysis.* Hoboken, New Jersey: Wiley.

[3] Cressie, N. (1993). *Statistics for spatial data* (Revised ed.). New York: Wiley.

[4] Shekhar, S., & Chawla, S. (2003). Introduction to spatial data mining. In *Spatial databases: a tour* (pp. 182-226). Upper Saddle River, NJ: Prentice Hall.

[5] Isaaks, E. H., & Srivastava, R. M. (1989). *An introduction to applied geostatistics.* Oxford: Oxford University Press.

[6] Santibanez, S., Kloft, M., & Lakes, T. (2015). Performance analysis of machine learning algorithms for regression of spatial variables. A case study in the real estate industry. *In Proceedings of the GeoComputation Conference.* Dallas, Texas.

[7] Santibanez, S., Lakes, T., & Kloft, M. (2015). Performance analysis of some machine learning algorithms for regression under varying spatial autocorrelation. *AGILE.* Lisbon.

[8] Anselin, L. (1988). *Spatial econometrics: Methods and models.* Dordrecht, The Netherlands: Kluwer.

[9] Fotheringham, A. S., Brunsdon, C., & Charlton, M. (2002). *Geographically weighted regression.* Chichester, England: Wiley.

[10] Fotheringham, A. S., Brunsdon, C., & Charlton, M. (2000). *Quantitative geography: Perspectives on spatial data analysis.* London: Sage.

[11] Shekhar, S., Zhang, P., & Huang, Y. (2010). Spatial data mining. In *Data mining and knowledge discovery handbook* (2nd ed., pp. 837-854). New York: Springer.

[12] Tobler, W. R. (1970). A computer movie simulating urban growth in the Detroit region. *Economic Geography , 46*, 234-240.

[13] Shekhar, S., Zhang, P., Huang, Y., & Vatsavai, R. R. (2003). Trends in spatial data mining. In H. Kargupta, & A. Joshi (Eds.), *Data mining: Next generation challenges and future directions* (pp. 357-380). AAAI/MIT Press.

[14] Gould, P. R. (1970). Is statistix inferens the geographical name for a wild goose? *Economic geography , 46*, 439-448.

[15] Case, K. E., & Shiller, R. J. (1989). The efficiency of the market for single family homes. *American Economic Review , 79* (1), 125-137.

[16] Pace, R. K., Barry, R., Clapp, J. M., & Rodriquez, M. (1998). Spatio-Temporal autoregressive models of neighborhood effects. *Journal of Real Estate Finance and Economics , 17* (1), 15-33.

[17] Mertens, B., & Lambin, E. F. (2000). Land-cover-change trajectories in southern Cameroon. *Annals of the Association of American Geographers , 90* (3), 467-494.

[18] Lucas, R. M., Honzak, M., Foody, G. M., Curran, P. J., & Corves, C. (1993). Characterizing tropical secondary forests using multi-temporal Landsat sensor imagery. *International Journal of Remote Sensing , 14* (16), 3061-3067.

[19] Alves, D. S., & Skole, D. L. (1996). Characterizing land cover dynamics using multi-temporal imagery. *International Journal of Remote Sensing , 17* (4), 835-839.

[20] Tucker, C. J., Dregne, H. E., & Newcomb, W. W. (1991). Expansion and contraction of the Sahara Desert from 1980 to 1990. *Science , 253*, 299-301.

[21] Gokaraju, B., Durbha, S. S., King, R. L., & Younan, N. H. (2011). A machine learning based spatio-temporal data mining approach for detection of harmful algal blooms in the Gulf of Mexico. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing , 4* (3), 710-720.

[22] Franklin, J. (1995). Predictive vegetation mapping: geographic modelling of biospatial patterns in relation to environmental gradients. *Progress in Physical Geography , 19* (4), 474-499.

[23] Shekhar, S., Evans, M. R., Kang, J. M., & Mohan, P. (2011). Identifying patterns in spatial information: A survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery , 1* (3), 193-214.

[24] Kanevski, M., Parkin, R., Pozdnukhov, A., Timonin, V., Maignan, M., Demyanov, V., et al. (2004). Environmental data mining and modeling based on machine learning algorithms and geostatistics. *Environmental Modelling & Software , 19* (9), 845-855.

[25] Gilardi, N., & Bengio, S. (2003). Comparison of four machine learning algorithms for spatial data analysis. In G. Dubois, J. Malczewski, & M. D. Cort, *Mapping radioactivity in the environment: spatial interpolation comparison 97* (pp. 222-237). Luxembourg: Office for Official Publications of the European Communities.

[26] Can, A. (1990). The measurement of neighborhood dynamics in urban house prices. *Economic Geography , 66* (3), 254-272.

[27] Can, A. (1992). Specification and estimation of hedonic housing price models. *Regional Science and Urban Economics , 22* (3), 453-474.

[28] Dubin, R. A. (1992). Spatial autocorrelation and neighborhood quality. *Regional Science and Urban Economics , 22* (3), 433-452.

[29] Dubin, R. A. (1998). Predicting house prices using multiple listings data. *Journal of Real Estate Finance and Economics , 17* (1), 35-59.

[30] Kisilevich, S., Mansmann, F., Nanni, M., & Rinzivillo, S. (2010). Spatio-temporal clustering. In *Data mining and knowledge discovery handbook* (2nd ed., pp. 855-874). New York: Springer.

[31] Sahu, S. K., & Mardia, K. V. (2005). Recent trends in modeling spatio-temporal data. *In Proceedings of the special meeting on Statistics and Environment* (pp. 69-83). Messina, Italy: The Societa Italiana di Statistica; Universita Di Messina.

[32] Basu, S., & Thibodeau, T. G. (1998). Analysis of spatial autocorrelation in home prices. *Journal of Real Estate Finance and Economics , 16* (1), 61-85.

[33] Gilardi, N., & Bengio, S. (2000). Local machine learning models for spatial data analysis. *Journal of Geographic Information and Decision Analysis , 4* (1), 11-28.

[34] Bellman, R. E. (1961). *Adaptive control processes.* Princeton: Princeton University Press.

[35] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction* (2nd ed.). New York: Springer.

[36] Evangelista, P. F., Embrechts, M. J., & Szymanski, B. K. (2006). Taming the curse of dimensionality in kernels and novelty detection. In *Applied soft computing technologies: the challenge of complexity* (pp. 425-438). Berlin: Springer.

[37] Devroye, L., Györfi, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition.* Springer.

[38] Theodoridis, S., & Koutroumbas, K. (2009). *Pattern Recognition* (4th ed.). Elsevier.

[39] Leon, S. J. (2010). *Linear algebra with applications* (8th ed.). Upper Saddle River, NJ: Prentice Hall.

[40] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review , 65* (6), 386-408.

[41] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning , 20* (3), 273-297.

[42] Vapnik, V. N. (1995). *The nature of statistical learning theory.* New York: Springer.

[43] Vapnik, V. N. (1998). *Statistical learning theory.* New York: Wiley.

[44] Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees.* CRC press.

[45] Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Cambridge, MA: Ph.D. Thesis, Harvard University.

[46] Bishop, C. M. (1995). *Neural networks for pattern recognition.* Oxford University Press.

[47] Haykin, S. S. (1999). *Neural networks: a comprehensive foundation* (2nd ed.). Prentice Hall.

[48] Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *In Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (pp. 144-152). ACM.

[49] Lewis, D. D., & Gale, W. A. (1994). A sequential algorithm for training text classifiers. *In Proceedings of the 17th International ACM-SIGIR Conference on Research and Development in Information Retrieval* (pp. 3-12). Springer-Verlag.

[50] Schohn, G., & Cohn, D. (2000). Less is more: Active learning with support vector machines. *In Proceedings of the 17th International Conference on Machine Learning* (pp. 839-846). Morgan Kaufmann.

[51] Yuksel, S. E., Wilson, J. N., & Gader, P. D. (2012). Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems , 23* (8), 1177-1193.

[52] Li, J., Heap, A. D., Potter, A., & Daniell, J. J. (2011). Application of machine learning methods to spatial interpolation of environmental variables. *Environmental Modelling & Software , 26* (12), 1647-1659.

[53] Breiman, L. (2001). Random forests. *Machine Learning , 45* (1), 5-32.

[54] Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing , 14* (3), 199-222.

[55] Kanevski, M., Timonin, V., & Pozdnoukhov, A. (2011). Automatic Mapping and Classification of Spatial Environmental Data. In *Geocomputation, Sustainability and Environmental Planning* (pp. 205-223). Springer.

[56] Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics , 33* (3), 1065-1076.

[57] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation , 3* (1), 79-87.

[58] Cracknell, M. J., & Reading, A. M. (2014). Geological mapping using remote sensing data: Acomparison of five machine learning algorithms, their response to variations in the spatial distribution of training data and the use of explicit spatial information. *Computers & Geosciences , 63*, 22-33.

[59] Gahegan, M., German, G., & West, G. (1999). Improving neural network performance on the classification of complex geographic datasets. *Journal of Geographical Systems , 1* (1), 3-22.

[60] He, T., Sun, Y.-J., Xu, J.-D., Wang, X.-J., & Hu, C.-R. (2014). Enhanced land use/cover classification using support vector machines and fuzzy k-means clustering algorithms. *Journal of Applied Remote Sensing , 8* (1).

[61] DeFries, R. S., & Chan, J. C.-W. (2000). Multiple criteria for evaluating machine learning algorithms for land cover classification from satellite data. *Remote Sensing of Environment , 74* (3), 503-515.

[62] Barrett, B., Nitze, I., Green, S., & Cawkwell, F. (2014). Assessment of multi-temporal, multi-sensor radar and ancillary spatial data for grasslands monitoring in Ireland using machine learning approaches. *Remote Sensing of Environment , 152*, 109-124.

[63] Rizzo, D., Martin, L., & Wohlfahrt, J. (2014). Miscanthus spatial location as seen by farmers: A machine learning approach to model real criteria. *Biomass and Bioenergy , 66*, 348-363.

[64] Ballabio, C., & Sterlacchini, S. (2012). Support vector machines for landslide susceptibility mapping: the Staffora River Basin case study, Italy. *Mathematical Geosciences , 44* (1), 47-70.

[65] Dube, T., Mutanga, O., Elhadi, A., & Ismail, R. (2014). Intra-and-inter species biomass prediction in a plantation forest: testing the utility of high spatial resolution spaceborne multispectral rapideye sensor and advanced machine learning algorithms. *Sensors , 14* (8), 15348-15370.

[66] Quinlan, J. R. (1996). Bagging, boosting, and C4.5. *In Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 725-730). Portland, Oregon: AAAI Press.

[67] Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *The Annals of Statistics , 28* (2), 337-407.

[68] Hardle, W. (1990). *Applied nonparametric regression.* Cambridge, UK: Cambridge University Press.

[69] Fan, J., & Gijbels, I. (1996). Local polynomial modelling and its applications: monographs on statistics and applied probability. London: CRC Press.

[70] Wasserman, L. (2006). *All of nonparametric statistics.* Berlin: Springer.

[71] Minsky, M. L., & Papert, S. (1988). *Perceptrons, expanded edition.* MA: MIT Press.

[72] Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation.* Addison-Wesley.

[73] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press.

[74] Bazaraa, M. S., Sherali, H. D., & Shetty, C. M. (2006). *Nonlinear programming: theory and algorithms* (3rd ed.). New Jersey: Wiley.

[75] Bertsekas, D. P. (1999). *Nonlinear programming* (2nd ed.). Belmont, Massachusetts: Athena Scientific.

[76] Fletcher, R. (1987). *Practical methods of optimization* (2nd ed.). Chichester, West Sussex, England: Wiley.

[77] Nash, S. G., & Sofer, A. (1996). *Linear and nonlinear programming.* New York: McGraw-Hill.

[78] Ripley, B. D. (1994). Neural networks and related methods for classification. *Journal of the Royal Statistical Society, Series B , 56* (3), 409-456.

[79] Witten, I., Frank, E., Hall, M., & Pal, C. (2016). *Data Mining: Practical machine learning tools and techniques* (4th ed.). Morgan Kaufmann.

[80] Richard, M. D., & Lippmann, R. P. (1991). Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural Computation , 3* (4), 461-483.

[81] Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers , 14* (3), 326-334.

[82] Mercer, J. (1909). Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A , 209*, 415-446.

[83] Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis.* Cambridge university press.

[84] Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press.

[85] Courant, R., & Hilbert, D. (1953). *Methods of Mathematical Physics* (Vol. I). New York: Interscience.

[86] Wolberg, W. H. (1992). *Breast Cancer Wisconsin Data Set*. Retrieved 2017, from http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29

[87] Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks , 1* (4), 295-307.

[88] SAS Institute Inc. (2010). *SAS/STAT® 9.22 User's Guide.* Cary, NC: SAS Institute Inc.

[89] Esri. (2012, 09 28). *ArcGIS Help Library*. Retrieved from ArcGIS Resource Center: http://help.arcgis.com/En/Arcgisdesktop/10.0/Help/index.html#//0031000000mq000000

[90] Bohling, G. (2005). Introduction to geostatistics and variogram analysis. *Kansas geological survey , 2*.

[91] Shumway, R. H., & Stoffer, D. S. (2010). *Time series analysis and its applications: with R examples* (3rd ed.). New York: Springer.

[92] Stevens, S. S. (1946). On the theory of scales of measurements. *Science , 103*, 677-680.

[93] Cliff, A. D., & Ord, J. K. (1973). *Spatial autocorrelation.* London: Pion.

[94] Unwin, D. J. (1981). *Introductory spatial analysis.* London: Methuen.

[95] Chatfield, C. (2016). *The analysis of time series: An introduction* (6th ed.). CRC press.

[96] *Oceanographic and surface meteorological readings taken from a series of buoys positioned throughout the equatorial Pacific.* Pacific Marine Environmental Laboratory , National Oceanic and Atmospheric Administration , US Department of Commerce.

[97] Fitzgerald, R. W., & Lees, B. G. (1995). Spatial context and scale relationships in raster data for thematic mapping in natural systems. In T. C. Waugh, & R. Healey (Eds.), *Advances In GIS Research* (pp. 462-475). Southhampton: Taylor and Francis.

[98] Fitzgerald, R. W., & Lees, B. G. (1996). Temporal context in floristic classification. *Computers & Geosciences , 22* (9), 981-994.

[99] Huang, Z., & Lees, B. G. (2004). Combining non-parametric models for multisource predictive forest mapping. *Photogrammetric Engineering & Remote Sensing , 70* (4), 415-425.

[100] Huang, Z., & Lees, B. G. (2005). Representing and reducing error in natural-resource classification using model combination. *International Journal of Geographical Information Science , 19* (5), 603-621.

[101] Lees, B. (2006). The spatial analysis of spectral data: Extracting the neglected data. *Applied GIS , 2* (2), 14.1-14.13.

[102] Lees, B. *The Kioloa GLCTS Pathfinder Site.* http://fennerschool-associated.anu.edu.au/pathfinder/.