Regression modelling of spectroscopic data using lazy learning and deep neural networks

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Masters of Science (Research)

at

The University of Waikato

by

Huon Fraser

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2022

Abstract

Neural networks show promise in modelling infrared (IR) spectroscopic data, but many of the proposed solutions in the literature are either Multilayer Perceptrons with a single hidden layer or are adapted from successful solutions in other domains such as convolutional networks from image processing.

This thesis investigates the use of Monte Carlo search to build candidate deep neural networks from scratch. As the prediction targets are real numbers and the modelling is prone to error from outliers, lazy, locally weighted methods are also investigated.

Predictions from the network are compared to predictions made from locally weighting the features extracted from the same network. To measure progress, these results are compared to classical statistical methods commonly used to model spectroscopic data, such as partial least squares (PLS) regression.

Results suggest that deep networks trained via Monte-Carlo search outperform classical approaches. They also show that PLS dimension reduction as a preprocessing technique can improve the quality of search at little cost in predictive ability. Encouraging results using this approach are obtained on the publicly available Mangoes dataset. Finally, a streaming regression solution is investigated in which selected deep neural network models perform well. In this setting, however, locally weighted regression can result in sporadic outlier predictions, so an ensemble solution is proposed as a remedy.

Acknowledgements

I Would like to thank Professor Geoff Holmes, Dale Fletcher and Professor Albert Bifet for their supervision of this thesis.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In recent years improvements in computational power and network architecture have seen an explosion in the effectiveness and popularity of deep neural networks. Applications have been transformative for natural language processing and computer vision tasks, with state-of-the-art architectures often comprising hundreds of layers or billions of parameters.

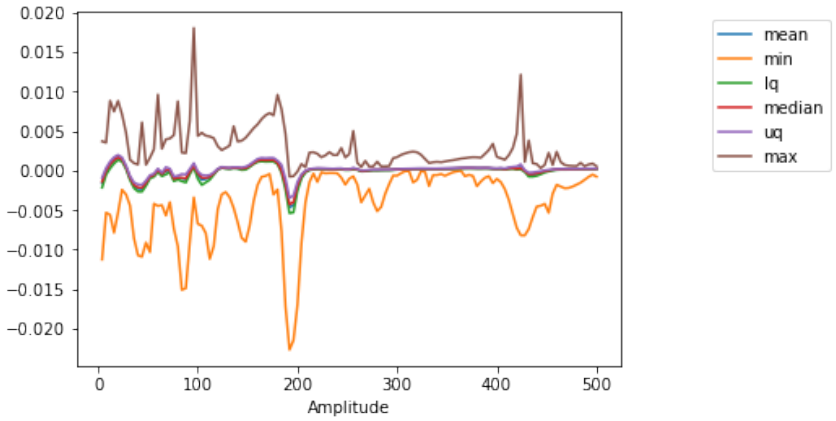
In contrast, state of the art has incrementally improved for traditional tabular learning data, with most advances being fine-grained improvements in ensemble techniques that originated in the mid-1990s.

An important factor behind the growth of deep neural network techniques has been the abundance of data, allowing larger and deeper models to reach even higher levels of generalisation. The problem domain of this thesis, infrared (IR) spectroscopy, has not yet reached the same levels of abundant data, although handheld instruments [80] may yet lead to this in the future.

These devices complement traditional destructive lab analysis. Previously labelled data can be used to calibrate devices for taking rapid and non-destructive measurements outside of a lab.

An example is testing the sweetness of kiwifruit, which accumulates something known as dry matter—the solid components of the plant—which is as-

Figure 1.1: Spectra from the D4 dataset

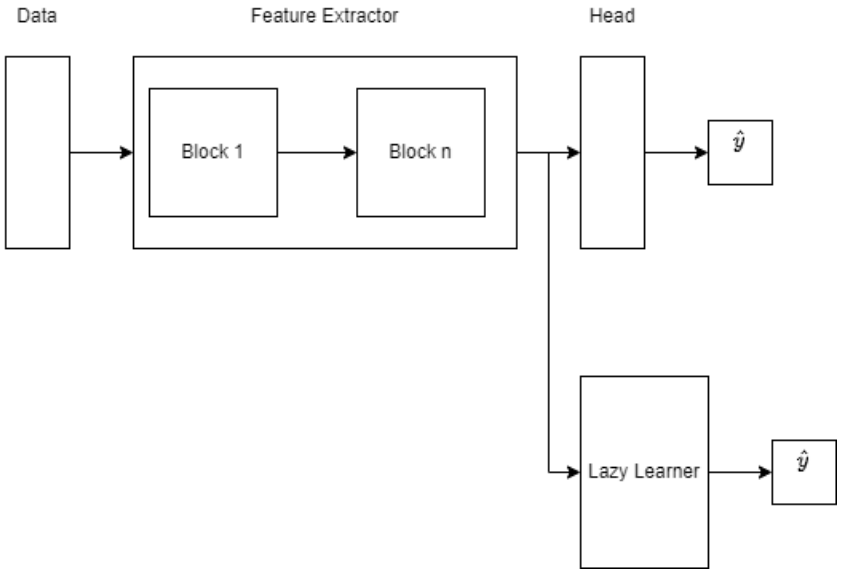


sociated with different stages of kiwifruit development. Accurately knowing the dry matter level can determine harvest time. Testing the dry matter level is required regularly towards the end of a harvest. If the conventional lab analysis of the dry matter level is recorded for samples that have also undergone infrared spectroscopy, then the spectrum can be associated with the lab analysis. Providing a requisite number of samples are analysed, a model can be built that predicts dry matter using only spectral data [39].

As was the case in the above example, the calibration for an IR spectrometer is typically a regression problem; how can the desired measurement target, such as dry matter content, be approximated by the spectrum of amplitudes at a given range of wavelengths? Spectral data typically consists of a large number of features and a high degree of multicollinearity, making it a case of tabular data where deep neural networks are, in theory, able to perform relatively well compared to classical approaches.

The state-of-the-art for this calibration problem is generally hard to determine, with domain-specific preprocessing and specific modelling techniques such as partial least squares (PLS) and locally weighted regression (LWR) being predominant in academic literature and industry. While applications of neural networks go back decades, and there has been a more recent uptake in convolutional neural networks (CNN), the neural network approaches are still not widespread.

Figure 1.2: **Proposed Architecture**



1.2 Contribution

This thesis investigates the use of deep neural networks in the spectroscopic setting. Rather than attempting to develop networks from solutions derived from other domains, such as CNN from image processing, the approach uses Monte Carlo search to find good candidate architectures for deep multilayer perceptrons (deep-MLP). Compared to shallow MLP models with a single hidden layer, deep models can better take advantage of larger datasets and result in smaller individual layers.

As most regression methods learn by interpolating the data that they are trained on, predictions may not generalise to data that differs from the training set. This problem is particularly pronounced for deep neural networks, where high parameter counts can skew these models towards overfitting.

The deep neural network models developed here are enhanced at prediction time by extracting features from the penultimate layer and inserting them into a lazy regression procedure (Figure 1.2). This procedure adds additional regularisation and non-linear dynamics to a neural network. The intended outcome is to produce more precise predictions for instances similar to seen data and more robust predictions for less similar instances.

Lazy regression can be found in IR spectroscopy applications, with locally weighted regression a standard method to add non-linear dynamics to an otherwise linear model. The lazy property ensures that this adds no extra overhead to training and that these predictors can be calculated incrementally, a property that makes the resulting models suitable for streaming.

The main contributions of the thesis are:

- Deep neural network models selected by the Monte Carlo approach outperform PLS benchmarks, especially when the search process is assisted by PLS preprocessing.

- Over a population of models, locally weighted regression marginally improves the best model while converging non-optimal models towards the best model.

- On the publicly available Mangoes dataset, the approach presented here improves on all previously published results in the literature.

- A practical streaming method using the above approach is described but requires guarding against making large erroneous predictions. Presented is a suitable guarding mechanism that makes use of ensembles.

The thesis is structured as follows. Chapter 2 presents a literature review covering classical regression techniques, neural networks, lazy learning, and infrared spectroscopy. Chapter 3 explains the methodology, including datasets, evaluation approaches and the search strategy for building deep neural networks. Results are presented in Chapter 4, first in a batch setting, second in comparison to other work on the public domain Mangoes dataset, and finally in a streaming setting. The thesis ends with conclusions in Chapter 5 and suggestions for future work.

Chapter 2

Literature Review

As previously stated, this thesis investigates regression for infrared spectral data, a supervised learning problem. The i-th instance of data is represented in the form $(\bar{X}_i, y_i)$, where $\bar{X}_i = (x_1, ...x_d)$ is a vector of d scalar features (independent variables, in this thesis typically spectra) and y_i is a scalar labelled target (dependent variable).

The goal of a supervised regression is to find a function, $f()$, that captures the relationship between the features and the target, such that predictions of the target $\hat{y}$ can be made:

$$\hat{y}_i = f(\bar{X}_i) \tag{2.1}$$

This is typically done such that the error ϵ, measured as the difference between the predicted and observed values is minimised:

$$\epsilon_i = y_i - \hat{y}_i \tag{2.2}$$

We typically refer to these values using matrix notation, with $\mathbf{y}$ a column vector of targets and $\mathbf{X}$ a matrix with rows corresponding to instances and columns corresponding to features. The term model refers to functions capturing the relationship between $\mathbf{X}$ and $\mathbf{y}$, with the internal state of a model (the variables learnt during training) known as parameters. External variables (variables decided before training) are referred to as hyperparameters.

This chapter reviews techniques found in the literature when infrared spectral data is modelled. They include classical regression techniques (Chapter

2.1), neural networks (Chapter 2.2), and locally weighted learning (Chapter 2.3). Chapter 2.4 reviews the connection of these techniques to the infrared spectroscopy domain and applications. Applications of lazy methods for deep neural networks in other domains are reviewed in Chapter 2.5.

2.1 Classical Regression Techniques

This section covers the classical methods that are investigated in Chapter 4 or that are common in IR spectroscopy applications. Starting with linear regression, a variety of techniques for building more powerful regression models are covered. A more detailed review of regression techniques for IR spectroscopy problems is given later in this chapter (Chapter 2.4).

Linear Regression

The simplest form of regression is to estimate a relationship between $\mathbf{X}$ and $\mathbf{y}$ as a linear combination of a vector of coefficients (β, also known as parameters):

$$\mathbf{y} = \mathbf{X}\beta + \epsilon \tag{2.3}$$

Linear regression models are ubiquitous as they are interpretable and are simple to fit for even small quantities of data. Simple linear regression is the case of a single independent variable, and multiple linear regression is the case of two or more independent variables. We use the term linear regression (LR) to refer to both interchangeably.

Fitting a linear regression is typically achieved analytically by ordinary least squares:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})X^T\mathbf{y} \tag{2.4}$$

The simplicity of a strictly linear regression comes with several drawbacks. A linear model will underfit any non-linear relationships present in the data. Correlations between features or multicollinearity can lead to a poorly fitting model as small perturbations in data can lead to large changes in predictions. If

two or more variables are perfectly correlated, known as perfect multicollinearity, an analytical solution for ordinary least squares cannot be found.

Regularisation and Feature Selection

As software packages offering linear regression emerged in the 1970s, so did variants offering improvements over the base formulation [26].

One common strategy is to add additional penalties to a loss function. Ridge regression [28] uses an l_2 (Euclidean distance) penalty multiplied by the parameter vector $\lambda||\beta||_2^2$ and is often used as a solution for multicollinearity. Compared to unpenalised loss functions, l_2 regularisation biases the optimisation problem toward solutions with smaller valued parameters. In ordinary least squares terms, l_2 regularisation is calculated by adding a penalty multiplied by the identity matrix (I):

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X} + I\lambda)\mathbf{X}^T\mathbf{y} \tag{2.5}$$

An alternative strategy for multicollinearity is to perform feature selection; calculate a regression for different subsets of features before selecting the best performing model. Today many software implementations, such as Weka [77], include both feature selection and l_2 regularisation in their default regression settings.

Preprocessing Techniques

In the case of linear regression, the model is invariant to the mean and variance of each variable. However, for other models, this information may affect predictions. Neural networks, for example, are particular about their inputs, with many models either assuming standardised inputs or including a standardisation layer at the start of a network [34].

Standardisation is the practice of first mean-centring the data and then scaling each feature such that its variance is 1. Many widely used models, such as principal components (discussed below), assume mean centred data or

a similar level of variance between variables. In later chapters, standardisation ensures consistency between data sets; model inputs are of the same mean and variance, independent of the dataset.

Whitening is a technique for ensuring equal variance that also acts as a change of basis, giving de-correlated or orthogonal features. In whitening, mean-centred features are transformed onto orthogonal axes. The result is a set of new features with covariance equal to the identity matrix, requiring variables to be uncorrelated and have a variance of 1.

While both standardisation and whitening scale variances, whitening also offers a solution to the multicollinearity problem.

Principal Components

Principal Components Analysis (PCA) performs a change of basis much like whitening. Compared to whitening, PCA does not scale the final features and, as such, includes additional information. Depending on the specific application, this extra information can be advantageous or disadvantageous. In Chapter 4, several experiments consider this design choice.

The full set of principal components (T) is given as the dot product of the data $\mathbf{X}$ and a weights matrix W. W is a matrix consisting of columns which are eigenvectors of $\mathbf{X}^T\mathbf{X}$:

$$T = W\mathbf{X} \tag{2.6}$$

PCA is commonly used as a dimensionality reduction technique or to rank features. Rather than taking the W matrix directly, columns from W are selected iteratively. The eigenvector (column) that maximises the average squared distance of transformed points is selected at each iteration.

Using data transformed by PCA as the input for linear regression is known as principal components regression (PCR).

Partial Least Squares

Principal components analysis is an unsupervised technique—a decomposition is performed without respect to the target value—meaning that the selected components may not have the maximum correlation to the target.

To account for the above problem Partial Least Squares (PLS) was developed as a supervised alternative to PCA by Herman Wold with the Non-linear Iterative PLS (NIPALS) algorithm [78].

PLS is a family of algorithms for modelling the cross-covariance of two blocks of data, the design, $\mathbf{X}$, and the response, $\mathbf{y}$. The blocks are decomposed into projection matrices (T and U) and orthogonal loading matrices (P and Q):

$$\begin{aligned} \mathbf{X} &= TP^T \\ \mathbf{y} &= UQ^T \end{aligned} \tag{2.7}$$

PLS is calculated through an optimisation problem to select the orthogonal spaces that maximise the covariance between T and U. Similarly to PCA, calculations are made iteratively. The axis that maximises the covariance between X and Y at each iteration is selected.

The algorithm examined in this thesis is PLS1, a case of PLS where the second block consists of a single target. PLS is often used as a regularisation technique similar to PCA. Compared to PCA, PLS is better able to untangle collinearity and still works in the case when the number of variables is greater than the number of observations [76]. As much, PLS is widely used in IR-spectroscopy applications, with these discussed in more detail later in Chapter 2.4.

Non-Linear Methods

Earlier in this chapter, we noted two problems with linear regression, non-linearity and multicollinearity. Approaches for dealing with multicollinearity are reviewed above, and we now discuss methods for dealing with non-linearity.

Non-linear regression is a broad family of techniques. In IR spectroscopy applications, many non-linear techniques regression have been found hard to train and prone to over-fitting [56]. Consequently, non-linear regressions in this field are often limited to kernel regression and neural networks. We discuss the details of neural networks in the next section (Chapter 2.2).

Least squares SVM, also known as kernel ridge regression, is a regression equivalent of the SVM learner for classification [67]. SVM is a kernel-based method, with non-linearity added through transforming the data by a kernel function. For a specific class of functions, the kernel trick allows this to be done without actually transforming the data, adding computational efficiency [57].

SVM models are flexible due to a wide range of choices in the kernel function, such as Gaussian and polynomial kernels. SVM regression is often used to add non-linearity to PCA or PLS regression [55].

2.2 Neural Networks

Artificial neural networks (ANN) consist of a computation graph, with nodes (neurons) connected by weights (synapses). In a feedforward neural network, a function is computed by propagating values from input neurons, through intermediate neurons, to output neurons. Learning is done by treating the weights as parameters of an optimisation problem on the computational graph, with weights updated according to gradient descent.

Much of the literature on neural networks is well established and can be sourced from a textbook such as Aggarwal [1]. Below we describe neural networks in enough detail to provide context for the design choices considered in this thesis.

The Perceptron

Neural networks can be traced back to the Mark 1 perceptron in 1958 [54], a physical machine that defined a linear function by connecting a layer of input nodes ($\mathbf{X}$) to a single output node ($\hat{y}$) with a physical weights matrix (W). At the output node, inputs are transformed by the weights such that the output $\hat{y} = W \cdot \mathbf{X}$, a linear regression.

An advantage of the perceptron over other linear regression techniques is that non-linearities can be added to the perceptron by applying non-linear activation functions to the outputs of nodes. For example, the classic use case of the perceptron is binary classification; which can be achieved by applying the sign function to an output value; $\hat{y} = sign(W \cdot X)$, where the sign function maps negative values to -1 and positive values to 1.

The second advantage of the perceptron lies in the method by which they are trained. Similar to other classic techniques like logistic regression, perceptrons are trained by algorithms that minimise error. However, perceptrons define this process in terms of a computation graph. Subsequent advances, taking the perceptron as a building block, have created models with more complicated computational graphs.

Multilayer Perceptrons

Stacking multiple perceptron layers interspersed with non-linear activation functions gives a feedforward neural network, also known as a multi-layer perceptron (MLP). An MLP consists of an input layer and an output node (as per the perceptron), and one or more intermediate layers (also known as hidden layers).

Compared to the perceptron, MLPs have more nodes, allowing more abstract representations to be made; an MLP of sufficient size can approximate any function, a property known as universal approximation [62].

The key advance allowing MLPs to move from theory to practice is back-propagation. Calculating a value in an MLP (known as a forwards pass)

consists of propagating function compositions forwards through a network.

To optimise the network—calculating the gradient of the loss with respect to each parameter—gradients are propagated backwards through the network. These gradients are calculated using the multivariate chain rule, which is made computationally feasibly by using dynamic programming for storing and reusing values.

Model size, Overfitting, and Training Cost

Larger models often come at the cost of a bias-variance trade; as more parameters are added, allowing more complex functions to be approximated, networks are better able to overfit by memorising the training data.

The typical solution to this problem is to train the network on more data, increasing training costs. Furthermore, larger neural networks are more expensive to train as the training cost for each layer is quadratic to the layer width.

Deep Neural Networks

Building larger models that are deeper instead of wider is one solution to overfitting. As more layers are added to a model, the repeated composition of non-linear activation functions allows the fitting of complex hierarchical relationships. Compared to a shallower network with the same number of parameters, a deep network can effectively generalise more complicated functions and is less able to memorise data [18].

However, deeper models are not a free lunch; what they gain by being less prone to overfitting, they lose in optimisation cost. As models get deeper, they generally get harder to train, as the underlying optimisation problem becomes highly non-convex [45].

Many advances in neural networks have been in techniques that make training deeper networks easier. Examples include shortcut layers that allow CNNs (discussed below) to be hundreds of layers deep [25] and attention heads al-

lowing transformers to be trained in parallel [69]. A general trend in machine learning is that larger models allow higher-level abstractions given sufficiently large data sets.

Convolutional Neural Networks

Convolutional neural networks (CNNs) are responsible for much of the popularity of machine learning in computer vision. However, they are not restricted to images but instead work for any-dimension data with a regular grid-like structure. Recent applications of convolution-based networks include IR spectra problems, with these discussed later in Chapter 2.4.

A convolution layer consists of a fixed number of hidden channels. Each channel represents a filter, which is stepped over sliding windows of the data. In each step, the output is a feature representing the dot product between the filter and the windowed data. The kernel size (size of the sliding window filter) and stride (step size for the sliding window filter) are typically governed by hyperparameters.

Typically a convolution layer is followed by a pooling layer, which downsamples feature maps into the average or maximum values. Using multiple filters increases the dimension of outputs, with downsampling used to maintain the previous dimension. The downsampling operation introduces translational invariance; pooled feature maps are invariant to small translations in position.

A side effect of convolution layers is that they create sparsity; rather than a matrix defining the relationship between all nodes in two layers, a relationship is established between the inputs and parameters of the filter. In this way, not only can neural networks be used to identify features, but while training, they can learn what such features are.

Deep Networks for Tabular Data

Despite techniques like convolutions enabling advances in many machine learning applications, for tabular data, deep networks are typically outperformed by

more traditional methods. The typical state-of-the-art for a tabular problem is a variant of a gradient-boosted decision tree (for example, [32]).

For instance, Gorishniy et al. [23] compared both a ResNet and a Transformer architecture for a variety of tabular datasets. The authors found that both architectures outperformed other deep networks yet, in turn, failed to outperform a gradient-boosted decision tree.

Borisov et al. surveyed deep learning for tabular data [9]. The authors concluded that state-of-the-art deep learning methods are still inferior to ensembles of decision trees. However, the authors catalogued several promising methods based around regularising MLP models.

Shavitt and Segal [59] proposed Regularisation Learning Networks, where each weight is paired with a unique regularisation coefficient, effectively allowing each weight to learn independently. A promising result of this approach was that they created very sparse networks, with the majority of nodes being regularised to zero.

Furthering this line of investigation, Kadra et al. [31] use 13 different regularisation techniques with a search method for joint optimisation of each method. The authors find that with sufficient regularisation, MLP models outperform both state-of-the-art deep architectures and ensemble tree methods.

Later in this chapter, we discuss IR spectra data and provide specific applications of neural networks. For now, we note that high dimensionality and multicollinearity are properties for which MLP architectures are naturally suited. Consequently, these models may be advantaged compared to the general case of more heterogeneous tabular data covered in this section.

Activation Functions

Having reviewed the advances in neural networks from the perceptron to the state-of-the-art, we now discuss the individual building blocks and design choices needed for a neural network architecture. We begin with activation functions, denoted $\Phi(v)$, which determine the non-linear dynamics within a

neural network. We briefly discuss the properties of the classical sigmoid, tanh, and ReLU functions.

The sigmoid function takes any real valued input and produces an output within the range $(0, 1)$:

$$\Phi(v) = \frac{1}{1 + e^{-v}} \tag{2.8}$$

The tanh function takes any real valued input and produces an output within the range $(-1, 1)$:

$$\Phi(v) = \frac{e^{2v} - 1}{e^{2v} + 1} \tag{2.9}$$

Unlike the sigmoid function, the tanh function is a zero centred function. These functions are relatively expensive to calculate as they involve exponents and suffer from the vanishing and exploding gradients problem, which we discuss below.

Backpropagation defines gradients for a layer in terms of the previous layer's gradient. For deep networks, with the repeated composition of activation functions, gradients become increasingly sensitive to each other. Given that these functions have a derivative of close to zero for most of the space of real numbers while having a small non-linear region, relative magnitudes of updates across layers tend to either go to zero (vanish) or explode,

The rectified linear unit (RelU) is the baseline activation function for most modern neural networks. It is inexpensive to calculate and with a piecewise gradient of $\{0, 1\}$, suffers less from the vanishing and exploding gradients problem:

$$\Phi(v) = \max\{0, v\} \tag{2.10}$$

In previous spectral applications of deep networks, ReLU activation functions offered a middle-ground between performance and ease of training [40].

A multitude of variants exists that seek to improve on these classical functions. As in Chapter 3, we select ReLU functions for our architecture, we discuss ReLU further.

The downside to the ReLU function is that it cannot handle negative valued inputs. Cases may arise where a neuron gives a negative value regardless of

input, in which case ReLU causes the neuron to be "dead". Care needs to be made with regard to initialisation and initial learning rate so that the outputs of neurons are positive valued.

A proposed solution to the dead neuron problem is to allow small gradients for dead neurons. LeakyReLU does this by including a small slope for negative values, with a gradient controlled by a hyperparameter [38]:

$$LeakyRELU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \text{negative\_slope} \times x & \text{otherwise} \end{cases} \tag{2.11}$$

The randomized leaky rectified linear unit (RRELU) [79], rather than including a fixed slope, includes a small, randomised gradient.

$$RRELU(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{otherwise} \end{cases} \tag{2.12}$$

where a is randomly sampled from $U(min, max)$

Loss Functions

The choice of loss function used to optimise a neural network depends on the problem at hand. For regression problems, the mean squared error (MSE) is typically used:

$$MSE = \frac{1}{n}\sum_{i=i}^{n}(y_i - \hat{y}_i) \tag{2.13}$$

Regularisation for Deep Networks

l_2 regularisation is often used to deal with overfitting. This is done as per ridge regression by adding a penalty term to the loss function (to be employed at each layer). As a consequence of stochastic gradient descent (discussed below), regularisation acts as weight decay; large but infrequent weight changes are penalised relative to small changes.

Another form of regularisation is dropout. Dropout layers are typically applied following a dense layer. During training, a dropout layer randomly

zeroes a small percentage of its inputs [27]. At test time, the dropout is deactivated. Dropout encourages a neuron to more evenly weight its inputs, much like ridge regression.

Early stopping is another form of regularisation. Performance gains between epochs measured on the training set may indicate overfitting to the training data. Instead, performance should be evaluated on a hold-out set distinct from the training and test sets. Either the model state is set to the one with the best performance on the hold-out set or a heuristic halts training.

Optimisation

Neural networks are trained by optimising a loss function using a variant of gradient descent[1].

In the past, this was done by calculating gradients for the entire dataset or intervals for single points. Today, stochastic gradient descent refers to sampling the dataset into (mini-) batches and calculating gradients across each batch (previously referred to as a mini-batch). As opposed to calculating gradients across all data instances, stochastic gradient descent (SGD) is faster and acts as a form of regularisation.

SGD is parameterised by the learning rate, typically denoted α, which controls the rate at which weights are updated. Care is needed; weights will update too slowly to converge to an optimum if the learning rate is too low. Conversely, if the learning rate is too high, weights will zig-zag while never finding an optimum. This is particularly pronounced when the level of curvature (rate of change in the gradient) is high.

Commonly optimisation is done by using multiple passes or epochs of SGD. Typically a fixed number of epochs are run, or the search is run until the model converges or a computational budget is met.

A common strategy is to use learning rate decay; start with a high learning

[1]Second derivative training methods exist, but are not commonly used as they are computationally expensive.

rate and decrease it according to a heuristic. A common rule is denoted by "reduce-rate-on-plateau", which reduces the learning rate by a fixed factor if the loss hasn't improved after a given number of observed epochs.

A now-ubiquitous optimiser is Adam [33]. Adam combines multiple advances in optimisation such as smoothing gradients, learning rate decay and momentum into a single function and is a default starting point for many state-of-the-art applications.

Initialisation

The initialisation of weights can be important. One important property is that expected output values at initialisation should be invariant to the number of inputs. A common rule is to sample values from a uniform distribution ($U(0,1)$) and then apply a scaling factor such as $\frac{1}{\sqrt{n}}$.

Hyperparameter Search

To recap, neural networks are learnt by using gradient descent methods to solve an optimisation problem. This problem is of a large dimensionality and is often highly non-convex. Search for an optimal parameter state for a network is typically stochastic, and both the initial parameter state and the ordering of data can influence the outcome of a search.

To effectively solve this gradient descent problem, applications of neural networks must also determine appropriate values for the parameters that govern the gradient descent method. These include the gradient descent method (e.g. SGD or ADAM), the learning rate, and a learning rate update rule.

Deciding on an appropriate architecture (network architecture search) is a third problem which must be solved in conjunction with the two mentioned above.

We briefly cover several approaches to search and outline why the random search or Monte Carlo approach was selected for experiments in this thesis.

Broadly speaking, a search can be run manually, randomly (the Monte

Carlo approach) or exhaustively (grid search). More advanced techniques (adaptive search) combine these methods by attempting to learn good search candidates based on previous search iterations. Cyclical optimisation searches break grid search down into smaller subsets and optimise for variables conditionally on other variables. Bayesian optimisation [65] iteratively learns promising regions for sampling future searches. Genetic searches [63] define mutation operations that randomly iterate on previously found good solutions, often with a population initialised by a grid-based rule.

For each search strategy, there is a trade-off between exploration and exploitation. For Bayesian optimisation and genetic search, exploration is controlled by the population size and exploitation by the number of generations. This trade-off makes these approaches computationally expensive and is further pronounced by the curse of dimensionality. Furthermore, both Bayesian optimisation and genetic search require several parameters which must be optimised in turn.

Monte Carlo circumvents the exploration-exploitation trade-off through random sampling, making it an effective baseline search method. The success of this approach is thought to lie in the fact that often many of the dimensions of the search space have little effect on performance. While other approaches may waste computational cycles iterating over these variables or adapting to random noise, Monte Carlo search can ignore such dimensions [6]. We note that the aforementioned initial parameter states and gradient descent paths are examples of such variables; not explicitly sampling over variables may add a large amount of noise to grid-based methods, further favouring Monte Carlo methods.

Another reason why the Monte Carlo method may be effective is the lottery ticket hypothesis [20], an argument that much of the success of very large CNN architectures is due to randomly initialised substructures. As these substructures are only a small part of the network, performance gains can be made by building overly large models that are then pruned down. The lottery ticket

approach has a conceptual overlap with the Monte Carlo approach of explicitly searching for random configurations of models.

The selection of search strategy is also dependent on the problem at hand. Much of the reported literature consists of optimising a small number of hyperparameters for a fixed architecture or varying a single design choice for a network. Bayesian optimisation is particularly effective in these applications [58]. Other applications look at very large models with regular substructures. In these applications, genetic search has been effective as these substructures lend themselves to permutations [52, 35, 37].

2.3 Locally Weighted Learning

In supervised learning, where models are trained on labelled instances to make queries or future predictions of unlabelled instances, lazy learning refers to any technique where generalisations about the data (such as parameters for a model) are constructed at prediction time. This is in contrast to eager learning, where generalisations are built at training time. In this approach, unique models are built around each prediction instance rather than building a single global model.

Locally weighted learning is an approach to lazy learning, where query-specific models are built by weighting the training data by proximity to the query point.

Locally weighted learning encapsulates a family of lazy techniques; nearest neighbour methods (such as kNN, described below), weighted average methods (non-parametric methods), and locally weighted regression (parametric regression). The applications of these techniques for IR spectral data are reviewed later in Chapter 2.4.

An intuitive way to understand locally weighted learning is as an interpolation method to smooth scatter plots. Incidentally, this technique was first introduced for solving this problem [13]. We introduce the example of a

sine curve with Gaussian noise added as a visual example (Figure 2.1a) for a data-scarce (left-hand-side) and data-abundant case (right-hand-side). In this example, the linear model under-fits, independent of the level of noise. The nearest neighbour model requires abundant data to provide a good level of fit, and even then still overfits to the noise. Locally weighted models offer a middle ground that smoothly interpolates the data.

Some properties are common across local learning techniques. Building local models helps to deal with changes in the problem domain and fit more complex problem domains. On the other hand, these techniques are memory-intensive and slower to evaluate. Often trickery is required to make these techniques useful for modern data techniques. Another downside to local models is that they are less explainable, an often cited reason for not using them in spectral applications, even when they offer performance benefits [56].

kNN

A ubiquitous lazy learning technique is k-Nearest Neighbours. At training time, data is stored in an efficient structure. At prediction time, a kNN learner selects the k nearest neighbours from the training set based on a defined distance metric. These predictions are then aggregated or voted on by a rule, such as mean, median, or weighted mean. kNN is a non-parametric technique that can model non-linear relationships. It is known to work well for classification, where it has strong performance bounds [5]. For regression, kNN performance is often limited by the level of variance in the dataset.

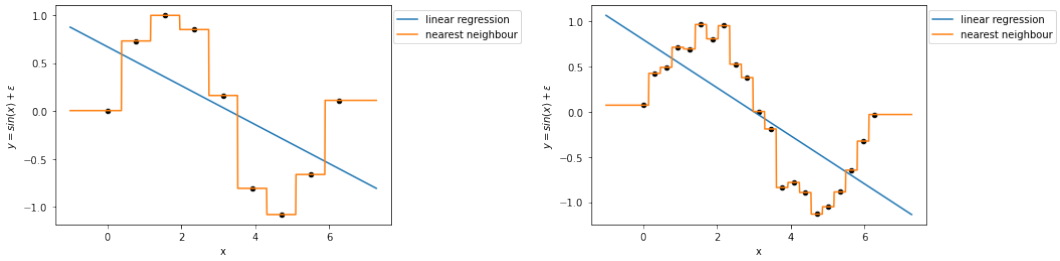
In streaming applications, sliding window-based kNN models are a common benchmark technique as they can adapt readily to concept drift [7].

Non-parametric Kernel Regression

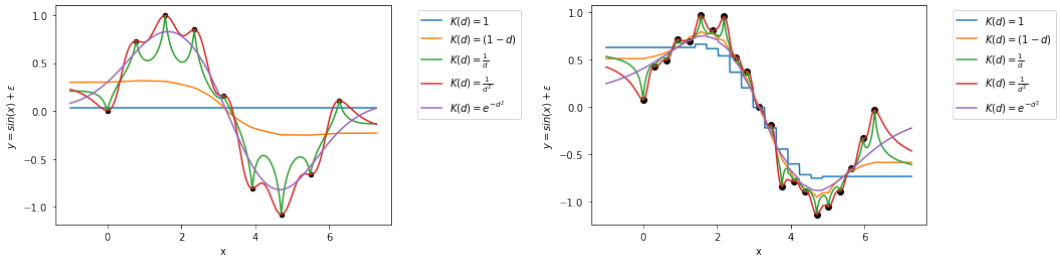
The task of a global regression is to model the distribution between $\mathbf{X}$ and $\mathbf{y}$ for all instances of $\mathbf{X}$. However, a local regression can build a model for every unique $\bar{X}_i \in \mathbf{X}$. This means we can focus on modelling the condi-

Figure 2.1: **Locally Weighted Learners on Artificial Data**. This figure shows how various locally weighted regression models interpolate the noised sine curve $y = \sin(x) + N(0, 0.1)$ with 9 points (left) and 19 points (right). We examine an under-fitting linear regression and an over-fitting nearest neighbour in the top row, kernel options for non-parametric regression in the second row, and kernel options for parametric regression in the bottom row

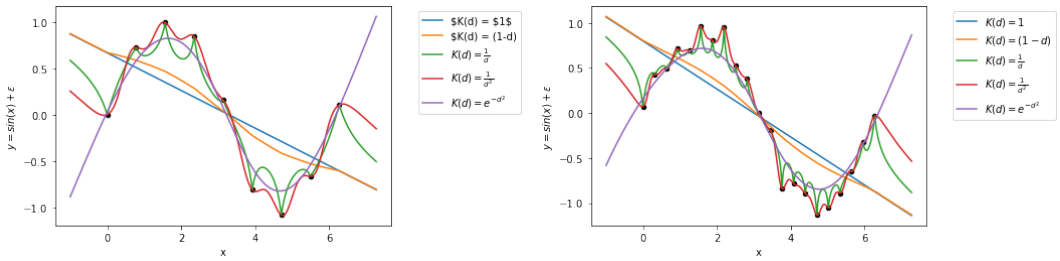
(a) Baseline Methods



(b) Distance Weighted Methods



(c) Locally Weighted Regressions



tional $E(\mathbf{y}|\mathbf{X}) = m(\mathbf{X})$. Models that follow this structure are generally known as kernel regression, a non-parametric approach to regression. The classic technique, known as a Nadaraya–Watson kernel regression (with a single scaling/bandwidth parameter h) [68] calculates a kernel for each queried instance $\bar{X}_q$ as:

$$m_h(\bar{X}_q) = \frac{\sum_i y_i K_h(\bar{X}_q - \bar{X}_i)}{\sum_i K_h(\bar{X}_q - \bar{X}_i))} \tag{2.14}$$

We see that this technique calculates the relationship between $\mathbf{X}$ and $\mathbf{y}$ by voting on $\mathbf{y}$ based on the distances in $\mathbf{X}$, with the kernel (and its bandwidth parameter) used to determine the level of smoothness (over-fitting to data points). This technique prioritises data instances closer to the query point over instances that are further away.

Parametric Kernel Regression

We now consider the case of a global, non weighted, regression model, defined by a parameter vector β and a function $f(\mathbf{X}, \beta)$. For each query (q), we need to minimise the cost function (C) with the given loss function (L).

$$C(q) = \sum_i L(f(\bar{X}, q_i, \beta), y_i) \tag{2.15}$$

For the least squares criterion this is:

$$C(q) = \sum_i (f(\bar{X}, q_i, \beta) - y_i)^2 \tag{2.16}$$

This "global" regression is standard ordinary least squares regression, familiar from Chapter 2,1. Parameters can be found by taking derivatives and are query independent. A local regression can be created by adding weights for each training instance, with these weights being determined by a kernel function $K_h()$ and a distance function between training and query instances $d(\bar{X}, q)$. Optimising the cost function now gives parameters conditional on both the training data and the specific query instance:

$$C(q) = \sum_i L(f(\bar{X}, q_i, \beta), y_i) K(d(\bar{X}_i, q)) \tag{2.17}$$

To generalise the above approaches, local learning techniques centre a kernel around each query point. Then either a linear regression is calculated with instances weighted by a kernel function (known as LOESS, or Savitsky-Golay), a polynomial is fitted (known as LOWESS), or we can directly apply kernel weights (weighted average).

In addition to the regression technique, locally weighted regression involves the choice of three parameters, the distance function, the shape of the kernel, and the scaling parameter (where applicable). We consider each in turn.

Distance Metrics

We limit distance functions to the case where the same function is used across all queries. Distance functions exist that use predefined distance data, vary functions (heterogeneous data instances/structures), or search (optimise) multiple functions for each query.

The most common type of distance uses the Minkowski metric, such as the L_2 norm (Euclidean distance). These metrics typically require data to be scaled such that features are of similar relevance. They also suffer heavily from the curse of dimensionality as distances tend to converge (surface area of n-hypersphere grows exponentially with n).

$$d_m(\bar{X}, q) = (\sum_i (|x_i - q_i|)^p)^{\frac{1}{p}} \tag{2.18}$$

A commonly used alternative is Mahalanobis distance, where distance is calculated between a point and a distribution. However, calculating this requires reliable calculation for mean and covariance values. Mahalanobis distance fits an ellipsoid representing standard deviation in multi-dimensional space, with a distance-vector divided by the width of an ellipsoid along it to measure the distance from the centre. This works better than Euclidean distance for a high number of dimensions. However, it assumes a level of symmetry of data, which is not always the case, particularly when centring distribution around query points.

Kernels

Kernels are used to turn distance measures into a metric that has desirable properties for weighting data. Typical properties of kernels require a maximum value at zero distance, with smooth decay as distance increases with a value of 0 as the distance reaches infinity.

We give a range of common functions below (Equations 2.19 to 2.23), most of which are parameterised by a scaling parameter (h). Some of these enforce a strict boundary on the kernel (uniform and triangular variants), while others do not; values are smoothly reduced to low weight.

$$K_h(d) = \frac{1}{h}, d < h \qquad \text{(Uniform)} \qquad (2.19)$$

$$K_h(d) = \frac{(1-d)}{h}, d < h \qquad \text{(Triangular)} \qquad (2.20)$$

$$K_h(d) = e^{\frac{-d^2}{h}} \qquad \text{(Gaussian)} \qquad (2.21)$$

$$K_h(d) = \frac{1}{hd} \qquad \text{(Inverse)} \qquad (2.22)$$

$$K_h(d) = \frac{1}{hd^2} \qquad \text{(Squared inverse)} \qquad (2.23)$$

Bandwidth

The last parameter to consider is the scaling parameter or bandwidth, h. This is dependent on the kernel function chosen. This can be of fixed width; select all points within a distance of h (Radius-Neighbours) or variable width; select points subject to a criterion, such as the kth nearest neighbour. Depending on the choice of kernel, the bandwidth can represent a strict boundary (e.g. uniform and triangular kernels), or it can control the scale of a smooth function (e.g. standard deviation of a Gaussian).

Applications

A review of applications of locally weighted regression was undertaken by Centner and Massart [11]. PCR and PLS dimensionality reduction, Euclidean

and Mahalanobis distance metrics and uniform and cubic weightings were all compared. Findings were that locally weighted techniques improved upon PLS in two situations while not being worse in another two.

Locally weighted techniques are prominent for IR spectroscopy problems. Specific applications for these problems are discussed in the subsequent chapter (Chapter 1.4). The techniques covered in this section form the basis of the local methods presented in Chapters 3 and 4.

2.4 Spectral Analysis

Infrared Spectroscopy

Infrared Spectroscopy is a non-destructive technique to measure the chemical composition of a substance. As radiation makes contact, it either reflects, absorbs, or transmits through. Sensors are designed to measure the attenuation, the combination of absorption and scattering signals from an emitted source [47]. Typically absorbance is due to the presence of chemical components, while scattering is related to the structure of the substance.

The infrared spectrum is wide and instruments have been produced to cover different parts. Generally, the mid-infrared band (MIR) refers to the 3000-8000 nanometer (nm) band, while near-infrared (NIR) refers to 780nm to 2500nm. The visible spectrum is 400-800nm and is often included within NIR data due to overlap in equipment. Consequently, the term VIS-NIR refers to NIR data that may or may not include the visible band.

The data we look at is point-based; each spectrum consists of measures that are regularly spaced over a defined range of wavelengths.

Measurement

The details of collecting IR spectroscopy data are beyond the scope of this study. Most publications will include a reference to their data collection methods. A review of the technology including examples of applications is given by

Walsh et al. [71].

We make a brief mention of sources of error common with this data. Error can be implicit in measurement, either through the bias of different measurement devices or due to random noise. Variance can also be found between different samples, with instances from different populations possibly sampled with different properties. In fruit, for example, different orchards, cultivars, and growing regions may lead to differences in spectra readings [72].

Applications

Fruit and vegetables are commonly analysed as their high water content makes them suitable for NIR analysis. Pome fruit is the most common application for fruit, followed by stone fruit and citrus. This is partly due to their even size and thin skin minimising scattering. For these fruits, tasks are typically measuring dry matter or total soluble solids (TSS), where very good predictions ($< 0.1\%$ RMSEP) are obtainable. A wide range of other (harder) tasks is attempted, such as skin firmness and acidity [72]. These findings were repeated in a review by Wang et al. [73], noting that while the majority of publications reviewed focused on SSC, acidity and firmness, many other tasks are less successfully attempted.

Within Industry, applications are made at all stages of harvesting, such as determining harvesting time, automatic grading, and quality control [71].

Applications of mid-infrared spectroscopy, while measuring a different spectral band that is typically associated with different properties, overlap substantially with NIR spectroscopy. Applications include measuring the quality of soybeans [19] and the oxidation of oils [70].

IR Preprocessing Techniques

Much of the literature consists of techniques to make IR spectra more suitable for regression. Many of these are general techniques that have been previously covered in Chapter 2.1, while others are domain-related. As an example of

the latter, a typical IR application attempts to measure absorption, which is made easier if the influence of scattering can be removed [56].

Nicolai [47] gives an overview of spectral preprocessing techniques (2007). During the data acquisition process, averaging is often used. Often multiple scans are made and then averaged to reduce any background thermal noise. Depending on the resolution of the measuring device, spectra can be compressed to reduce dimension and provide smoothing.

Mean centring and variance adjustment techniques, as per tabular data, are commonly practised. Mean centring acts as a baseline shift, removing the influence of background thermal noise. Standardisation, where each wavelength is mean centred and scaled such that its variance is one, results in each wavelength having an equal influence in subsequent analysis. However, this can add noise to variables with low base variance, which may reduce performance.

Mean-centring baseline corrections and variance adjustments correct only additive scattering effects. However, multiplicative scattering effects may exist, and it may be desirable to correct this explicitly.

One such technique is standard normal variate correction (SNV). SNV standardises each instance such that every spectrum has a mean of 0 and a variance of 1. SNV corrects for the effects of scattering by removing variation for wavelengths with low absorption [17].

An alternative approach to SNV is to replace each spectrum with a measure of deviation from a baseline spectrum. This approach is known as multiple scatter correction (MSC). Often the mean spectrum is used as the baseline [29]. For each instance, a linear regression with parameters $\hat{a}_i$ and $\hat{b}_i$ is calculated between the instance and the baseline. Each spectrum is then replaced with the following measure of deviation from the baseline:

$$x^*_{ik} = \frac{x_{ik} - \hat{a}_i}{\hat{b}_i} \tag{2.24}$$

Smoothing is a common preprocessing technique, typically with Savitsky-Golay filters, another name for the non-parametric locally weighted regression

covered in Chapter 1.3 but used in a different context. These replace each data point with a polynomial fitted on a window of neighbouring points.

Often Savitsky-Golay filters are used to take derivatives. Typically the first and second derivatives are taken, with the first removing baseline shifts and the second removing the influence of superposed peaks.

Modelling of IR data

NIR data typically has two properties that make it a complex regression problem. The first is non-selectivity; the absorption features of chemical compounds tend to cover multiple, overlapping wavelengths. The second is a direct result of the first; spectra tend to have a high level of multicollinearity [56].

These problems have led to a movement in the literature away from linear regression to Partial Least Squares and non-linear regression techniques [3]. This shift has been enabled by the increase in software and computation power, trends that are still ongoing.

Results often differ by application. As an example, in an application for apples, Nicolai et al. [46] compared kernel PLS to linear PLS, finding no difference in performance. Conversely, in an application for kiwifruit, Lie et al. [36] found that SVM outperformed linear PLS regression.

In an early review, Nicolai et al. [47] argued that there was "little convincing evidence" that non-linear techniques offer advantages with respect to the classical linear algorithms, as many common NIR applications are very linear.

Recent literature reviews covering applications in fruits have identified that while there is no clear state of the art, most applications build a linear model supplemented by domain-related preprocessing. However, there has been growth in the usage of non-linear techniques.

Walsh et al. [72] reviewed Vis-NIR for post-harvest support for fruit in the period 2015-2020, finding 45% of publications use PLS, compared to only 9% using SVM. The authors noticed gaps between published literature and

techniques within commercial use. Saeys et al. [56] review the usage of various VIS-NIR techniques. The authors conclude that increased model complexity and over-fitting are why non-linear techniques have struggled to overtake PLS based regression in popularity.

Locally Weighted Methods

Locally weighted methods are widely used for IR spectra applications. Several implementations that are common in the literature are reviewed below.

While being familiar as a name for this family of techniques, LWR also refers to a specific implementation of a locally weighted regression where first data is transformed into PCA space, with neighbours selected based on Mahalanobis distance in the PCA space [44].

An early application was termed LOCAL. In this method, correlation coefficients are used as a distance metric, with a PLS regression being calculated on selected neighbours [61].

Shen et al. [60] compare two locally weighted methods. The first, Local PLS (LPLS), calculates neighbours in the space of the original spectra. A PLS is then calculated for the selected neighbours [60]. The second, LPLS-S, calculates PLS on the full set of spectra. A linear regression is then calculated based on a neighbourhood determined in the space of PLS scores. Calculating neighbours on PLS scores instead of the original feature space increased speed by an order of magnitude while maintaining the level of performance.

The Local Central Algorithm (LCA) first calculates PCA, which is used to calculate neighbours. Predictions are then made by selecting the mean or median neighbour [81], an example of a non-parametric, locally weighted regression.

These models are all recognisable as applications of (the previously reviewed) kernel regression techniques. While the specific language of these models may differ, the same core set of decisions are made; the space in which to calculate neighbours, the choice of kernel and distance function and the

regression technique.

Global and Specific Models

Much of the literature consists of building individual models on subsets of data delimited by a variable such as a season, cultivar, or growing region [51, 66, 8, 75].

The success of these subset specific methods over global models, or models built over an entire dataset, implies a tradeoff between generalisability and performance. In theory, more powerful models trained on larger, more diverse datasets should be able to give the performance of more specific models while being robust to variation. This has been a motivation for the recent increase in the popularity of neural networks for NIR applications [3] and the creation of larger datasets [4].

Deep Techniques

Applications of neural networks go back decades, with a more recent uptake as software and data availability have improved.

Anderson and Walsh [3] reviewed recent developments for deep networks for VIS-IR. The authors find that model robustness to population variance is a key motivator for many of these applications. Deep networks have been used for a range of purposes in spectral data, such as data reduction, pattern recognition and multivariate regression. [12].

In an application for grapes, Janik et al. [30] compared ANN to PLS, finding that ANN was more generalisable to new instances than both global and local PLS models. The authors also showed that the two techniques were complementary; using PLS scores as inputs for an ANN leads to quicker and easier to train neural networks by dimensionality reduction.

More recently CNN architectures have been proposed [14]. As many spectral preprocessing techniques consist of averages over a sliding window, a 1D-convolution can be seen as a deep equivalent of classical spectral preprocessing,

where the optimiser learns the appropriate sliding window relationship [15].

Rong et al. built a deep CNN model to classify peach varieties [53]. The model trained 100 data points for each of the five peach varieties using convolution layers combined with dropout, batch, normalisation and pooling. This model achieved 100% accuracy on the training set and 94% on the test set.

2.5 Lazy-Deep Networks

This thesis looks at an alternative to the above based on combining relevant techniques from the tabular deep learning and the IR spectroscopy fields in a complementary manner. Monte-Carlo search is used to find architectures for deep-MLP networks. PLS and locally weighted methods are used to enhance the start and end of these networks respectively. The next chapter outlines the methodology and techniques used to find these architectures.

We first present a brief review of other work that combines these specific techniques.

Outside of spectral analysis, lazy models have been used with deep models in several situations. The property of kNN models being robust to random noise [21] has led them to be proposed as a solution to labelling noise for CNN models. One example is kNet, where kNN and a deep approximation of kNN were applied after the penultimate layer of a deep neural network [42]. kNN models have also been used to add confidence estimates, model interpretability and robustness to deep neural networks [48]

The concept of using deep networks as feature extractors for classical predictors is common in the field of transfer learning, where pre-trained learners are adapted to new data. A common strategy to "fine-tune" a model is to freeze the feature extractor elements and fine-tune the head layer elements. When fine-tuning on small amounts of data (few-shot-learning), both fine-tuning and state-of-the-art approaches are often outperformed by classical methods such as logistic regression [24]. In such approaches, lazy kNN models are often

considered as candidate predictors [74].

Given the work on lazy-deep networks, the next chapter outlines the approach taken in the thesis to select good deep network architectures and apply lazy methods. Consideration is also given to the important topic of evaluation, which is crucial for making performance judgements when comparing different methods.

Chapter 3

Methodology

This chapter details methodology for finding deep network architectures and evaluating their performance. We start by discussing architecture for both deep networks, lazy regression and network search (Chapter 3.1). Some theoretical analysis of these models is provided in Chapter 3.2. Chapter 3.3 looks at the evaluation approach for subsequent experiments.

3.1 Architecture

The deep architecture is constructed as two components, a feature extractor and the prediction or head layer. During training, these are learnt end-to-end. Predictions can then be made by passing data through both the feature extractor and head layer or by extracting features which are passed onto an independently learnt lazy method.

3.1.1 Deep Architecture

The feature extractor is made up of a sequence of blocks. We restrict the neural network architecture to the deep-MLP class of models; each block contains an activation single dense (hidden) layer, an activation function and possibly other layers such as dropout or batch normalisation. The head layer is a single linear

layer which produces a real-valued scalar output[1].

In Chapter 2, several promising applications of MLP to tabular data were highlighted, with MLP approaches also seeing applications for IR spectra applications. While a deep-MLP approach may or may not improve performance compared to variants with a single hidden layer, building deeper networks allows us to decrease the width of layers, which is desirable for subsequent feature extraction. We use networks with between 2 and 5 hidden layers as a middle ground between depth and width.

3.1.2 Network Search

The particulars of Monte Carlo search involve 100 candidate models, each trained over 100 epochs using the cross-validation approach that is covered later in this chapter (Chapter 3.3). The optimisation is done with either standard SGD or Adam, with initial learning rates as discussed in the next section. MSE is used as a loss function, and a uniform batch size of 32 is used for all datasets.

As noted in the literature review, ReLU activation functions are computationally efficient and make optimisation easier for deep networks as compared to exponential-based activation functions. ReLU, as well as the RReLU and LeakyReLU derivatives, are used.

A complete summary of the variables explicitly sampled in Monte Carlo search is shown in Table 3.1. To make the search feasible, we restrict the size of each model. Depth is limited to no more than 5 layers and width to no more than 50 outputs for the penultimate layer. We also enforce that each layer must (non-strictly) decrease in size compared to the previous layer. Search is restricted such that only a single class of activation function is used in each model. The other hyperparameters governing blocks are batch normalisation

[1]The simple nature of the head layer is a product of the problem domain. A classification task would require the addition of a soft-max activation function for example. Likewise, CNN models may include a pooling layer.

Table 3.1: Search Variables

Variable	Options
Number of layers	$\{1, ..., 5\}$
Number of features	$\{1, ..., 50\}$
Activation function	{ReLU, RReLU, LeakyReLU}
Batch normalisation	$\{False, True\}$
Dropout	$\{False, True\}$
Optimiser	{SGD, Adam}
Learning rate update	{None, ReduceLROnPlateau ExponentialLR, CosineAnnealingLR}
Batch Size	32
Epochs	100

and dropout layers. These are defined as binary variables and are included in all or no blocks. If dropout is included, it is sampled from the range $(0.01, 0.1)$.

.

3.1.3 Learning Rate Finder

During preliminary experiments, a poorly selected learning rate dominated all other hyperparameters; too high a learning rate and the loss grew quickly; too low a learning rate and 100 epochs were insufficient to move models far from their initial state.

A solution to this problem is to experiment with learning rates and select sensible ranges manually. However, this may lead to better ranges being selected for specific datasets, or the sensible ranges may be conditional on other hyperparameters.

Another solution is to use a learning rate finder. The standard implementation of this is to start with a very low learning rate and increase it every

epoch, selecting as an initial learning rate the point with the sharpest decrease in loss [64]. However, this approach is highly sensitive to data ordering and model initialisation.

The solution implemented was to use learning rate schedulers that start with a very high learning rate and reduce it over an experiment. This makes use of two concepts. The first is that there is an asymmetry between a too high learning rate (infinite loss) and a too low learning rate (small improvements in loss). The second is that as we are training models in parallel, we can use information from other models to bring the learning rate down over an experiment.

This works as follows. If we observe an NaN/infinity error, we know the learning rate is too high. We reset the model and reduce the learning rate by an order of magnitude[2]. Secondly, if a model's performance (over an epoch) is over ten times the best current performer, we multiply (/divide) its learning rate by a factor sampled from $(0.1 - 0.9)$, with the operation dependant on the direction of the best model's current learning rate.

In practice, this worked well, but not perfectly. There may be interaction effects between the dynamic learning rate finding heuristic and the learning rate schedulers, with this effect growing with each interaction. It is also possible that models may get left behind, continually being reset while the learning rate of the best model continues to decrease. Another possibility is that the best learning rate for one model is not the best for another model.

3.1.4 Lazy Predictor Architecture

To recap from the previous review of locally weighted methods; key design choices are the neighbourhood space, kernel function and distance metric.

The approach taken to locally weighted regression is to define a uniform kernel with a bandwidth equal to the distance of the kth nearest neighbour;

[2] $lr_i = e^{-(t_0+t_i)}$ where $t_0 \sim Uniform(0.5, 3.5)$. This range is arbitrary and essentially constitutes a hyperparameter

we assign the k-nearest-neighbours a weight of 1, and all other instances a weight of 0. The choice of k is left as a hyperparameter to be varied in experiments. Linear regression is performed using standard ordinary least squares optimization.

Neighbourhoods are calculated based on the transformed feature sets. Even though this increases the influence of the deep network, as compared to calculating neighbourhoods based on the original features, the decrease in dimensionality has considerable benefits, particularly given that datasets experimented with vary greatly in the number of features.

During experimentation, we discovered instability with the LWR models, often a single outlier prediction that would substantially degrade the measured performance. We hypothesize that this is caused by making predictions at points outside the domain of the k-nearest neighbours. As a solution, we investigate a second setting for LWR, denoted LWR_b, where we restrict predictions to the range of $\mathbf{y}$ values for the k-nearest-neighbours. This is effectively a locally weighted regression which we do not allow to extrapolate:

$$\hat{y} = \begin{cases} f(x) & \min y_{train} < \hat{y} < \max y_{train} \\ \max y_{train} & \hat{y} > \max y_{train} \\ \min y_{train} & \hat{y} < \min y_{train} \end{cases} \tag{3.1}$$

3.2 Theoretical Analysis

Having outlined the architecture for combining deep networks with locally weighted methods, we now present a brief analysis of the interactions between these two components.

3.2.1 Deep Feature Extractors

By definition, the head layer represents an additive linear relationship between the extracted features and the target values; a linear regression. This structure means that the quality of predictions depends upon the ability of the feature

extractor to generate features with both linear relationships to the target and to avoid multicollinearity between features[3] and for the head layer to be well trained for a given feature extractor. We explore the dynamics of this by analysing the hypothetical performance of a model using a two-factor model, the quality of the feature extractor and the quality of the linear (Head_Layer) for the given feature extractor.

If we assign quality to be one of two arbitrary variables, good and bad; then a model belongs to one of the following four categories:

- Bad Features, Bad Head_Layer - Will give poor performance. These models can be discarded.
- Bad Feature, Good Head_Layer - May give good performance but will not likely generate good features for lazy predictions.
- Good Features, Bad Head_Layer - May give good performance but can likely be improved upon by lazy method.
- Good Features, Good Head_Layer - Will give good performance.

This analysis shows two ways in which alternate predictors can be beneficial. The first is by providing a more optimised head layer. The second is by adding non-linear dynamics for poorly optimised feature extractors. Model performance is not a good indicator of these two factors, as it may not distinguish between models with poor feature extractors and a well-trained head layer and the converse case.

In several experiments in Chapter 4, we compare locally weighted regression are compared to lazy linear regression. If the linear regression gives performance gains comparable to local methods, this likely indicates a poorly optimised head layer. Conversely, if local methods outperform a linear regression, then likely the non-linear dynamics are more effective at using a poorly optimised feature extractor. As a further comparison, we also compare the

[3]Note: this is more intuitive for classification where linear separations are made

approach to kNN models, for which performance gains should be greater in the case of a poorly performing feature extractor.

3.2.2 Local Learner Analysis

The above analysis has several implications for the local learners. First, the deep learners are trained to give features that have a linear relationship. Second, care needs to be taken regarding the bias of deep networks, given that these have a tendency to overfit. Third, care also needs to be taken regarding variance, as any error present in a deep model will add to the total error term of the architecture.

A notable contribution of using local learners is adding extra regularisation compared to predictions made with the head layer. By building a local, non-weighted regression (a uniform kernel) at each query point, we add further regularisation relative to other choices of the kernel. Enforcing a strict neighbourhood boundary excludes training instances that are outliers in the transformed feature space while avoiding weighting instances highly due to noise introduced by the deep learners.

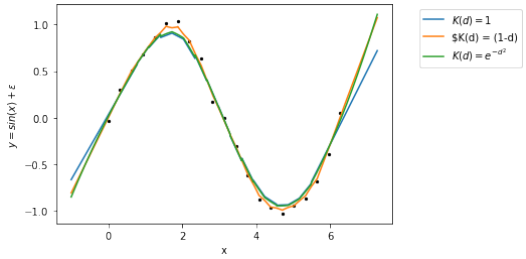
To gain some intuition about how the implementation of a locally weighted regression model behaves, we turn again to the sine curve example (Figure 3.1). In the top row, examples with scarce data are given. As the number of neighbours decreases, the non-linearity of the locally weighted models increases. With a low k, the uniform kernel still gives a good fit, with a triangle kernel being approximately equal to the Gaussian kernel.

In the bottom row, examples with abundant data are given. Here the choice of the kernel is less relevant. A smaller value of k leads to overfitting to noise, while higher values of k fit the data perfectly for all kernels. Note that as k continues to increase, the models would revert to being linear.

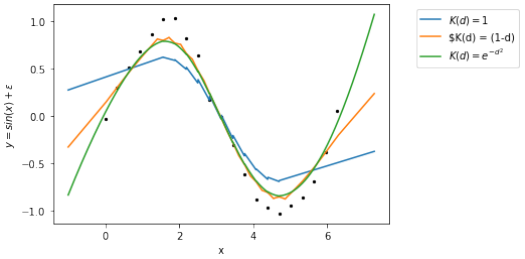
From this analysis, we can infer a trade-off between modelling more complex relationships and risking overfitting the data. The level of non-linearity can be controlled by the value of k, with appropriate values depending on the

Figure 3.1: **k-Neighbour Locally Weighted Regressions.** We compare the uniform, triangle and Gaussian kernels. This is shown for a scarce set of data with 20 training points (top), which we test with 5 and 10 neighbours. The case of an abundant set of data with 2000 training is also shown (bottom), which we test with 100 and 1000 neighbours.

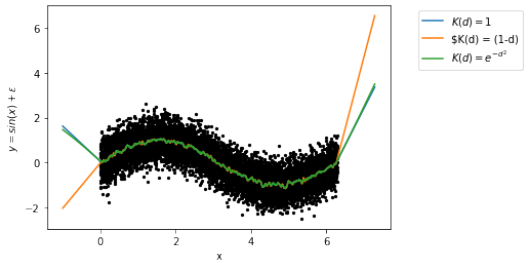
(a) k=5, Scarce Data



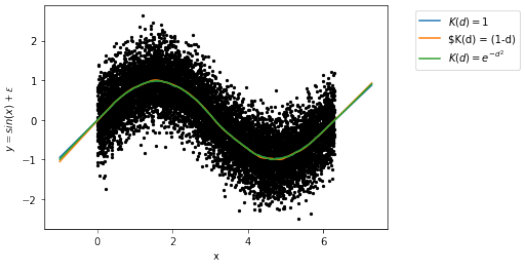
(b) k=10, Scarce Data



(c) k=100, Abundant Data



(d) k=1000, Abundant Data



abundance of data.

3.2.3 Asymptotic Complexity

An approximation of the asymptotic complexity of the various models evaluated in this thesis is made by breaking down each model into a series of matrix multiplication operations.

For two matrices of size $i \times j$ and $j \times k$ respectively, the asymptotic complexity of naive matrix multiplication is $O(ijk)$. Improvements beyond this depend on the relative shapes of the matrices; for example, for two square matrices of size n, the best-known implementation is approximately $O(n^{2.7})$ [16].

The cost of a pass through a deep network, calculating k-nearest-neighbours

Table 3.2: Asymptotic Analysis

Step	Train	Test
Deep	$O(n_1 L m^3)$	$O(n_1 L m^3)$
kNN	$O(n_1)$	$O(n_2 n_1 (m+k))$
LR	$O((n_1^2 m + n_1 m + m^3))$	$O(n_2 m)$
Deep-LR	$O(n_1(Lm^3 + (k^2 d + kd + d^3)$	$O(n_2 L m^3)$
Deep-kNN	$O(n_1 L m^3)$	$O(n_2(Lm^3 + n_1(d+k))$
Deep-LWR	$O(n_1 L m^3)$	$O(n_2(Lm^3 + n_1(d+k) + k^2 d + kd + d^3))$

and calculating a linear regression are in Table 3.2, with notation as follows:

1. A dataset has m attributes, n_1 train instances and n_2 test instances.

2. Deep networks have L layers, with the penultimate layer (feature extractor) having an output size of d.

3. A kNN has a neighbourhood of size k.

A Lazy-Deep approach adds little to no overhead at training time while increasing cost at test time. LWR models are more computationally expensive than kNN models. Key variables to control are the number of features (d), which is cubic for Deep-LWR and the neighbourhood size (k), which is quadratic for Deep-LWR. For streaming solutions, the sliding window size (n_1) is also important.

3.3 Evaluation

This section outlines the approach to experimentation. Discussed are the datasets, the evaluation for batch experiments and the evaluation for streaming experiments.

Table 3.3: Proprietary Datasets

Dataset	Num Instances	Num Attributes	**y** variance
D1	1438	1702	747.5032
D2	7329	1701	236.1148
D3	13916	146	344.3272
D4	411246	129	473.3076

3.3.1 Data Sets

Experiments on four (proprietary) infrared datasets, which we denote D1 to D4. These datasets are used because they are larger than those typically found in the public domain, providing a more robust environment for testing models. A public domain Mango dataset is included in the evaluation as it is relatively large and permits comparison to benchmarks published in the literature.

The number of features and instances are summarised in Table 3.3. Data is abundant for each set, making them suitable for deep networks. For D4, we sample 10,000 instances to keep experiments comparable to the other datasets. In streaming experiments, D4 is used due to its abundance of data.

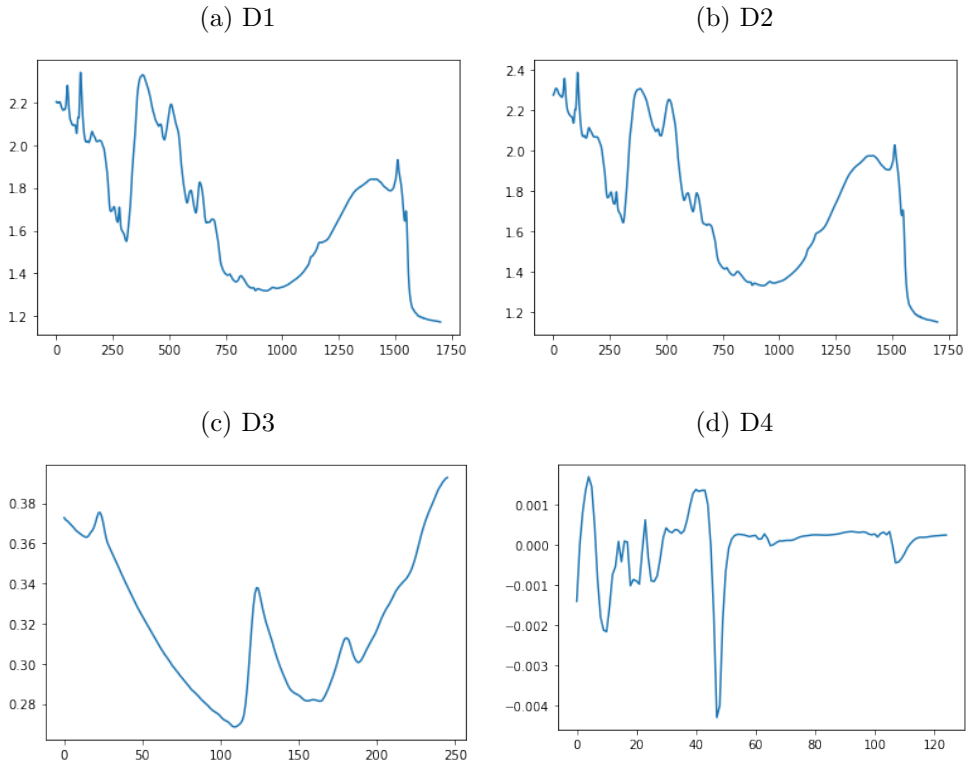
Spectra are shown in Figure 3.2. D1 and D2 show similar spectra, while the other two proprietary datasets show very different patterns for models to learn.

3.3.2 Batch Evaluation

The approach to modelling necessitates combining several different libraries. We use scikit-learn [50] for classical regression techniques and PyTorch [49] for neural networks. We also make use of River [43] for streaming. Heavy use was made of the adapter design pattern so that models from these libraries could be evaluated interchangeably.

Iterating over the datasets proved to be a computation bottleneck. As

Figure 3.2: **Median Spectra for each Dataset**



a result, we made the decision to train deep networks in parallel, with preprocessing applied directly after the data is loaded. As a consequence, this limits the deep models to all having the same batch size and having the same preprocessing.

Randomization is controlled with a seed (for random, numpy, and torch). We shuffle each dataset on loading, then leave everything thereafter (including cross-validation splits) deterministic where possible.

Batch experiments were performed using the following procedure. We reserve part of the data (1/6) as a test set, with the remaining data (5/6) used for training. The training data is split into 5 evenly sized partitions. We then perform five-way cross-validation (5-CV), such that in every fold, 3 parts are used for training, 1 for internal model validation and 1 for testing. This is done so that each instance is used for testing once. We score each experiment based on R^2 and MSE on the combined set of predictions. After cross-validation, we build a final model on the entire training set (except for a component reserved

for internal validation).

The extra fold for internal validation is withheld so that so deep models can stop early. This is a common practice to prevent overfitting, and we want the data used to determine this decision to not overlap with either the training set or the test set for each fold. To maintain consistency between classical and deep models, we reserve this partition for all models.

Performing cross-validation limits overfitting and provides an element of replicability; to perform well in these experiments, models must perform similarly over 5 independent runs, which involves training the same model (from the same initial state) several times over.

3.3.3 Streaming Evaluation

In the discussion on deep networks, most of the work has been done in the batch setting. Data is taken as fixed; we perform an explanatory analysis and break the dataset into a training and test scheme before building a final model.

In contrast to this, the streaming setting is concerned with data that arrives in real-time (e.g sensors, web media data and so forth). In the batch setting, new data might be handled by rebuilding the model to include this new data. In the streaming setting, we look to build models incrementally (without having to rebuild each model). An example of this is the Hoeffding tree, which converges asymptotically (with the amount of data) to a batch-trained decision tree [7]. Data streams involve additional complications over batch data. Firstly, care has to be taken with evaluation. Two key variables are the level of the increment (process every individual data, or wait for batches of a given size) and the evaluation setting (interleaved train-test, or prequential evaluation).

Furthermore, the data is not a known quantity; in a batch setting, we can remove obvious outliers. In the streaming setting, this has to be done in real-time. Also, class or feature distributions may change (the problem of concept drift) over time.

Lastly, we have to maintain resource constraints (we deal with potentially infinite amounts of data). Common approaches include the use of sketches (memory saving statistics) and maintaining sliding windows of data.

For all of these reasons, many deep networks are still computed in a batch setting, even for streaming data. A common, hybrid approach is known as online learning, where models are updated infrequently in response to new data (such as overnight). While this is becoming more popular and represents an improvement over the train-once model, this still falls short of true incremental learning.

Streaming experiments are done with prequential evaluation; each data instance is used once to test, then once to train. Models are pre-trained on all of the data used for cross-validation to be comparable to the batch setting.

Chapter 4

Experimental Results

This chapter contains the experiments and results for this thesis according to the methodology outlined previously in Chapter 3. Chapter 4.1 covers batch experiments for the 4 proprietary data sets. These experiments are then repeated on the public domain Mangoes data set in Chapter 4.2. Chapter 4.3 covers experiments performed in a streaming setting.

4.1 Batch Experiments

This section establishes performance for the lazy-deep methods in a batch setting. The starting point is to establish performance benchmarks for a range of classical learners. Deep learners are then evaluated as predictors and as feature extractors for lazy learners. Based on the outcomes of these experiments, we select candidate models to be compared on a held-out test set that has not been used during the training phase.

4.1.1 Classical Approaches

Three strategies for regression are evaluated, Linear Regression (LR), Partial Least Squares Regression (PLSR) and Principal Components Regression (PCR). For both PCR and PLSR, we consider a number of latent variables (components) ranging from 1 to 100 throughout these experiments.

The first set of experiments considers the following hyperparameters for these models:

- LR is parameterised by the choice of standardisation (LR_s) and l_2 regularisation (LR_r), including doing both ($\text{LR}_{s,r}$). The level of regularisation is set to an arbitrary value of 0.01.
- PLSR is parameterised by choice of standardisation (PLSR_s).
- PCR is parameterised by choice of standardisation (PCR_s) and whitening ($\text{PCR}_{s,w}$), including doing both.

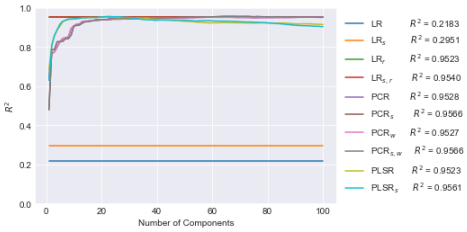
Results are shown in Figure 4.1. We go through each dataset in turn. For the D1 dataset (a), PCR and PLS approaches perform comparably to a ridge regression for a well-selected number of latent variables. All three regularisation strategies outperform a default LR. For the D2 dataset (b), we again see all three approaches perform similarly for certain hyperparameters, with non-l_2 LR again underperforming other approaches. D3 (c) shows the same pattern, with the difference that for LR, it is the standardisation rather than the L_2 regularisation that gives a good level of performance. D4 (d) displays more noise with regard to the number of latent variables, with all three strategies again displaying similar results for certain settings.

To summarise these experiments, each regularisation strategy gives a similar level of performance with appropriate hyperparameter settings. A consistent result across all datasets is that PLS requires fewer latent variables than PCR to converge in performance, an expected result given that PLSR selects features based on additional information to the target. Whitening (for PCA) and standardisation (for PLS and PCA) seemed to have little impact.

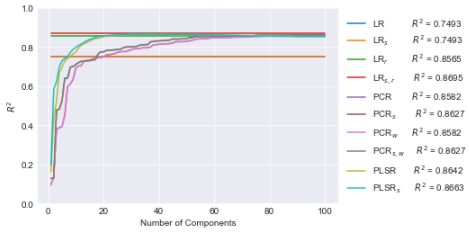
A surprising result was that l_2 regularisation was often sufficient to match the performance of PLSR and PCR. However, we ignore this technique for now, as compared to the other approaches, it does not perform any dimensionality reduction. Instead, it remains an additional tool that can be used to enhance other approaches.

Figure 4.1: **Cross-Validation Results for Classical Approaches**. Shown are results for the various hyperparameter configurations with the number of latent variables ranging from 1 to 100. Included in the labels are the best R^2 scores achieved across the range of latent variables.

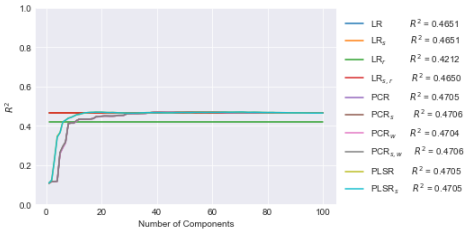
(a) D1



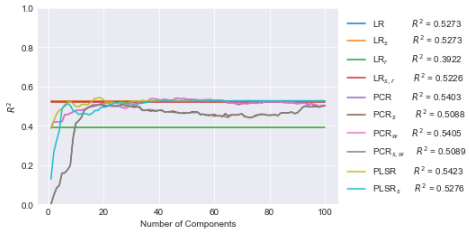
(b) D2



(c) D3



(d) D4



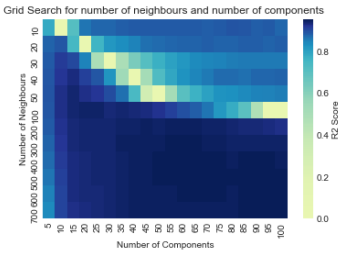
The next experiments looks at locally weighted regression on features extracted from PLS and PCA, denoted PLS-LWR and PCA-LWR, respectively. We compare these to results for locally weighted regression on the raw spectra. Based on the previous set of experiments, normalised PLS and PCA variants are selected.

A grid search is performed over the number of latent variables $lv = \{5, 10, ..., 95, 100\}$, and the number of neighbours $k = \{5, 10, ..., 45, 50, 100, 200, ..., 1000\}$, with results shown in Figure 4.2. The proportion of dark blue space corresponds to how hard the regression problem is for each dataset, with D1 giving higher R^2 scores than the other datasets

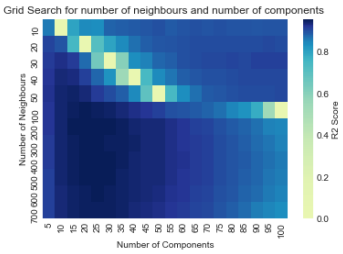
For each dataset, we see either a lighter diagonal line or a lighter upper-right diagonal region. This corresponds to the case of optimised linear regression, where the degrees of freedom equals the number of data points (square matrix). As a rule of thumb, we want to select a number of neighbours larger than the

Figure 4.2: **Parameters for Locally Weighted Regressors** with PCA (left) and PLS (middle) and no (right) preprocessing. Yellow indicate poor performance and dark blue good performance. Note that the colour gradient is non-linear in order to highlight the best regions.

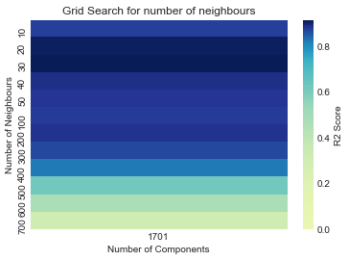
(a) D1, PCA



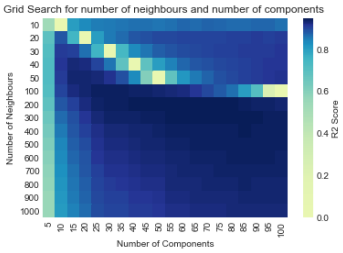
(b) D1, PLS



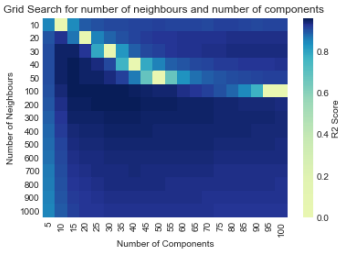
(c) D1



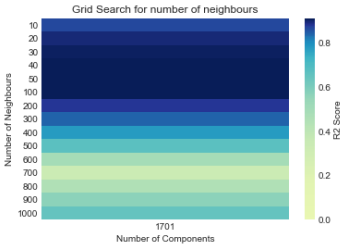
(d) D2, PCA



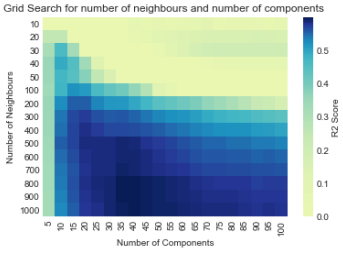
(e) D2,PLS



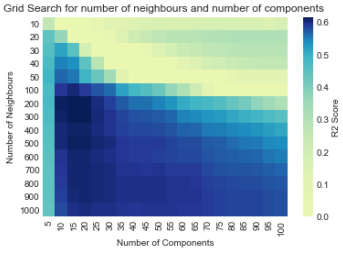
(f) D2



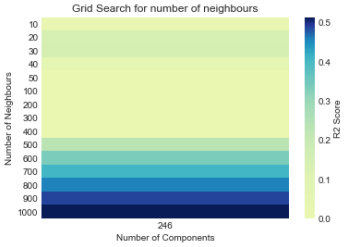
(g) D3, PCA



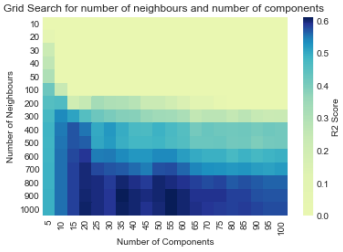
(h) D3, PLS



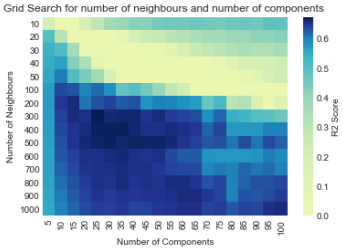
(i) D3



(j) D4, PCA



(k) D4, PLS



(l) D4

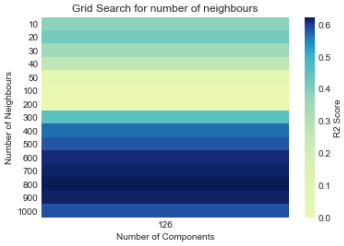


Table 4.1: **Cross-Validation Results for Classical Approaches.** The best result for each dataset is in bold.

	D1		D2		D3		D4	
Model	MSE	R^2	MSE	R^2	MSE	R^2	MSE	R^2
LR	33.86	0.9540	29.02	0.8695	175.8	0.4651	230.3	0.5270
PLSR	32.32	0.9561	29.75	0.8663	174.0	0.4705	222.7	0.5426
PCR	31.98	0.9566	30.56	0.8627	174.0	0.4706	223.1	0.5418
LWR	64.17	0.9128	20.1007	0.9096	160.14	0.5128	183.77	0.62258
PLS-LWR	**29.59**	**0.9598**	**10.19**	**0.9582**	**131.5**	**0.6154**	**160.1**	**0.6730**
PCA-LWR	34.87	0.9526	12.83	0.9474	137.0	0.5994	190.6	0.6107

number of latent variables.

Compared to PCA, PLS feature extraction gives darker blue regions further to the left, indicating that PLS-LWR models require fewer features than PCA-LWR models, consistent with the previous set of experiments. There is no clear best number of neighbours across datasets or feature extraction approaches.

We now compare the performance of each model's best hyperparameter configuration, with results shown in Table 4.1. For each dataset, the PLS-LWR model gave the best performance. PLS-LWR outperforms PCA-LWR even when the feature extractors used as predictors (PLSR and PCR) give very similar performance. Compared to the best non LWR model, this represents an improvement in R^2 of 0.0032 for D1, 0.0887 for D2, 0.1448 for D3, and 0.1304 for D4.

These experiments have set benchmarks for performance. Based on these results, the focus will be on PLS feature extraction in future experiments as it requires fewer latent variables (dimensions) than an equivalently performing PCA model. As there is no clear pattern for an optimal number of neighbours for LWR models, we will be required to keep searching for this hyperparameter in subsequent experiments.

4.1.2 Monte Carlo Search

We now experiment with deep networks. These are searched and selected as discussed in the methodology section, with each search consisting of 100 models

Table 4.2: **Input Features and Parameter Counts**. Inputs are consistent across models in Monte Carlo search while parameter counts are averages.

	Std		PLS	
Dataset	Inputs	Parameters	Inputs	Parameters
D1	1702	1,944,981	22	1,040
D2	1701	1,944,980	36	2,379
D3	146	47,930	63	7,967
D4	129	15,228	34	2,329

trained for 100 epochs. The cross-validation splits used for the classical models are repeated.

Two preprocessing schemes are evaluated. The first, Std-Deep, standardises inputs. The second, PLS-Deep, uses PLS preprocessing, with the number of latent variables selected from the range $(0, 100)$ based on the performance of a standalone PLS regression. For both approaches, the number of extracted features (the output size of the penultimate layer) is limited to 50. The effect of these preprocessing schemes on the input size and subsequent average parameter count of deep models is shown in Table 4.2. Parameter count reduces by three orders of magnitude for the datasets with over 1000 features.

In Figure 4.3 individual fold scores are compared to overall scores. While there is variation across folds, with the level of variance differing by dataset, overall individual fold scores tend to agree with each other. With Std preprocessing, we see that for approximately the 20 worst performing models for D1, D2 and D4, performance breaks down across all folds, while this is not observed for PLS preprocessing.

We also look at the distribution of errors for each model (Figure 4.4). The median error is low for all datasets; a heavy upper tail dominates the mean error. The magnitude of this effect varies by dataset and preprocessing technique; for five out of eight cases, the median error is below the 99% quantile.

PLS and Std preprocessing make little difference in the distributions of error.

The best five results of the Monte Carlo search for each scheme and dataset are shown in Table 4.3. We see that difference in performance is small across the five best models for both preprocessing schemes, indicating that the best performing models are not outliers. The best PLS-Deep model outperforms the best Std-Deep model for D1 and D3, with the converse being true for D2 and D4.

4.1.3 kNN predictors

A kNN approach is evaluated by performing a broad grid search. Values of k are taken from the set {5, 10, 20, 50, 100, 200, 500}, and a uniform (mean) voting rule is used[1]. Results for this search are compared to the baseline deep models shown in Figure 4.5. The choice of k is shown to be a key determinant of performance. While the optimum value differs by dataset, preprocessing technique, and model, very large ($k = 500$ and $k = 1000$) and occasionally small ($k = 5$) values lead to poor performance.

The five best deep models with a kNN predictor are compared to the base models in Table 4.3. These are selected by selecting the best performing kNN for each deep model and then taking the best 5 models. In four out of the eight cases, kNN predictors perform similar to or worse than the baseline deep models while performing better in the other four cases.

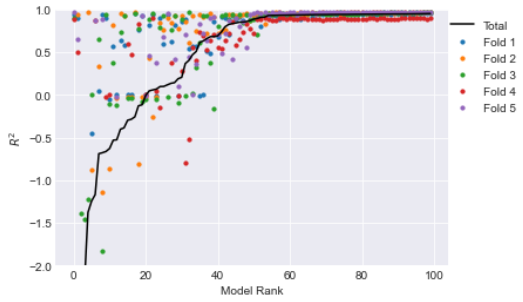
4.1.4 Locally Weighted Models

Locally weighted regression is evaluated, like the kNN approach, by performing a broad grid search over the set $k = \{100, 200, .., 1000\}$. The results of this search are compared to the base deep models in Figure 4.8. In each of the eight cases, LWR models improve on the low ranked deep models before eventually converging in performance or slightly improving upon the higher-

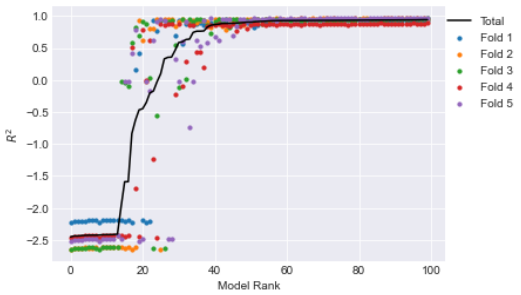
[1]Experiments were also performed with a distance weighted voting rule, however results between the two approaches were similar, so the latter were excluded for the sake of simplicity

Figure 4.3: **Cross-Validation Fold Scores**. Deep models are ranked by performance on the combined fold validation set.

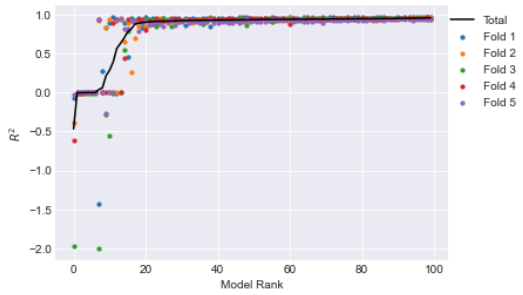
(a) D1, PLS



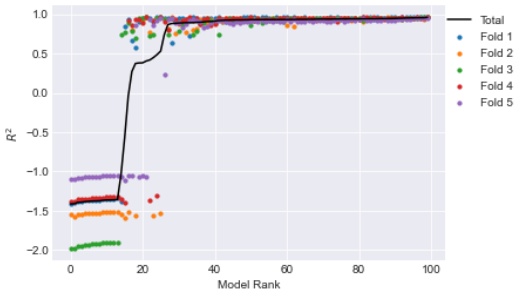
(b) D1, Std



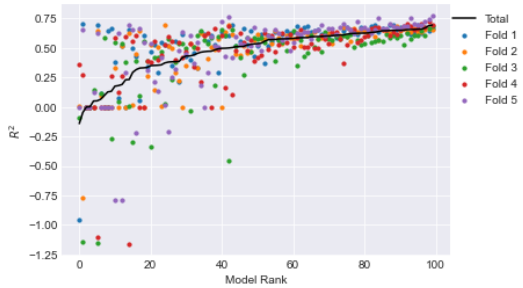
(c) D2, PLS



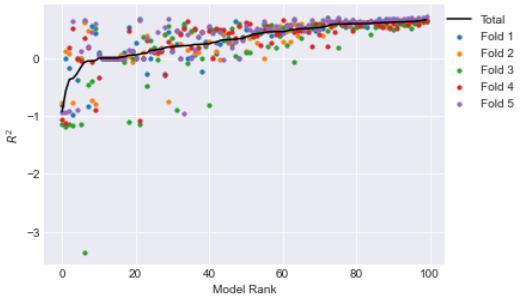
(d) D2, Std



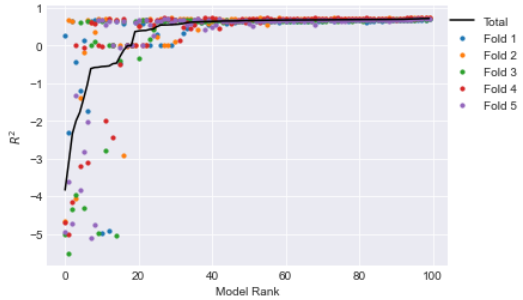
(e) D3, PLS



(f) D3, Std



(g) D4, PLS



(h) D4, Std

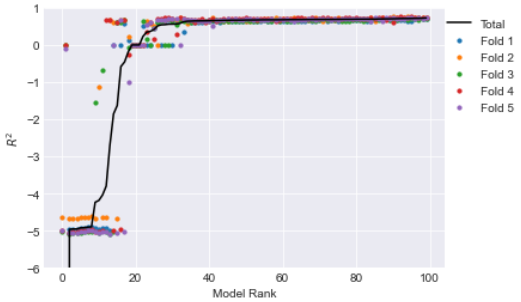
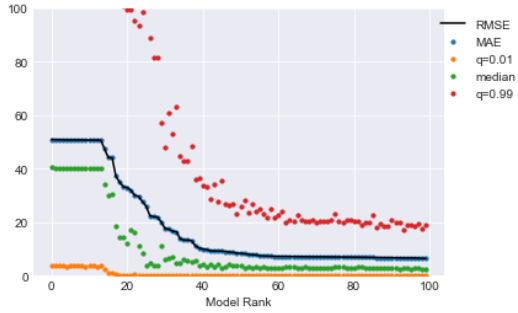


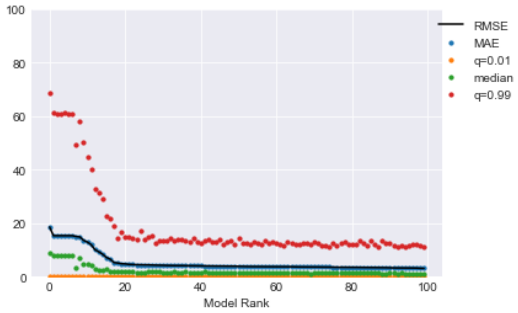
Figure 4.4: **Monte Carlo Error Distributions**.Deep models are ranked by performance on the combined fold validation set on the x axis. Average error metrics are shown as well as the 0.01 and 0.99 quantiles on the y axis.

(a) D1, PLS

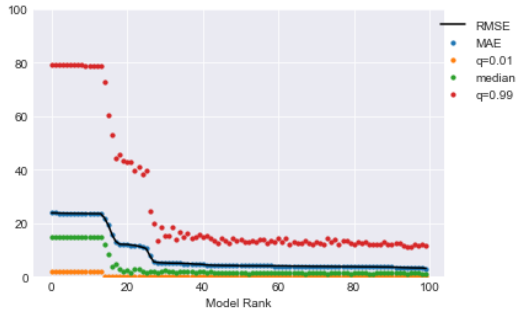
(b) D1, Std



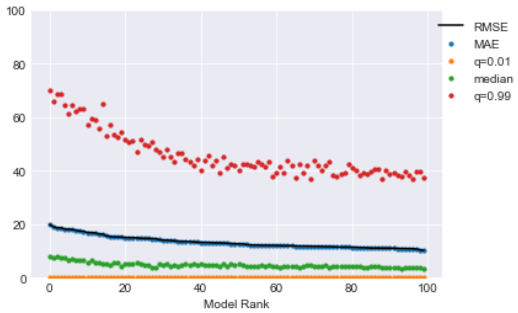
(c) D2, PLS



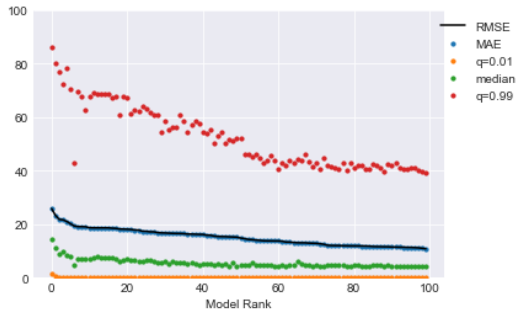
(d) D2, Std



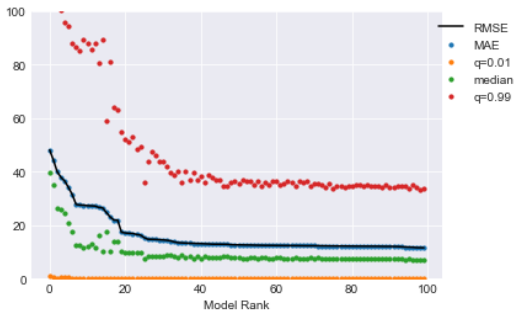
(e) D3, PLS



(f) D3, Std



(g) D4, PLS



(h) D4, Std

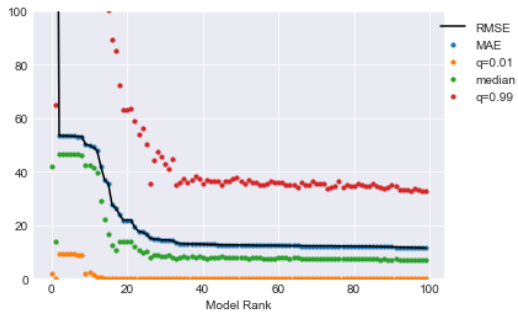
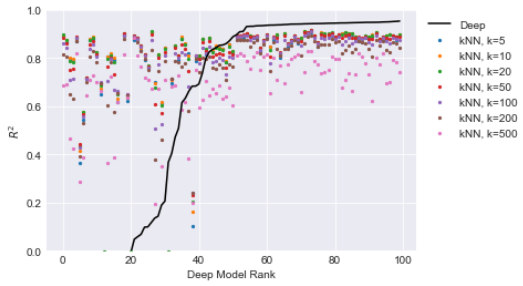


Table 4.3: **Cross Evaluation Results for Deep Networks**. Model numbers refer to the same model across rows to the ranking in each search rather than to a specific deep learner. We limit results to one lazy predictor per deep model. In bold are results for lazy predictors better than the best deep learner and in italics results for lazy predictors worse than the 5th best deep learner, for each given scheme.

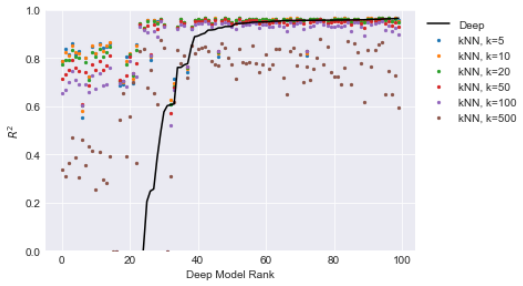
	D1		D2		D3		D4	
Model	Std	PLS	Std	PLS	Std	PLS	Std	PLS
Deep #1	0.9440	0.9538	0.9586	0.9585	0.6621	0.6937	0.7268	0.7200
Deep #2	0.9438	0.9526	0.9538	0.9575	0.6510	0.6884	0.7213	0.72
Deep #3	0.9428	0.9518	0.9536	0.9552	0.6390	0.6697	0.7176	0.7187
Deep #4	0.9416	0.9508	0.9529	0.9545	0.6345	0.6598	0.7172	0.7177
Deep #5	0.9411	0.9504	0.9522	0.9543	0.6301	0.6594	0.7144	0.7109
Deep-LR #1	**0.9470**	0.9552	**0.9641**	0.9607	**0.6697**	*0.6541*	0.7248	0.7185
Deep-LR #2	**0.9465**	0.9544	**0.9624**	0.9603	0.6601	*0.6288*	0.7216	0.7178
Deep-LR #3	**0.9463**	0.9542	**0.9606**	0.95881	0.6584	*0.6208*	0.7193	0.7168
Deep-LR #4	**0.9462**	0.9541	**0.9560**	0.9572	0.6532	*0.6162*	0.7187	*0.7104*
Deep-LR #5	**0.9460**	0.9539	**0.9593**	0.9572	0.6518	*0.6114*	0.7181	*0.7099*
Deep-kNN #1	**0.9692**	*0.9151*	**0.9620**	*0.9155*	0.6664	*0.5950*	**0.7409**	*0.5572*
Deep-kNN #2	**0.9683**	*0.9117*	**0.9602**	*0.9112*	0.6629	*0.5923*	**0.7395**	*0.5563*
Deep-kNN #3	**0.9679**	*0.9114*	**0.9595**	*0.9104*	0.6621	*0.5910*	**0.7393**	*0.5507*
Deep-kNN #4	**0.9674**	*0.9113*	0.9572	*0.9035*	0.6539	*0.5891*	**0.7381**	*0.5419*
Deep-kNN #5	**0.9667**	*0.9076*	0.9569	*0.9012*	0.6534	*0.5878*	**0.7375**	*0.5413*
Deep-LWR$_a$#1	**0.9483**	**0.9559**	**0.9674**	**0.9642**	**0.6823**	0.6858	0.7239	0.7151
Deep-LWR$_a$#2	**0.9469**	**0.9557**	**0.9668**	**0.9623**	**0.6789**	0.6787	0.7233	0.7142
Deep-LWR$_a$#3	**0.9465**	**0.9556**	**0.9667**	**0.96217**	**0.6685**	0.6639	0.7231	0.7139
Deep-LWR$_a$#4	**0.9464**	0.9546	**0.9666**	**0.9606**	**0.6676**	*0.6574*	0.7210	*0.7101*
Deep-LWR$_a$#5	**0.9464**	0.9546	**0.9666**	**0.9603**	**0.6668**	*0.6543*	0.7207	*0.7098*
Deep-LWR$_b$#1	**0.9481**	**0.9558**	**0.9628**	0.9585	0.6823	**0.6956**	0.7247	0.7155
Deep-LWR$_b$#2	**0.9467**	**0.9556**	**0.9624**	0.9581	0.6789	**0.6914**	0.7243	0.7149
Deep-LWR$_b$#3	**0.9464**	**0.9554**	**0.9612**	0.9579	0.6687	**0.6900**	0.7238	0.7144
Deep-LWR$_b$#4	**0.9462**	**0.9550**	**0.9609**	0.9575	0.6674	**0.6894**	0.7212	0.7120
Deep-LWR$_b$#5	**0.9462**	**0.9550**	**0.9605**	0.9574	0.6668	**0.6881**	0.7199	*0.7106*

Figure 4.5: **Cross-Validation Results for Deep-kNN**. Results for various values of k are compared to the performance of the standalone deep learners.

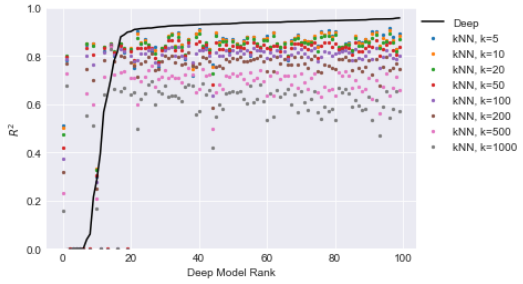
(a) D1, PLS



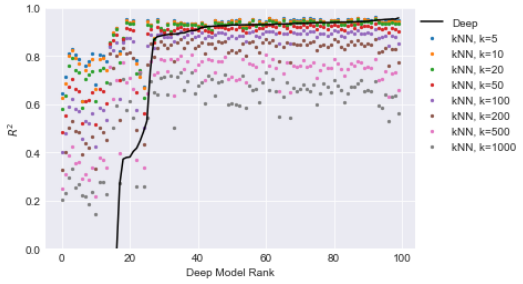
(b) D1, Std



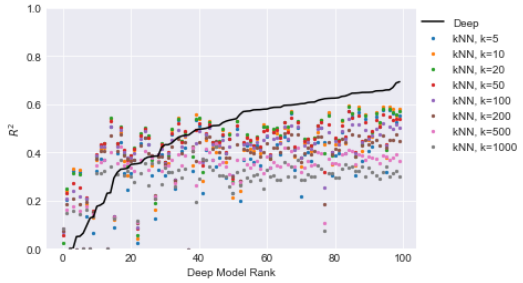
(c) D2, PLS



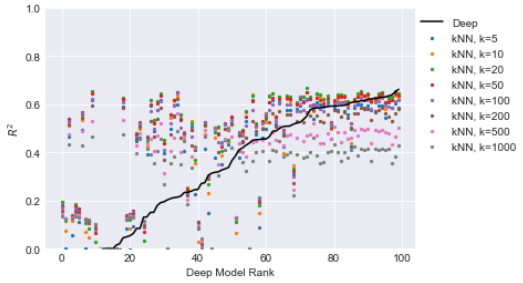
(d) D2, Std



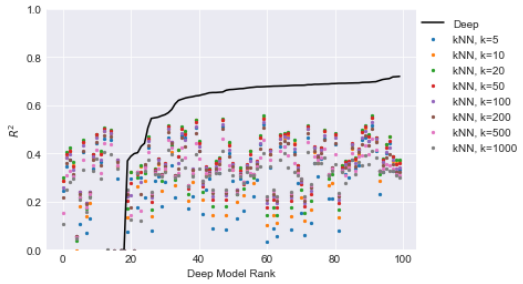
(e) D3, PLS



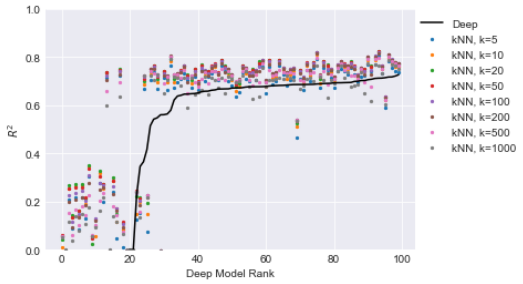
(f) D3, Std



(g) D4, PLS



(h) D4, Std



ranked models. While most LWR models perform well, a scattering of poorly performing models—with a wide range of k values—is observed.

The best LWR model is selected for each deep model, with the best five combinations being again compared to the best five deep models in Table 4.3 (denoted by Deep-LWR$_a$). Also included in these figures are results for linear regression predictors.

The best Deep-LR models are broadly in line with the best deep models. All of the results that are in bold use Std preprocessing, while all of the results in italics use PLS preprocessing. This suggests that the PLS models are better optimised than models with Std preprocessing, although the magnitude of this effect is negligible.

The best Deep-LWR models typically, but not always, represent a small level of improvement over the best Deep-LR models, with five columns displaying bold results compared to three with italics. This effect is partly due to D4 showing no improvements for LWR over the base deep networks.

4.1.5 Extrapolation

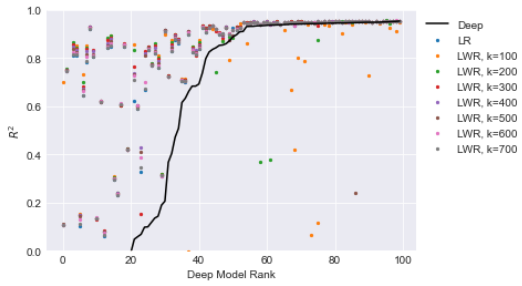
We earlier noted that a scattering of LWR models displayed a poor level of performance. One possibility for these observations is that locally weighted models are trained on a subset of the domain of the training set. Likely this increases the space of values for which they are forced to extrapolate. This theory is consistent with the observation that many poorly performing models have a relatively small number of neighbours ($k = 100$).

We experiment with a second variation of LWR models, denoted Deep-LWR$_b$, where predictions are bounded to the range of targets observed in each prediction neighbourhood.

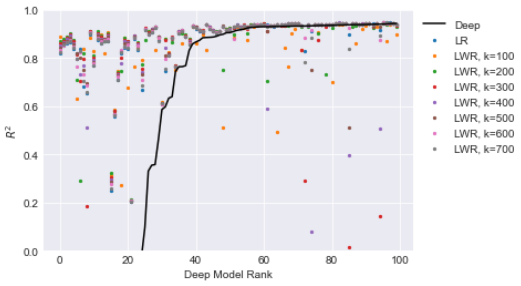
The deep models with bounded predictions perform slightly worse than the standard ones, suggesting this is a bias-variance trade-off; extrapolation allows LWR models to perform better for many predictions while occasionally performing much worse.

Figure 4.6: **Cross-Validation Results for Deep-LWR**. Results for various values of k are compared to the performance of the standalone deep learners.

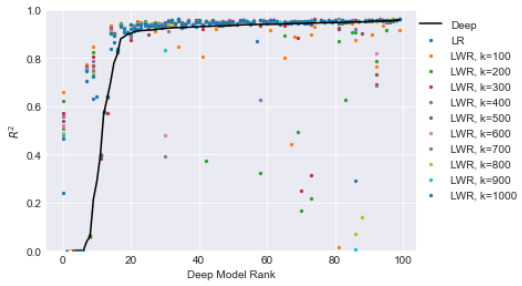
(a) D1, PLS



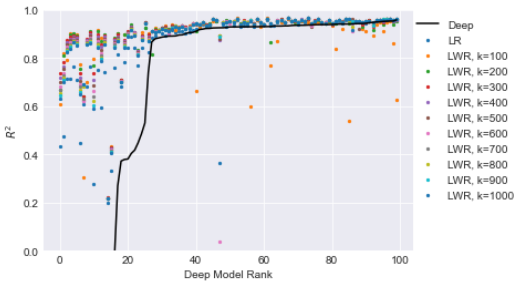
(b) D1, Std



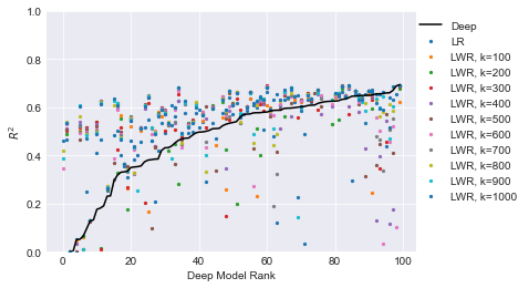
(c) D2, PLS



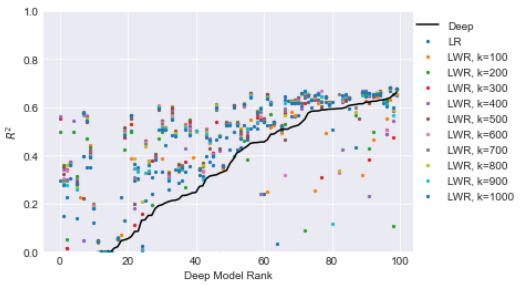
(d) D2, Std



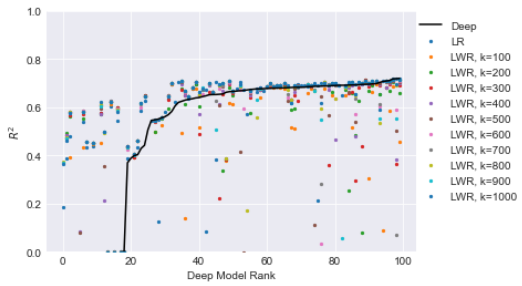
(e) D3, PLS



(f) D3, Std



(g) D4, PLS



(h) D4, Std

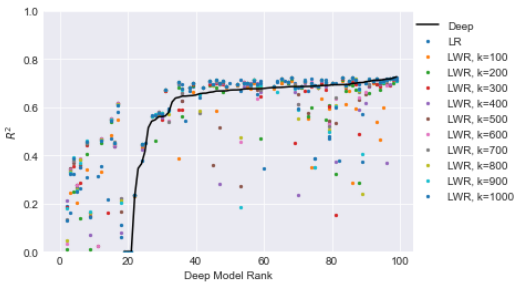


Table 4.4: **Test Set Results**. In each row, the best model based on cross-validation performance is selected for each dataset. The best result for each dataset is in bold.

	D1		D2		D3		D4	
Model	MSE	R^2	MSE	R^2	MSE	R^2	MSE	R^2
LR	47.1400	0.9413	30.0667	0.8987	259.6859	0.3854	197.2773	0.5895
PLS-LR	47.6376	0.9407	31.1087	0.8952	259.6013	0.3856	208.2680	0.5666
PLS-LWR	23.7224	0.9657	7.1299	0.9640	138.3838	0.6120	141.6336	0.6966
Std-Deep	23.0052	0.9680	10.6442	0.9552	117.8532	0.6417	135.5702	0.7183
PLS-Deep	**17.4872**	**0.9738**	8.3491	0.9646	112.6696	0.6575	**133.6739**	**0.7222**
Std-Deep-kNN	45.8021	0.9430	9.2070	0.9612	149.8450	0.5445	141.0961	0.7068
PLS-Deep-kNN	24.8357	0.9655	10.8747	0.9542	118.39612	0.6401	146.7114	0.69512
Std-Deep-LWR	22.1467	0.9692	**6.6378**	**0.97206**	**106.6613**	**0.6757**	139.08790	0.7110
PLS-Deep-LWR	17.9176	0.9751	7.7545	0.9674	113.8248	0.6540	138.1483	0.7129

4.1.6 Test Results

This section has conducted a range of experiments in a cross-validation setting. We now evaluate the performance of the approach on a previously held-out test set, with models trained on the complete set of data previously used for cross-validation.

Results for the best model for each predictor and preprocessing type, as evaluated during cross-validation are displayed in Table 4.4. D1 and D4 display their best results with a PLS-deep configuration, while the D2 and D3 datasets see their best results come from a Std-Deep-LWR configuration.

4.1.7 Generalisability

Having selected candidate models and evaluated their performance on a new set of data, we now analyse how the approaches generalised to this new data. LWR learners are compared to their respective base deep learners in Figure 4.7, with deep learners ranked by their scores on the cross-validation set. These figures show a level of noise for the deep learners, with performance on the test

set not always consistent with performance in cross-validation. The LR and LWR models display more stable performance, indicating that they improve their deep models' generalisability. When this is not the case, and LWR models perform poorly, the value of k is often small.

We further investigate the generalisability of the LWR models by showing scatterplots that compare cross-validation and test set performance (Figure 4.7). We again see, across all datasets, that LWR models (with a sufficient value of k) display more consistent performance on the test set.

This is also done for kNN models in figures 4.9 and 4.10. Many of the kNN models display a linear relationship between cross-validation and test performance, suggesting that they are providing a benefit in regularisation. This holds even for large values of k, where performance typically falls.

There is little overlap between cases where kNN models outperform standalone deep models on the training set and cases where they do the same on the test set. Looking across all kNN models, many outperform the best baseline deep model on the test set. However, when the best performing kNN model on the cross-validation set was selected, it performed poorly on the test set. Further study is required to understand this phenomenon.

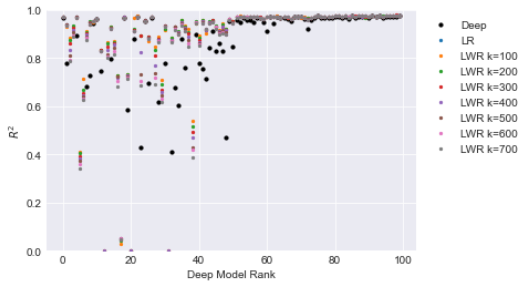
4.1.8 Summary

This section began by looking at classical techniques for spectral data. Neither PLS nor LWR techniques alone gave a good level of performance; however, once combined, they gave a much better baseline level of performance.

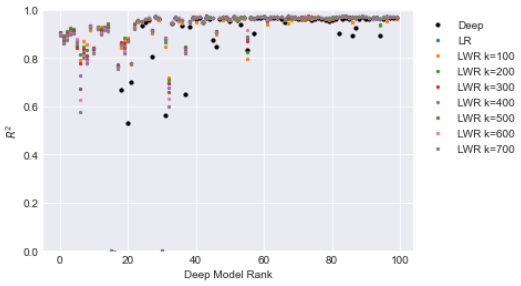
Deep models as selected by the Monte Carlo method improved on the PLS-LWR benchmark for all datasets. PLS preprocessing and dimension reduction consistently (but not always) gave better results than just standardising the data. Even though, in theory, this technique reduces the information input into a network and, therefore, the maximum level of performance, reducing parameter counts decreases the search space making optimisation easier for both gradient descent and Monte Carlo search.

Figure 4.7: **Comparing Deep-LWR to Standalone Deep Models on the Test Set**. Models are ranked based on cross-validation performance.

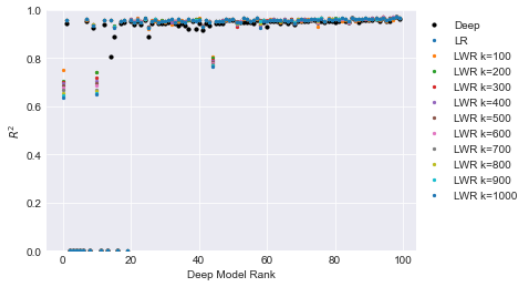
(a) D1, PLS



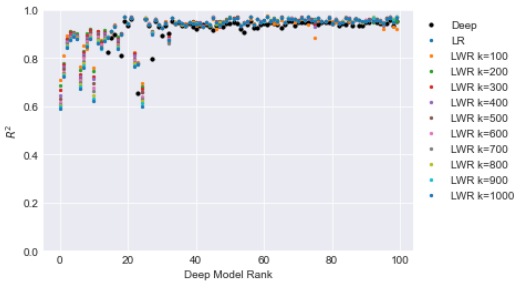
(b) D1, Std



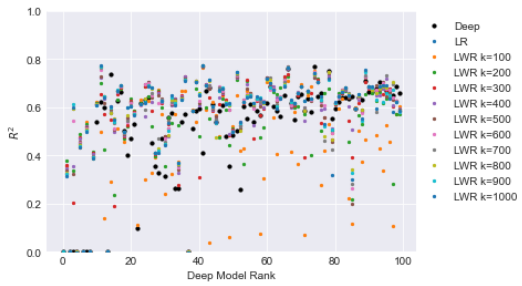
(c) D2, PLS



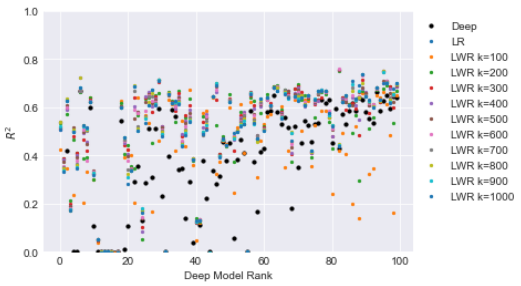
(d) D2, Std



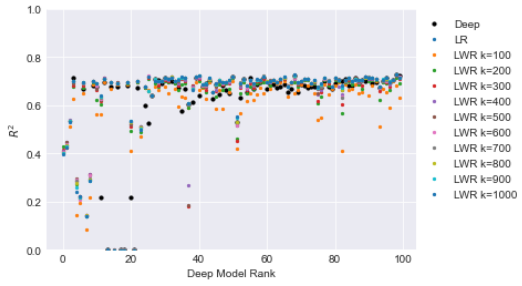
(e) D3, PLS



(f) D3, Std



(g) D4, PLS



(h) D4, Std

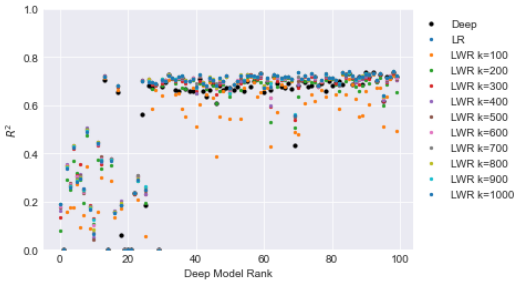
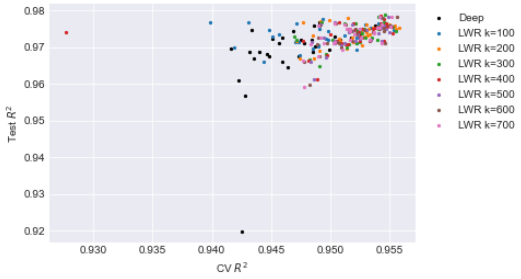
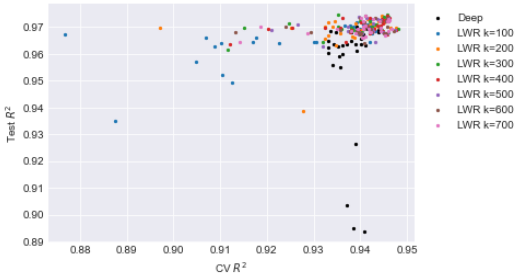


Figure 4.8: **Comparing Cross-Validation and Test Set Scores for Deep-LWR**. Training and test performance is compared for the 30 best performing models during training.

(a) D1, PLS

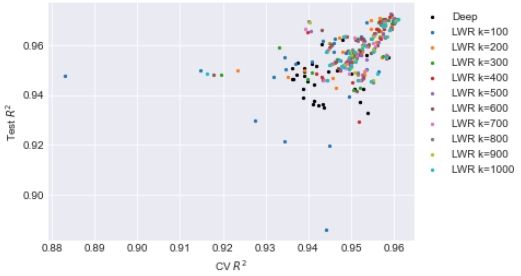


(b) D1, Std

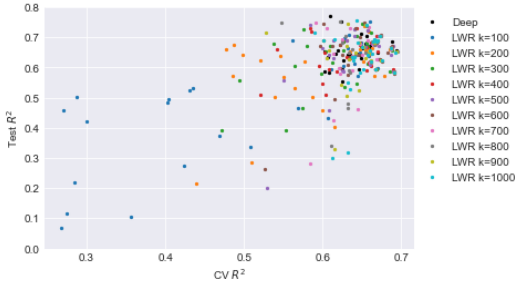


(c) D2, PLS

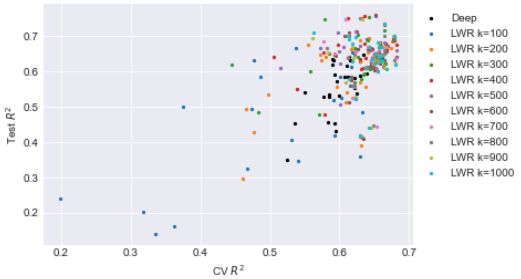
(d) D2, Std



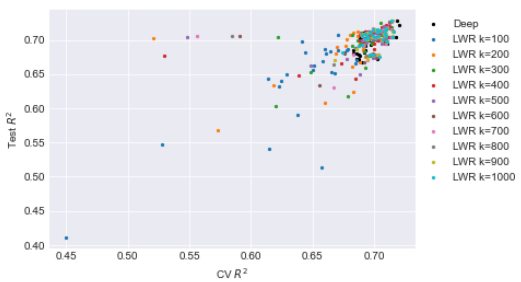
(e) D3, PLS



(f) D3, Std



(g) D4, PLS



(h) D4, Std

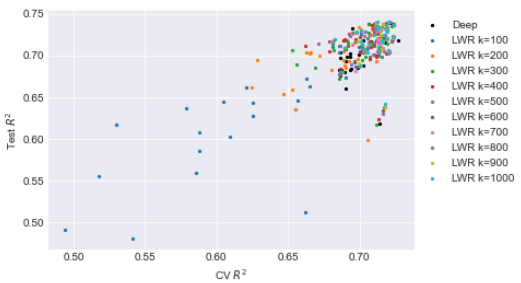
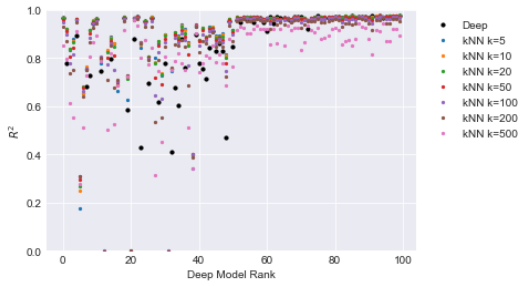
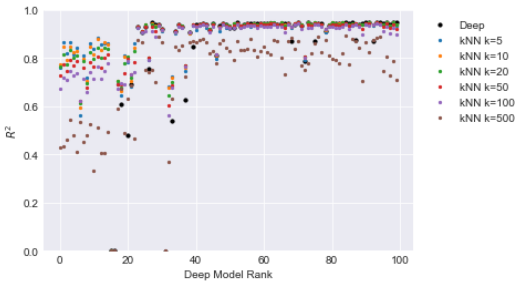


Figure 4.9: **Comparing Deep-LWR to Standalone Deep Models on the Test Set**. Models are ranked based on cross-validation performance.

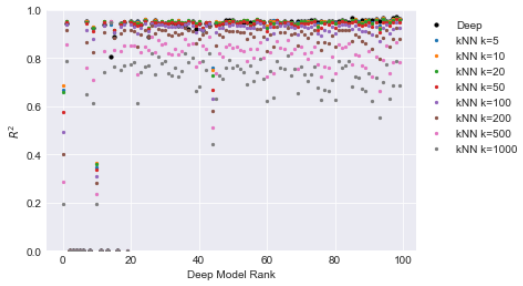
(a) D1, PLS



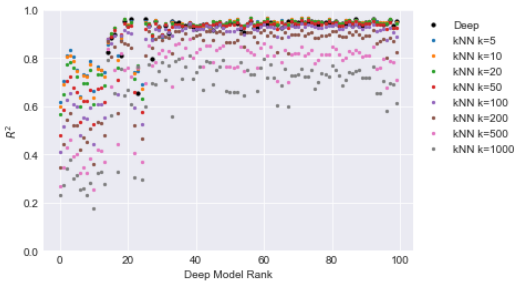
(b) D1, Std



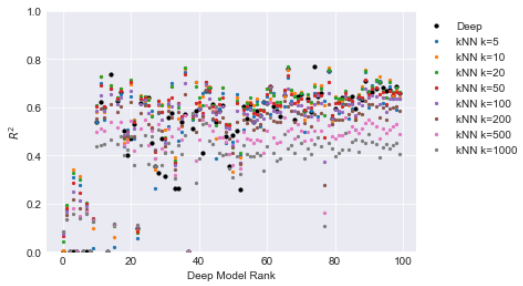
(c) D2 PLS



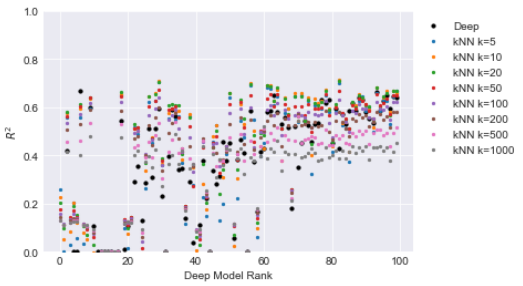
(d) D2, Std



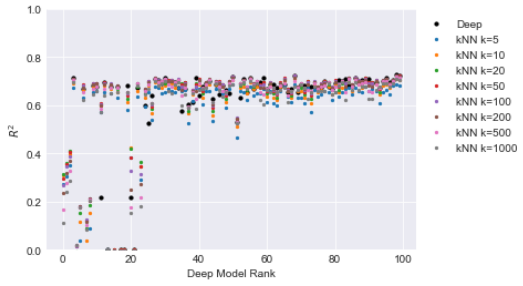
(e) D3, PLS



(f) D3, Std



(g) D4, PLS



(h) D4, Std

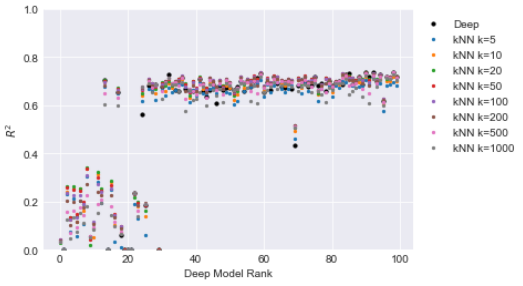
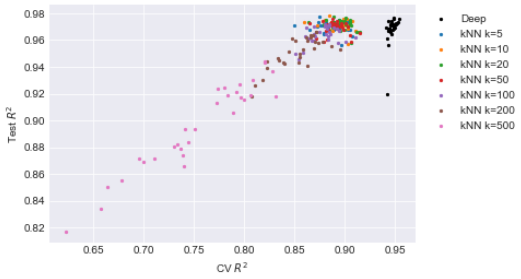
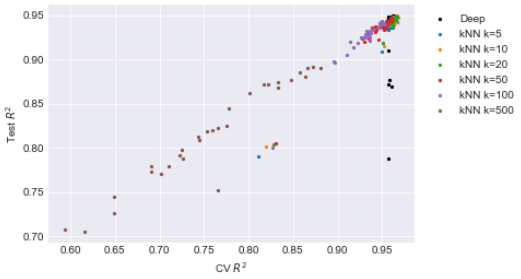


Figure 4.10: **Comparing Cross-Validation and Test Set Scores for kNN predictors**. Training and test performance is compared for the 30 best performing deep models during training.

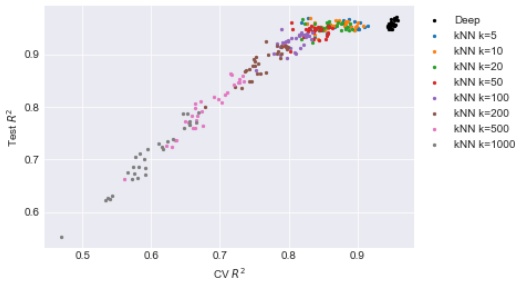
(a) D1, PLS



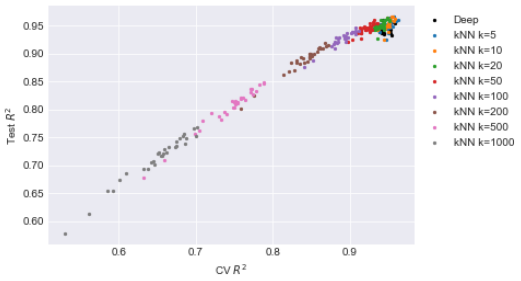
(b) D1, Std



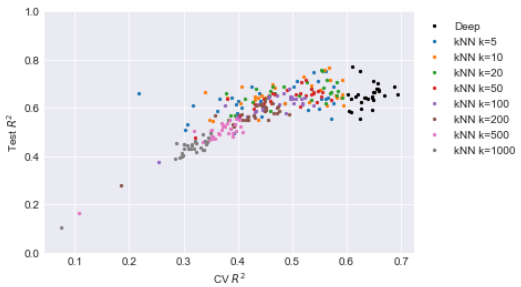
(c) D2, PLS



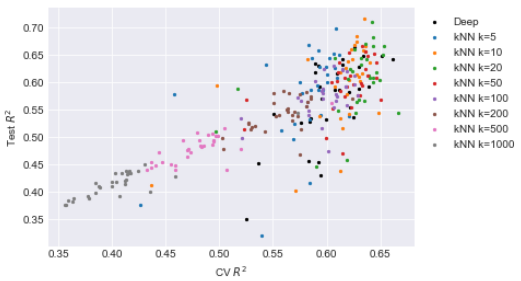
(d) D2, Std



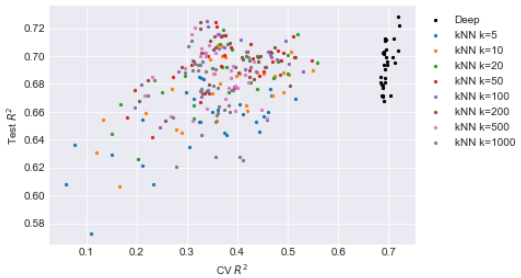
(e) D3, PLS



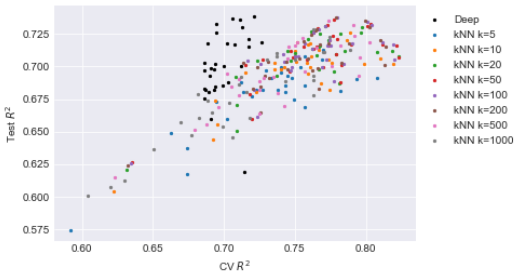
(f) D3, Std



(g) D4, PLS



(h) D4, Std



The use of alternative lazy predictors saw improvements in performance over baseline deep models in many of the dataset and preprocessing cases. Performance improvements were much more substantial across the set of searched models, with LWR models tending to improve the ability of a model to carry over performance from cross-validation training to the test set. In contrast, kNN models, while often displaying good performance, showed much less of a correlation between performance in cross-validation and performance on the test set.

In Chapter 3, we discussed two cases in which locally weighted regression can benefit a deep model. Some of the performance benefits seen from LWR models are also observed for LR predictors. A plausible explanation for this is that for many of the deep models, the head layer is not fully optimised; performance gains can be made by training these models with further epochs. Alternatively, the performance gains seen for both LR and LWR techniques may be due to extra regularisation offered by a (conditionally) independent learner.

4.2 Mangoes Experiments

While the other datasets investigated in this study are propriety, the Mangoes dataset is a public domain NIR dataset for which the methodology can be evaluated. The Mangoes dataset measures the dry matter content (DM/DMC) of green mango fruit at harvest time, which can be a useful proxy of the quality of fruit once ripened.

4.2.1 Dataset

The dataset consists of 11835 spectra spanning the 300-1100nm wavelength range. These spectra correspond to 4685 distinct fruit. Each fruit has two readings present in the dataset, with a small subset containing extra readings taken at different temperatures.

The dataset is built to be robust to variation in fruit. Samples are taken across 4 growing seasons, 10 cultivars and 2 growing regions. Also marked are the temperature and physiological stage (whether the fruit is hard green or ripened).

The authors provide training and validation sets. Data from the 4th season is reserved as the test set, and the remaining data (seasons 1-3) is split by population into calibration and tuning sets using the Kennard Stone algorithm at a 70-30 ratio. 143 spectra were detected as outliers and were removed from the dataset before being made publicly available.

The published version of the dataset consists of 11616 spectra, of which 7413 spectra (2864 fruit) belong to the calibration set, 2830 spectra (1086 fruit) to the tuning set and 1448 spectra (725 fruit) to the test set.

4.2.2 Background

Anderson et al. released two papers, the first looking at Global Partial Least Squares models [4], the second looking at Local PLS models and non-linear models [2].

Broadly speaking, the first Anderson et al. experiments consist of four main parts, outlier removing, preprocessing selection, wavelength selection, and building final models for both the whole data set and for specific subsets of the data set.

The authors built a mean-centred PLSR model with seven components using a first derivative Savitsky-Golay filter with a second-order polynomial fitted to nine data points based on previous research (f"Poly2Point9). This is done over the 729-975nm wavelength for the entire dataset. Using this model, they calculate GH, a distance metric calculated as Mahalanobis distance divided by the number of latent variables, in this case, seven, and removed 143 variables with a GH value greater than twelve. The remaining data was divided into calibration, tuning, and testing sets.

The authors then looked at various pre-treatments based on Savitsky-Golay

filtering, training models on the 729-975 wavelength range of the calibration set and testing on the tuning set. The authors established f'Poly2Point3 and f"Poly2Point17 Savitsky-Golay as well-performing pretreatments for mean-centred partial least squares models.

They then go on to optimise wavelength. This is done using Leave-One-Out-Population (LOOP) cross-validation over data from the first three seasons. They test wavelengths over the range 513-1050nm with a 9nm step. Models are evaluated with the aforementioned pre-treatments, with LVs selected such that no LV can be added that explains at least 1% of the $\mathbf{y}$ variance. They establish the best model on the 684-990nm range with f"Poly2Point17 Savitisky-Golay preprocessing and a mean-centred PLSR with eleven latent variances, which gives an RMSE of 0.86 and an R^2 of 0.84.

In addition to models trained on the combined calibration and tuning sets, specific models are built for the physiological stage (Ripened or Green). Variations of these models are then used to report results on the test set. Specific models are also built by cultivar using leave one out population cross-validation, where for each population present in the test set, models are trained on all other data.

These results are mostly calculated with the Unscrambler package (see for example [10], which performs model selection. For instance, the number of latent variables changes across both variables for local models and cross-validation folds, rather than being a constant 11.

Anderson et al. in [2] extended this work by looking at locally weighted regression and other non-linear models, including SVM kernel models and Gaussian-process regression. Various locally weighted methods were used and are discussed previously in Chapter 2.4. LOCAL was implemented with both a set number of latent variables and as ensembles of models with varying numbers of latent variables. LPLS and LPLS-S were also implemented, with the number of latent variables LV = {10, 12, ..., 40} selected by grid search. Another local method was used, with neighbourhoods calculated by Mahalanobis distance in

the space of PLS transformed features. Latent variables for PLS are selected by measuring decay in y variance rather than model performance to increase computational speed.

Using partial least squares with 600 neighbours and 14 components gave an RMSEP of 0.887 (MSEP of 0.787, R^2 of 0.903), with an ensemble method giving an R^2 of 0.904.

4.2.3 Related Work on Mangoes

Mishra and Passos in [41], look at combining domain-specific techniques with deep networks. They use PLS to prefilter the training set, identifying several outliers that they remove. They investigate preprocessing with standard normal variate schemes as a standalone technique and combined with 1st and 2nd derivatives. Features from each preprocessing scheme are stacked, giving an augmented data set with five times the number of features. The deep network consists of a single 1D convolution layer, followed by dense layers of sizes 36, 18, and 12. eLU activation functions are used, and l_2 regularisation is applied to the linear layers. Models are trained using the Adam optimiser for a maximum of 750 epochs, with a reduce-lr-on-plateau rule. This results in an RMSEP of 0.79 for the Mangoes test set. As a comparison, their best PLS method trained on augmented data gave an RMSEP of 0.95.

As a follow-up, Mishra and Passos in [41], developed an independent set of data to evaluate models previously trained on the Mangoes dataset. They find that on this set, their DL model overfits (sees reduced performance), while the PLS model is more robust to overfitting. They retrain their deep model on the entire Mangoes data set, using the test set as a validation set for hyperparameter search. They search 210 models consisting of variation in convolution width, l_2 regularisation and batch size. They also built a PLS model, selected by 10-fold cross-validation on the whole Mangoes data set. When these models are tested on their new set of data, the best deep model outperforms PLS; it generalises better to previously unseen data.

4.2.4 Reproducing Results

As part of validating the approach, we reproduce Table 1 and Table 4 from Anderson et al., which show results for various preprocessing schemes on the tuning set and results for the best models on the test set, respectively.

The implementation uses the python package SKLearn, for which PLSR is only available in standardised and non-standardised variants, with no mean-centred variant available. We also reproduce these tables in Weka for further validation purposes, using an explicit mean-centring PLSR implementation available in this library.

Results comparing the preprocessing approaches are shown in Table 4.5. Models are trained on the calibration set and testing on the tuning set. We see identical results for ABS raw preprocessing across all implementations; however, for Savitsky-Golay preprocessing, the SKLearn PLSR fails to see the performance gains observed by the original authors.

We then compare the results for the best schemes in Table 4.6. In addition to results for a model trained on the combined calibration and tuning data and tested on the testing data, we show cross-validated results and stage-specific models. The models here are PLSR with *f*"Poly2Point17 on the 684-900nm wavelength range. The number of components is 11 for all of the reproduced results and may vary for results from Anderson et al.

The majority of the reproduced results show performance just below that of the original paper. We see this for both SKLearn and Weka implementations of PLS. Part of this difference is likely due to implicit model selection in the Unscrambler. While the results are not perfectly aligned, they are close enough to give us confidence in further comparisons to public benchmarks for the Mangoes data set.

4.2.5 Experimental Results

We repeat the batch experiments from Chapter 4.1 for the Mangoes dataset. As an initial exercise, we investigate the selection of wavelength ranges. Three

Table 4.5: Reproducing Savitsky Golay Preprocessing Results

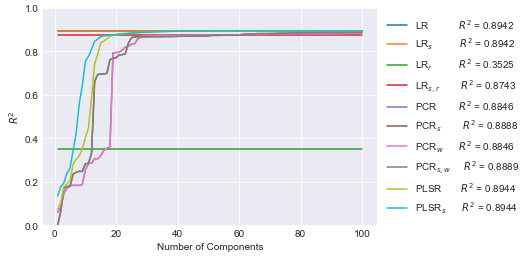
		Original		SKLearn		Weka	
Preprocesssing	#LV	R^2	RMSE	R^2	RMSE	R^2	RMSE
ABS raw	8	**0.80**	**0.96**	0.7952	0.9598	**0.8023**	**0.9588**
f'Poly2Point3	7	**0.84**	**0.87**	0.8041	0.9386	0.8288	0.8934
f"Poly2Point9	10	0.79	0.98	0.8048	0.9369	**0.8266**	**0.8881**
f"Poly2Point17	9	**0.83**	**0.87**	0.8056	0.9350	0.8187	0.9128

Table 4.6: **Reproducing Mangoes Test Set Results**

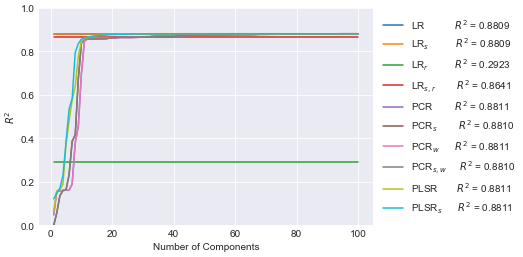
			Original		SKLearn		Weka	
# LV	Train set	Test set	R^2	RMSE	$R2$	RMSE	R^2	RMSE
11	all	Test-all	**0.86**	**1.01**	0.85	1.03	0.8493	1.0453
$*$	stage specific	Test-by stage*	**0.89**	**0.88**	0.88	0.91	-	-
$*$	specific*	Test-by cv*	0.89	**0.88**	0.88	0.91	-	-
11	all	hard green	**0.86**	**1.01**	0.84	1.06	-	-
11	hard green	hard **green**	**0.86**	1.01	0.84	1.05	0.8476	1.0734
12	ripening	hard green	**0.89**	**0.91**	0.86	0.97	-	-
11	all	ripening	**0.85**	**1.02**	0.83	1.02	-	-
12	ripening	ripening	**0.91**	**0.78**	0.90	0.80	0.902	0.7958
11	hard green	ripening	0.79	1.16	**0.83**	**1.04**	-	-

Figure 4.11: **Cross-Validating Classical Approaches on the Mangoes**. We show results for each wavelength in figures a-c and compare all three for a PLSR model in d.

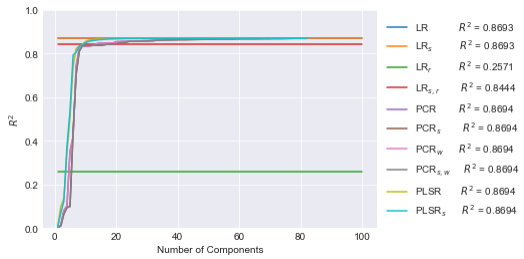
(a) 513-1050nm



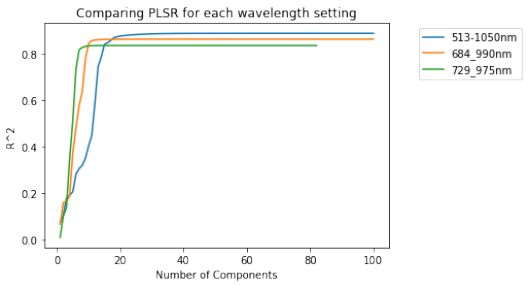
(b) 684-990nm



(c) 729-990nm



(d) Comparison



ranges are compared, 513-1050nm; the broadest possible range for which complete data is available, 729-975nm; used for preliminary experiments in Anderson et al. and 684-990nm; the wavelength settled upon for final experiments in Anderson et al.

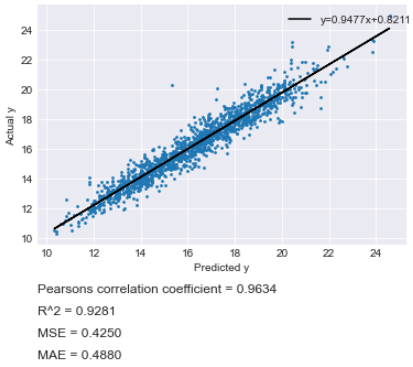
Trends are broadly consistent across all three wavelengths (Figure 4.11); normed linear regression gives a good level of performance, with PLS-LR and PCA-LR models converging to this level as the number of latent variables increases. Taking a smaller wavelength makes learning easier at the cost of performance; PLSR and PCA models require a smaller number of latent variables to converge but converge to a lower point.

For reproducibility and comparison purposes, we stick to the 684-990nm range for Mangoes in future experiments while noting that better performance will be possible for models trained on broader wavelengths of data.

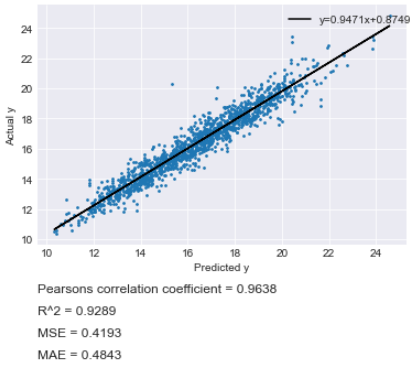
We present results for Mangoes on the 684-990nm wavelength as per the

Figure 4.12: **Predictions for Mangoes Test Set**.

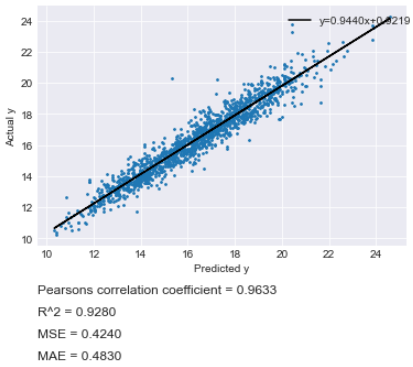
(a) Deep



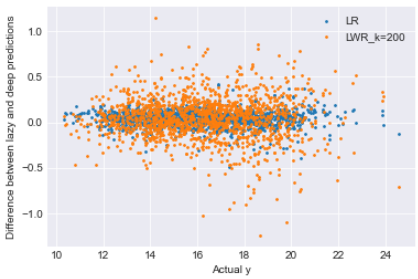
(b) Deep-LR



(c) Deep-LWR



(d) Comparing Lazy to Deep predictions



previous experiments (Chapter 4.1) in Table 4.7. The best-performing models during cross-validation are selected. Performance is reported for cross-validation and on the test set and compared to results from previous work. We observe similar scores across both sets, indicating that the cross-validation approach has been effective at preventing over-fitting.

Anderson et al. set a benchmark MSE of 1.02 for a global PLS model and 0.787 for a local PLS model. The PLS-LR, selected to have more latent variables, gave an MSE of 0.7114.

The best deep approach (PLS-Deep-LR) gave an MSE of 0.4240 on the test set, better than any result achieved by Anderson et al. and the 0.5776 achieved by Mishra and Passos. Both LR and LWR lazy predictors outperformed the base deep network.

Table 4.7: Mangoes Results

Model	Cross-Validation		Test Set	
	MSE	R^2	MSE	R^2
Anderson et al. Global PLSR	-	-	1.0200	0.8600
Anderson et al. LPLS	-	-	0.787	0.9030
Anderson et al. Ensemble ANN	-	-	0.7039	0.904
Mishra & Passos Deep	-	-	0.5776	-
PLS-LR	0.7221	0.8811	0.7123	0.8785
PLS-LWR	0.5788	0.9047	0.5785	0.9014
Std-Deep	0.7014	0.8838	0.6820	0.8867
PLS-Deep	0.4272	0.9297	0.4250	0.9275
Std-Deep-kNN	0.5583	0.9075	0.5039	0.9163
PLS-Deep-kNN	0.5143	0.9153	0.6693	0.8858
Std-Deep-LR	0.6665	0.8896	0.6641	0.8897
PLS-Deep-LR	0.4291	0.9293	**0.4193**	**0.9285**
Std-Deep-LWR	0.6589	0.8908	0.6259	0.8960
PLS-Deep-LWR	**0.4256**	**0.9299**	0.4240	0.9277

4.3 Streaming Experiments

Streaming experiments are evaluated on the D4 dataset due to its abundance of data. 100,000 instances are sampled for evaluation. We first look at the performance of fixed batch trained models and their purely incremental equivalents (Section 4.3.1). We then look at lazy incremental predictors for fixed deep networks in Section 4.3.2 and investigate ensemble approaches in Sections 4.3.3 and 4.3.4. Finally, we test the best performing models from these experiments on a fresh set of 100,000 instances in Section 4.3.5.

4.3.1 Preliminaries

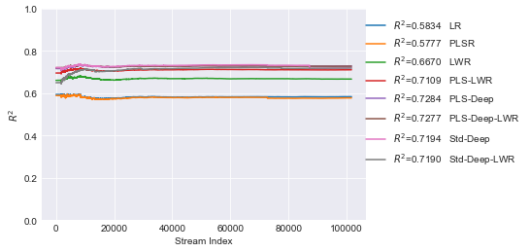
The starting point for these experiments is to evaluate the performance of models previously trained in a batch setting across the designated streaming interval. LR, LWR, Deep and Deep-LWR are evaluated, with models and hyperparameters selected based on batch performance. As for previous experiments, both Std and PLS preprocessing options were used.

We present these results as performance up to each point (Figure 4.13 a) and rolling performance over a window of size 1000 (Figure 4.13 b). We see consistent total performance across the evaluation window, indicating no concept drift. We observe several noticeable negative spikes in rolling performance for PLS-LR at index 75,000 and std-deep at index 87,217, but these are insufficient to degrade total performance. Consistent with the batch results, classical LR models perform poorly, with an R^2 of just below 0.6. LWR models perform better, with an R^2 score of just below 0.7, but are outperformed by the four deep models, which are closely clustered together just below 0.75. These results are consistent with those from the batch setting.

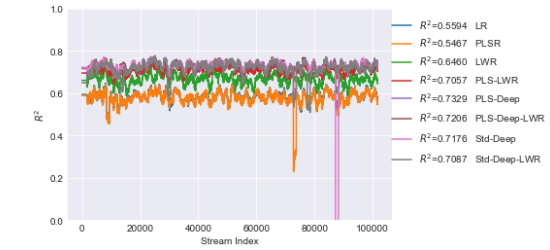
The next set of experiments evaluates incremental learned streaming models. Considered are LR, LWR and kNN models. Results are shown for total performance in (Figure 4.13 c) and rolling performance over a window of size 1000 (Figure 4.13 d). The kNN models, non-standardised LWR, and ridge LR

Figure 4.13: **Performance of Streaming Preliminaries.** Performance values at the end of the streaming evaluation are included in the legend.

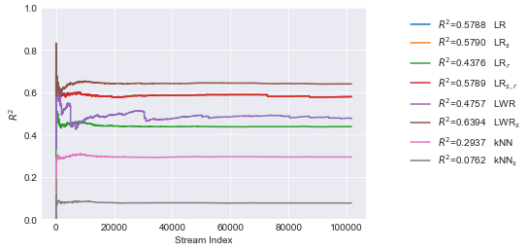
(a) Batch Models



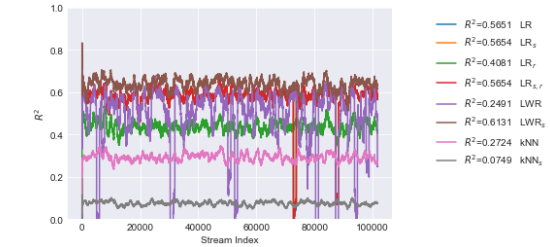
(b) Batch Models, rolling average



(c) Streaming Models



(d) Streaming Models, rolling average



models underperform the default LR and standardised LWR. A substantial amount of noise is observed in the rolling average predictions for the non-standardised LWR, with performance for other approaches being more stable.

4.3.2 Hybrid Streaming Models

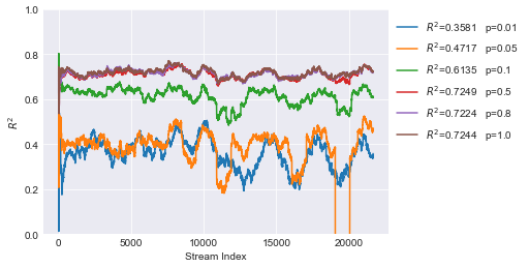
With purely batch and purely streaming models evaluated, we now combine deep network feature extractors, as trained in the previous batch experiment section, with streaming versions of the lazy alternative predictors.

Hyperparameters are selected, with results for a broad grid search given for kNN (Figure 4.15) and LWR (Figure 4.14). We find that a window size of 10,000 instances is sufficient, with diminishing returns combined with increased computation cost beyond this point. We selected 1000 neighbours for LWR models for this window size and 20 for kNN models.

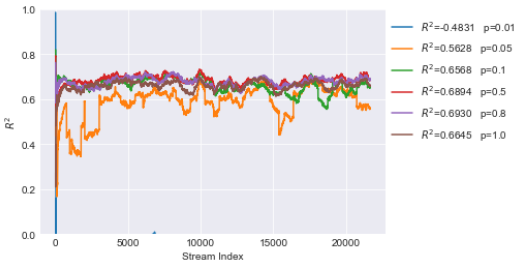
As for the batch experiments, we look at the best five deep models for each dataset and type of preprocessing. Results are shown in Figure 4.16 for std

Figure 4.14: **LWR Hyperparameter Search**. A grid search is performed over sliding windows of size 1000, 1000, and 20000 with the number of neighbours being a proportion of the data, $p = \frac{k}{w} = \{1, 0.8, 0.5, 0.1, 0.05, 0.01\}$. Performance values at the end of the streaming evaluation are included in the legend.

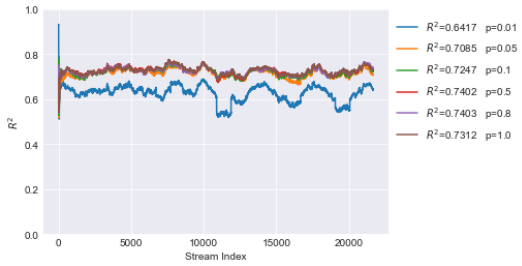
(a) PLS, Window size = 1000



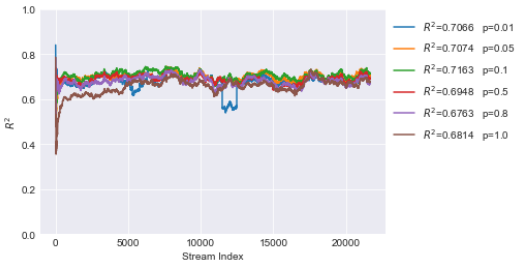
(b) Std, Window size = 1000



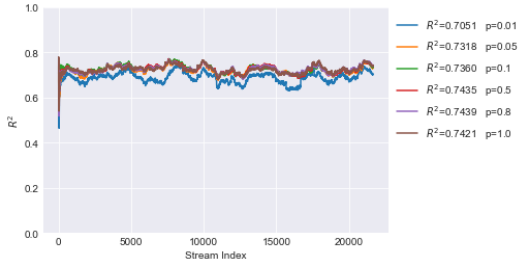
(c) PLS, Window size = 10000



(d) Std, Window size = 10000



(e) PLS,window size = 20000



(f) Std,window size = 20000

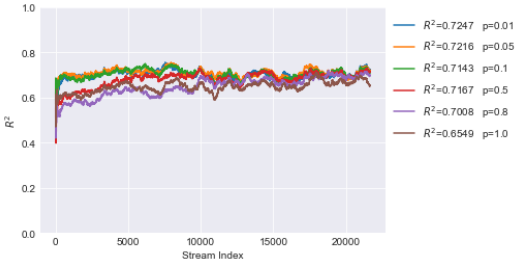
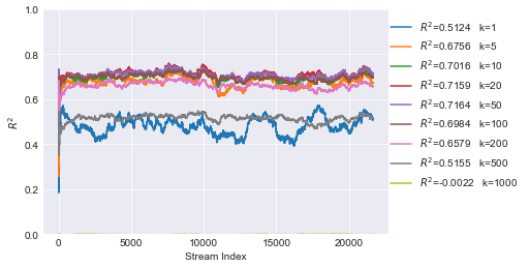
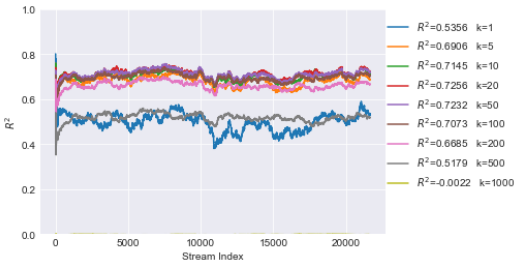


Figure 4.15: **kNN Hyperparameter Search**. A grid search is performed over sliding windows of size 1000, 1000, and 20000 with the number of neighbours $k = \{1, 5, 10, 20, 50, 100, 200, 500, 1000\}$. Performance values at the end of the streaming evaluation are included in the legend.

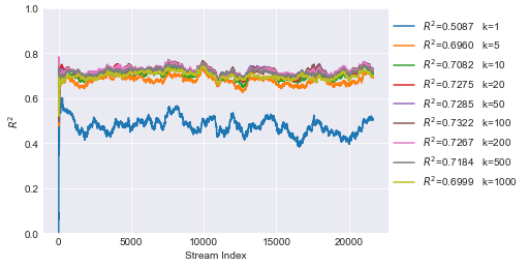
(a) PLS, Window size = 1000



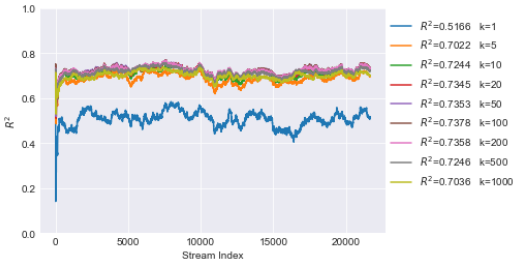
(b) Std, Window size = 1000



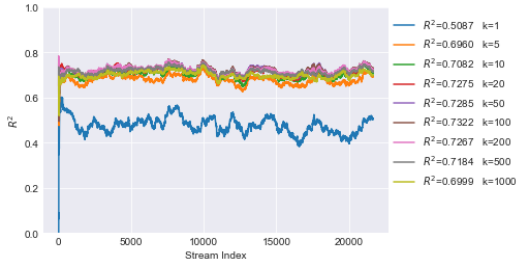
(c) PLS, Window size = 10000,rolling average



(d) Std, Window size = 10000,rolling average



(e) PLS,window size = 20000, rolling average



(f) Std,window size = 20000, rolling average

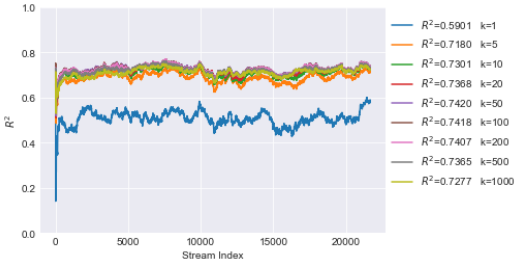
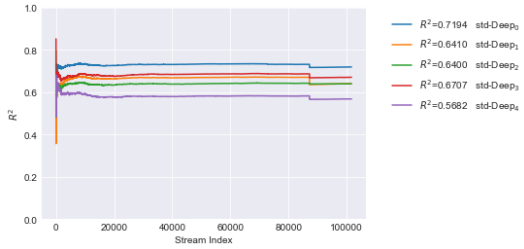
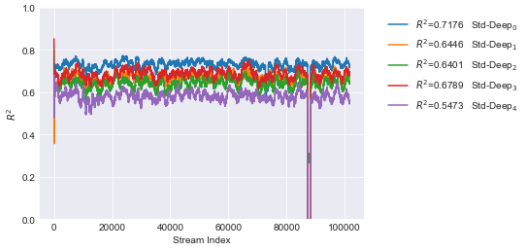


Figure 4.16: **Hybrid Streaming models with Std Preprocessing**.Performance values at the end of the streaming evaluation are included in the legend.

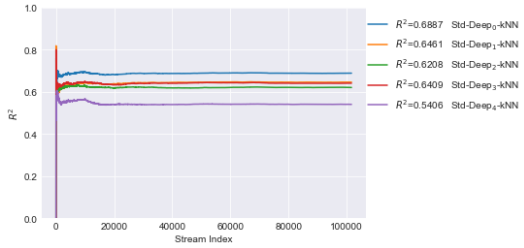
(a) Std-Deep



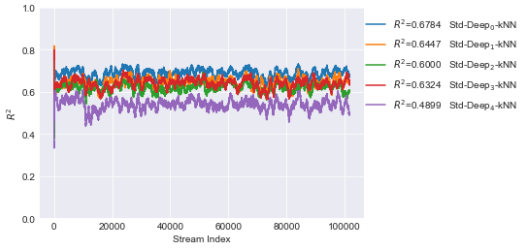
(b) Std-Deep, Rolling



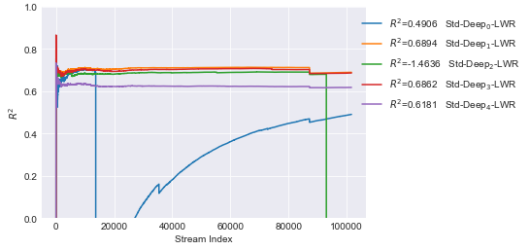
(c) Std-Deep-kNN



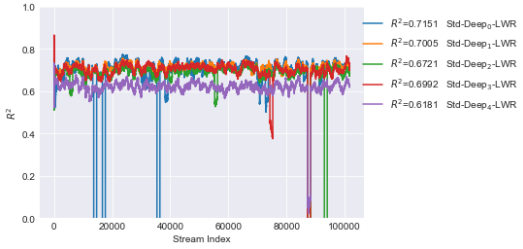
(d) Std-Deep-kNN, Rolling



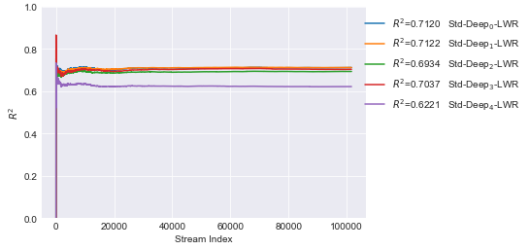
(e) Std-Deep-LWR$_a$



(f) Std-Deep-LWR$_a$, *Rolling*



(g) Std-Deep-LWR$_b$



(h) Std-Deep-LWR$_b$, *Rolling*

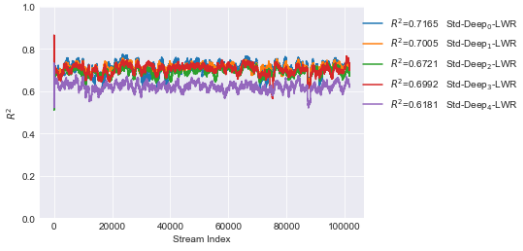
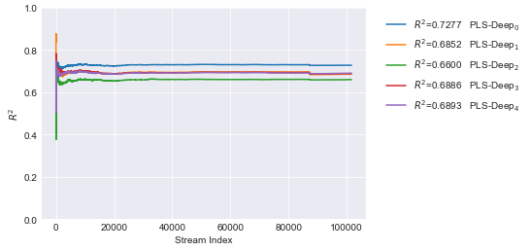
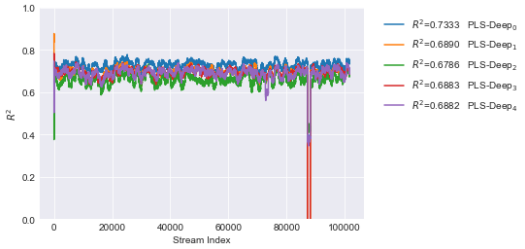


Figure 4.17: **Hybrid Streaming models with PLS Preprocessing**.Performance values at the end of the streaming evaluation are included in the legend.

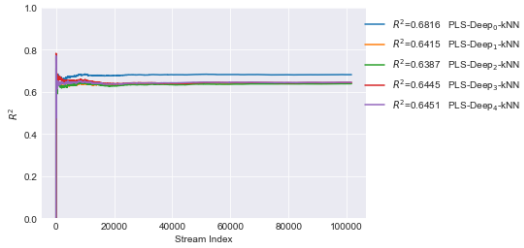
(a) PLS-Deep



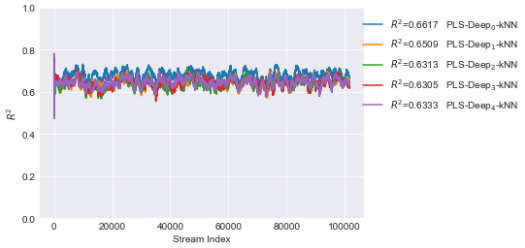
(b) PLS-Deep, Rolling



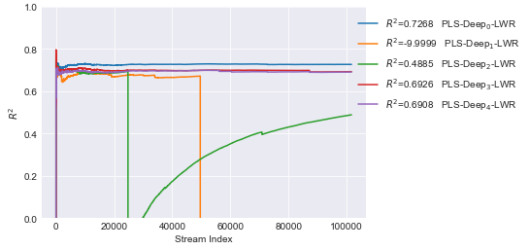
(c) PLS-Deep-kNN



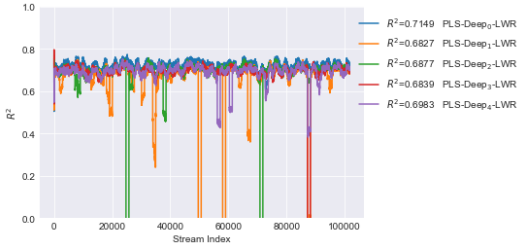
(d) PLS-Deep-kNN, Rolling



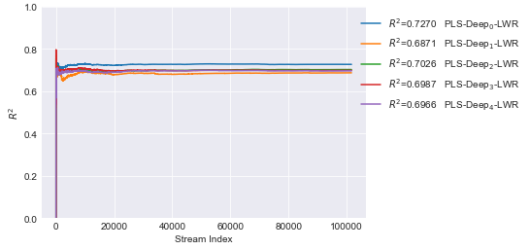
(e) PLS-Deep-LWR$_a$



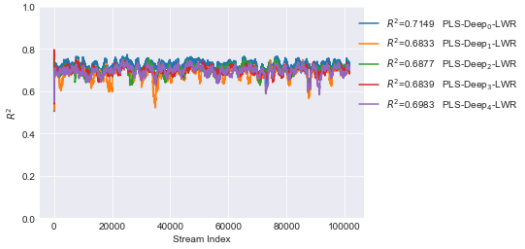
(f) PLS-Deep-LWR$_a$, *Rolling*



(g) PLS-Deep-LWR$_b$



(h) PLS-Deep-LWR$_b$, *Rolling*



preprocessing and Figure 4.17 for PLS preprocessing.

The base deep models maintain performance throughout the stream. Performance mostly follows the ordinal ranking, with PLS preprocessed models grouped closer than models with Std preprocessing. An outlier is again observed at index 87,217. In this instance, there is a slight dip in total results with a much greater dip in the sliding window results.

kNN models display smooth performance, including not being affected by the outlier common to other models. However, R^2 scores are lower than for other approaches.

While many of the Deep-LWR$_a$ models display consistent performance, others display severe prediction outliers sufficient to decrease total performance. These outliers are observed for both PLS and Std preprocessing. These outliers are not observed for the Deep-LWR$_b$ models (models were $\hat{y}$ is bounded), suggesting that these outliers in performance are due to out of domain effects.

4.3.3 Ensembles

So far, the streaming experiments have established that infrequent but large errors degrade the performance of LWR models, likely caused by out-of-domain predictions. The solution established in the batch setting, restricting the range of predictions, slightly worsens performance. In theory, being able to extrapolate to unseen before data is a driver of model performance.

As an alternative, we investigate ensembles with median voting rules. Ensembles are a general technique; combining several models can reduce bias or variance compared to any single model.

We experiment with two types of ensembles. The first approach ensemble heterogeneous models (diversity in models ensembles). We ensemble the five best deep models (based on cross-validation performance in Chapter 4.1) with and without LWR.

The second approach takes homogeneous models and creates diversity through data permutations. There are several standard methods for creating diversity

in data; bagging (or subsampling), subspaces (feature sampling), streaming random patches (combining subsampling and subspaces [22]) and input smearing (adding Gaussian noise).

In preliminary experiments, we ruled out feature sampling methods (subspaces and random patches) as these interfered with the implicit feature selection mechanisms of the deep models. Input smearing also performed poorly, leaving bagging as an ensemble technique for these experiments.

The streaming approximation of bagging takes a single parameter $\lambda = 1$ and samples instance weights from a Poisson distribution with mean *lambda*. A λ value of 1 results in each instance being included once per model on average, with approximately 36% of instances not included in any one model.

We investigate two variants of bagging ensemble, one with weights as sampled and the other with weights sampled and then limited to belong to {0,1}. We also investigate the difference between five models in an ensemble (to be consistent with the heterogeneous model ensembles) and three (the minimum number needed for median voting). Included is an arbitrary larger ensemble with 11 models to see if this had any impact. Results for this grid search are presented in (Figure 4.18). No benefits are observed for larger ensembles. Consequently, the minimum value of three is selected as the computational cost is proportional to ensemble size.

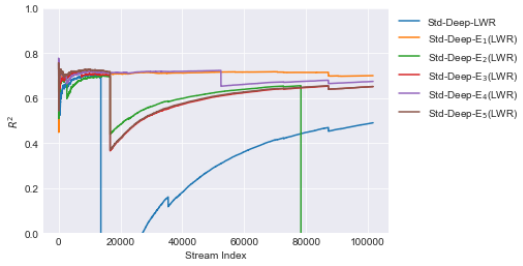
The two approaches of ensembles are compared by building a bagging based ensemble for each of the best five deep models and comparing these to heterogeneous ensembles of the best five deep models, both with and without LWR (Figure 4.19).

The heterogeneous approaches give the best performance with high rolling R^2 performance and only slight variance across the evaluation window.

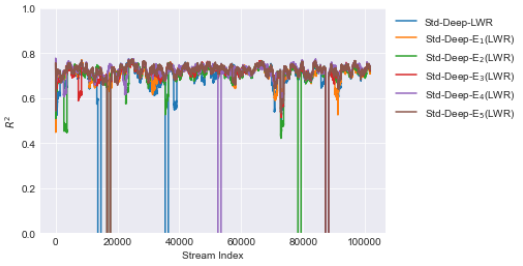
The bagging based ensembles mostly improve upon the base models; however, several ensembles still give erroneous predictions, although to a lesser degree than the non-ensembled versions. A takeaway here is that it is important to select deep models that perform well as feature extractors, for which

Figure 4.18: **Ensemble Hyperparameters**. E_1 is an ensemble of size 3, with values sampled 0 or 1 times. E_2 is an ensemble of size 3 with no limit on the maximum number of samples. E_3 is an ensemble of size 5, with values sampled 0 or 1 times. E_4 is an ensemble of size 5 with no limit on the maximum number of samples. E_5 is an ensemble of size 11 with no limit on the maximum number of samples.

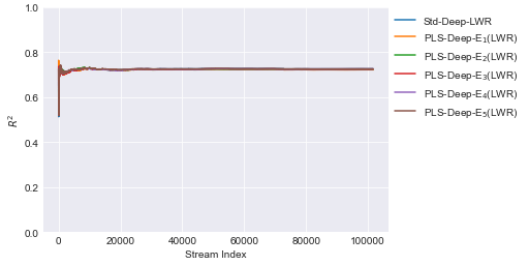
(a) Std-Deep



(b) Std-Deep, Rolling



(c) PLS-Deep



(d) PLS-Deep, Rolling

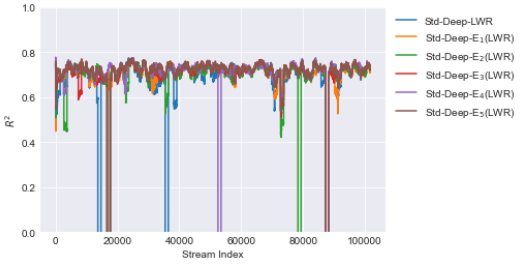
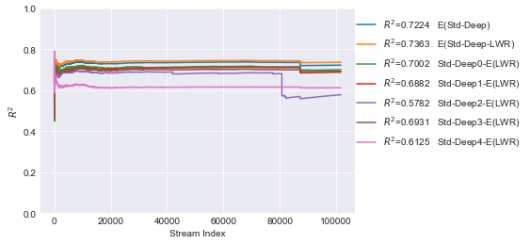
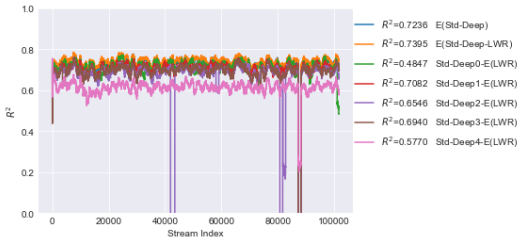


Figure 4.19: **Ensemble Results**. Performance values at the end of the streaming evaluation are included in the legend.

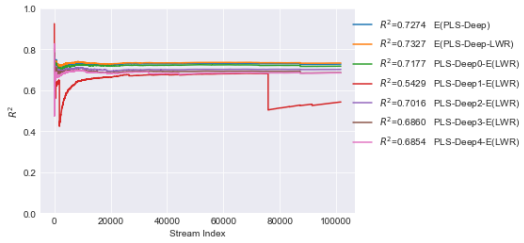
(a) Std-Deep Ensembles



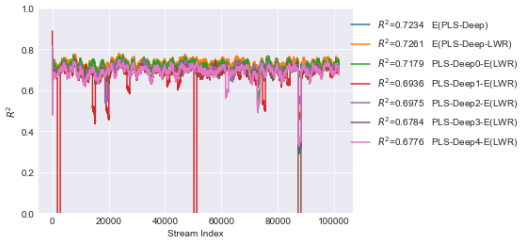
(b) Std-Deep Ensembles, Rolling Average



(c) PLS-Deep Ensembles



(d) PLS-Deep Ensembles, Rolling Average



the performance of the base model is not necessarily a good indicator.

4.3.4 Ensemble Diversity

To better compare the above approaches, we look at diversity in the ensembles, as measured by the correlation coefficient between the predictions of each candidate model and the final voted predictions. We report these in Table 4.8, with extra columns representing the correlation coefficient $p(v, y)$ and R^2 between the voted predictions (v) and the true target values y.

Care needs to be made when interpreting these diversity numbers. On the one hand, ensembles rely on diversity to improve predictions, with models making bad predictions on certain instances outvoted by the other models. On the other hand, diversity can also result from each model making different but equally bad predictions for specific instances. With the median voting rule, good ensemble performance relies on models agreeing (little diversity) for instances when individual predictions are good and disagreeing (lots of diversity) for instances where one or more individual predictions are bad. Given the

Table 4.8: Ensemble Diversity (after 100,000 instances)

Ensemble	Model 1	Model 2	Model 3	Model 4	Model 5	p(v,y)	R^2
E(pls-deep)	0.9834	0.9741	0.9606	0.9756	0.9773	0.8533	0.7274
E(pls-deep-lwr)	0.9818	-0.0043	0.3630	0.9752	0.4400	0.8560	0.7327
E(std-deep)	0.9805	0.9489	0.9396	0.9533	0.9054	0.8503	0.7224
E(std-deep-lwr)	0.3638	0.9697	0.9116	0.9631	0.9232	0.8586	0.7363
pls-deep0-E(lwr)	0.9231	0.9936	0.9952	-	-	0.8474	0.71767
pls-deep1-E(lwr)	0.0105	-0.0027	0.0007	-	-	0.7589	0.5429
pls-deep2-E(lwr)	0.8110	0.9917	0.9944	-	-	0.8378	0.7016
pls-deep3-E(lwr)	0.9933	0.9935	0.9935	-	-	0.8286	0.6860
pls-deep4-E(lwr)	0.9687	0.7854	0.9718	-	-	0.8287	0.6854
std-deep0-E(lwr)	0.7238	-0.0075	0.9457	-	-	0.8380	0.7002
std-deep1-E(lwr)	0.9970	0.9962	0.9981	-	-	0.8300	0.6882
std-deep2-E(lwr)	0.8630	0.9958	0.9935	-	-	0.7729	0.5781
std-deep3-E(lwr)	0.2953	0.9661	0.8977	-	-	0.8332	0.6931
std-deep4-E(lwr)	0.9794	0.9903	0.9889	-	-	0.7831	0.6125

magnitude of several of the errors encountered for LWR models, it is likely that much of the large reported diversity scores are the result of large disagreement over only a few values.

While the heterogeneous ensembles show several individual models with a high level of variance against the voted predictions, most of the models show a high level of agreement ($p > 0.9$) to the vote. With a relatively high level of correlation of the vote to the actual values, we see that these ensembles are effective at overruling bad predictions made by individual models.

The bagging models display more variance in diversity for individual models. Furthermore, several correlations between median votes and the actual values are lower $p < 0.8$, indicating that either there is insufficient diversity to overrule bad predictions or that two or more models are displaying different bad predictions for the same instances.

4.3.5 Test Results and Summary

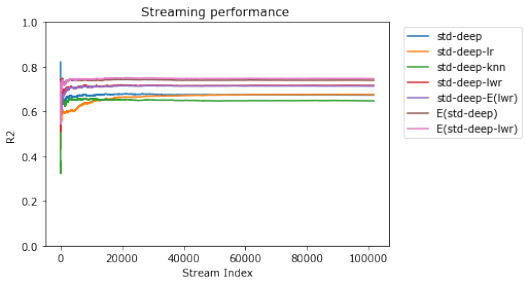
The best-performing models in the preceding experiments are now evaluated on a fresh set of 100,000 instances. The previous experiments have made several passes over the same data set, including hyperparameter searches. We want to ensure that results are not selected to perform well only on the previous data set. In these experiments, ensembles of LWR models are set to have unbounded predictions, while predictions for non-ensembled LWR models are bounded to their neighbourhood set.

Results are shown in both figure (Figure 4.20) and Table (Table 4.9) form. On this test set, the best overall model is a heterogeneous ensemble (E(Std-Deep-LWR)) with an MSE of 121.3208. The best performing non-ensemble was Std-Deep-LWR (MSE = 125.2043), with the best performing base deep model being PLS-Deep (MSE = 127.5589).

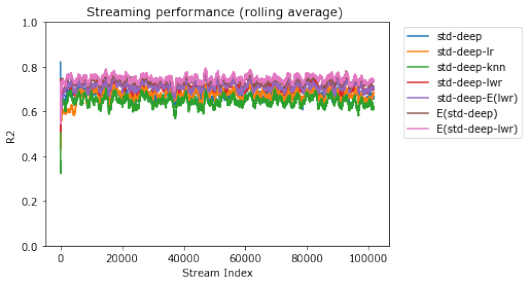
No significant dips in performance are observed across this test evaluation set, which we attribute to a combination of selection effects and both bounded predictions and ensemble strategies being effective for LWR models.

Figure 4.20: Streaming Test Results

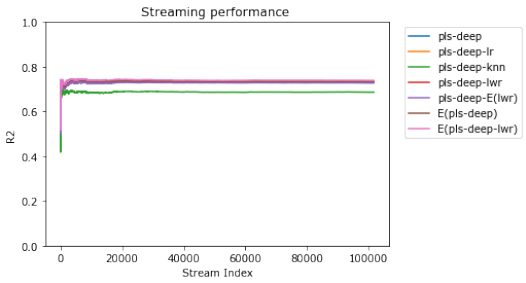
(a) Std-Deep



(b) Std-Deep, Rolling



(c) Pls-Deep



(d) Pls-Deep, Rolling

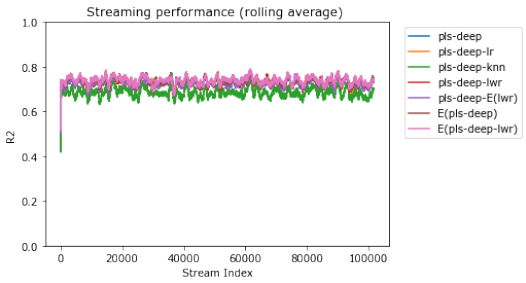


Table 4.9: Streaming Test Set Results

Model	Std		PLS	
	MSE	R2	MSE	R2
Deep	156.4715	0.6734	127.5589	0.7337
Deep-LR	155.4898	0.6754	127.14204	0.7346
Deep-LWR$_b$	135.9844	0.7161	128.7869	0.7310
E(Deep-LWR)	**121.3208**	**0.7467**	125.2043	0.7386
Deep-E(LWR)	137.2359	0.7135	130.8086	0.7269

Chapter 5

Conclusion

This thesis investigated using deep neural networks with locally weighted regression to solve regression problems with infrared spectral data.

For this problem, Partial Least Squares (PLS), combined with either linear or non-linear regression, are predominant in the literature and Industry. Past applications of neural networks to this problem have tended to use only a single hidden layer, and more recent publications have adopted CNN architectures.

The approach described here consisted of building deep, dense networks with high-performing architectures and parameter states selected by Monte Carlo search. These models proved effective compared to a PLS baseline across several proprietary spectral datasets. The datasets varied in size from 1.4K to 411K instances and, as such, are much larger than those normally reported in the literature. Results for all four datasets showed that deep models always outperform classical methods. Two of the datasets (the smallest, D1 and largest, D4) benefit from PLS preprocessing, while the other two (D2 and D3) perform best with the combination of standardisation preprocessing and LWR. This latter result suggests that neighbourhood effects may be more prevalent but this phenomenon requires further investigation.

On the public domain Mangoes dataset, PLS preprocessing combined with the deep modelling outperformed all other published results, including a recent CNN architecture. In addition, a PLS preprocessing method substantially de-

creases the parameter count while discarding a minimal amount of information relevant to the target of interest. The addition of lazy LR or LWR improves performance even further. Interestingly, PLS-LWR outperformed Std-Deep on this dataset, something not seen on D1 to D4, indicating that PLS is particularly effective for the Mangoes dataset.

Lazy learners are also commonly used for modelling non-linear dynamics in infrared spectral analysis. As such, the technique was extended to features extracted from deep networks to allow better extrapolation to unseen data for deep models. The findings for these methods were mixed; kNN methods, while proving robust to outliers, struggled to generalise from cross-validation performance to test set performance. Locally weighted regression, on the other hand, displayed much better performance. For kNN and locally weighted regression, performance gains were much stronger for non-optimal deep networks, with most searched models converging towards the single best model. Improvements on the single best model were mixed; performance is either maintained or improved.

For streaming data, LWR predictors are naturally suited to this environment as they can be calculated incrementally based on sliding windows of data. Care needs to be taken to guard against local models extrapolating too far beyond the domain of their neighbourhood set, as this can lead to severe outliers. An ensemble containing multiple heterogeneous models was found to solve this problem, with a bagging-based ensemble proving less effective.

5.1 Future Work

Future applications of this approach can further optimise many of the design choices made here. A key variable is the number of latent variables for PLS prepossessing, which functions as a bias-variance trade-off, with larger, more powerful models being harder to train.

The current approach treats locally weighted regression as conditionally

independent of the deep learners. Future work could incorporate the error information garnered when training deep learners in neighbourhood decisions. The error could be used as a measure of the quality of neighbours. Alternatively, high errors could be a focus, and a boosting framework developed.

While streaming was only briefly investigated, the suitability of a lazy-deep approach for streaming was part of the motivation behind this work. For pre-trained deep networks, lazy predictors, besides allowing incremental updates, can be used to select further training data for deep networks or incorporated into self-supervised approaches.

References

[1] Charu C. Aggarwal. An introduction to neural networks. In Charu C. Aggarwal, editor, *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, Cham, 2018.

[2] Nicholas Anderson, Kerry Walsh, Jamie Flynn, and J Walsh. Achieving robustness across season, location and cultivar for a nirs model for intact mango fruit dry matter content. ii. local pls and nonlinear models. *Postharvest Biology and Technology*, 171:111358, 10 2020.

[3] Nicholas T Anderson and Kerry B Walsh. Review: The evolution of chemometrics coupled with near infrared spectroscopy for fruit quality evaluation. *Journal of Near Infrared Spectroscopy*, 30(1):3–17, February 2022. Publisher: SAGE Publications Ltd STM.

[4] N.T. Anderson, K.B. Walsh, P.P. Subedi, and C.H. Hayes. Achieving robustness across season, location and cultivar for a nirs model for intact mango fruit dry matter content. *Postharvest Biology and Technology*, 168:111202, 2020.

[5] Eric Bax, Lingjie Weng, and Xu Tian. Validation of k-nearest neighbor classifiers using inclusion and exclusion. *CoRR*, abs/1410.2500, 2014.

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[7] Albert Bifet, Ricard Gavaldà, Bernhard Pfahringer, and Geoffrey Holmes. *Machine Learning for Data Streams with Practical Examples in MOA*. 03 2018.

[8] Els Bobelyn, Anca-Sabina Serban, Mihai Nicu, Jeroen Lammertyn, Bart M. Nicolai, and Wouter Saeys. Postharvest quality of apple predicted by nir-spectroscopy: Study of the effect of biological variability on spectra and model performance. *Postharvest Biology and Technology*, 55(3):133–143, March 2010.

[9] Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *ArXiv*, abs/2110.01889, 2021.

[10] Steven D Brown. The unscrambler®, version 5.5. multivariate analysis software for ms-dos. *Journal of Chemometrics*, 9(6):527–529, 1995.

[11] Vitezslav Centner and D. Luc Massart. Optimization in locally weighted regression. *Analytical Chemistry*, 70(19):4206–4211, October 1998.

[12] Dragan A Cirovic. Feed-forward artificial neural networks: applications to spectroscopy. *TrAC Trends in Analytical Chemistry*, 16(3):148–155, 1997.

[13] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

[14] Chenhao Cui and Tom Fearn. Modern practical convolutional neural networks for multivariate regression: Applications to nir calibration. *Chemometrics and Intelligent Laboratory Systems*, 182, 07 2018.

[15] Chenhao Cui and Tom Fearn. Modern practical convolutional neural networks for multivariate regression: Applications to nir calibration. *Chemometrics and Intelligent Laboratory Systems*, 182:9–20, November 2018.

[16] A. M. Davie and A. J. Stothers. Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh: Section A Mathematics*, 143(2):351–369, 2013.

[17] M.S. Dhanoa, S.J. Lister, R. Sanderson, and R.J. Barnes. The link between multiplicative scatter correction (msc) and standard normal variate (snv) transformations of nir spectra. *J. Near Infrared Spectrosc.*, 2(1):43–47, Jan 1994.

[18] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. *CoRR*, abs/1512.03965, 2015.

[19] D. S. Ferreira, O. F. Galão, J. A. L. Pallone, and R. J. Poppi. Comparison and application of near-infrared (nir) and mid-infrared (mir) spectroscopy for determination of quality parameters in soybean samples. *Food Control*, 35(1):227–232, January 2014.

[20] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018.

[21] Wei Gao, Xin-Yi Niu, and Zhi-Hua Zhou. On the consistency of exact and approximate nearest neighbor with noisy data. *CoRR*, abs/1607.07526, 2016.

[22] Heitor Murilo Gomes, Jacob Montiel, Saulo Martiello Mastelini, Bernhard Pfahringer, and Albert Bifet. On ensemble techniques for data stream regression. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[23] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 18932–18943. Curran Associates, Inc., 2021.

[24] Yunhui Guo, Noel C. Codella, Leonid Karlinsky, James V. Codella, John R. Smith, Kate Saenko, Tajana Rosing, and Rogerio Feris. A broader study of cross-domain few-shot learning. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 124–141. Springer International Publishing, 2020.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[26] Donald E Hilt and Donald W Seegrist. *Ridge, a computer program for calculating ridge regression estimates*, volume 236. Department of Agriculture, Forest Service, Northeastern Forest Experiment, 1977.

[27] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[28] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[29] Tomas Isaksson and Tormod Næs. The effect of multiplicative scatter correction (msc) and linearity improvement in nir spectroscopy. *Appl. Spectrosc.*, 42(7):1273–1284, Sep 1988.

[30] L.J. Janik, D. Cozzolino, R. Dambergs, W. Cynkar, and M. Gishen. The prediction of total anthocyanin concentration in red-grape homogenates using visible-near-infrared spectroscopy and artificial neural networks. *Analytica Chimica Acta*, 594(1):107–118, June 2007.

[31] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Regularization is all you need: Simple neural nets can excel on tabular data. *CoRR*, abs/2106.11189, 2021.

[32] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[35] Sanghyeop Lee, Junyeob Kim, Hyeon Kang, Do-Young Kang, and Jangsik Park. Genetic algorithm based deep learning neural network structure and hyperparameter optimization. *Applied Sciences*, 11, 01 2021.

[36] Mo Li, Reddy R. Pullanagari, Thamarath Pranamornkith, Ian J. Yule, and Andrew R. East. Quantitative prediction of post storage 'hayward' kiwifruit attributes using at harvest vis-nir spectroscopy. *Journal of Food Engineering*, 202:46–55, June 2017.

[37] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary Yen, and Kay Tan. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, PP, 08 2021.

[38] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[39] V Andrew McGlone and Sumio Kawano. Firmness, dry-matter and soluble-solids assessment of postharvest kiwifruit by nir spectroscopy. *Postharvest Biology and Technology*, 13(2):131–141, 1998.

[40] Puneet Mishra and Dário Passos. Deep chemometrics: Validation and transfer of a global deep near-infrared fruit model to use it on a new portable instrument. *Journal of Chemometrics*, 35(10):e3367, 2021.

[41] Puneet Mishra and Dário Passos. A synergistic use of chemometrics and deep learning improved the predictive performance of near-infrared spectroscopy models for dry matter prediction in mango fruit. *Chemometrics and Intelligent Laboratory Systems*, 212:104287, May 2021.

[42] Itzik Mizrahi and Shai Avidan. knet: A deep knn network to handle label noise. *arXiv:2107.09735 [cs]*, July 2021. arXiv: 2107.09735.

[43] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphaël Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. River: machine learning for streaming data in python. *CoRR*, abs/2012.04740, 2020.

[44] Tormod. Naes, Tomas. Isaksson, and Bruce. Kowalski. Locally weighted regression and scatter correction for near-infrared reflectance data. *Analytical Chemistry*, 62(7):664–673, April 1990. Publisher: American Chemical Society.

[45] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2603–2612. PMLR, 06–11 Aug 2017.

[46] Bart M Nicolai, Karen I Theron, and Jeroen Lammertyn. Kernel pls regression on wavelet transformed nir spectra for prediction of sugar content of apple. *Chemometrics and intelligent laboratory systems*, 85(2):243–252, 2007.

[47] Bart M. Nicolaï, Katrien Beullens, Els Bobelyn, Ann Peirs, Wouter Saeys, Karen I. Theron, and Jeroen Lammertyn. Nondestructive measurement of fruit and vegetable quality by means of nir spectroscopy: A review. *Postharvest Biology and Technology*, 46(2):99–118, November 2007.

[48] Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv:1803.04765 [cs, stat]*, March 2018. arXiv: 1803.04765.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Van-

derplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[51] Ann Peirs, Jeroen Tirry, Bert Verlinden, Paul Darius, and Bart M. Nicolaı. Effect of biological variability on the robustness of nir models for soluble solids content of apples. *Postharvest Biology and Technology*, 28(2):269–280, May 2003.

[52] Michal Pinos, Vojtech Mrazek, and Lukás Sekanina. Evolutionary neural architecture search supporting approximate multipliers. *CoRR*, abs/2101.11883, 2021.

[53] Dian Rong, Haiyan Wang, Yibin Ying, Zhengyong Zhang, and Yinsheng Zhang. Peach variety detection using vis-nir spectroscopy and deep learning. *Computers and Electronics in Agriculture*, 175:105553, August 2020.

[54] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[55] Roman Rosipal and Leonard J Trejo. Kernel partial least squares regression in reproducing kernel hilbert space. *Journal of machine learning research*, 2(Dec):97–123, 2001.

[56] Wouter Saeys, Nghia Nguyen Do Trong, Robbe Van Beers, and Bart M. Nicolaï. Multivariate calibration of spectroscopic sensors for postharvest quality evaluation: A review. *Postharvest Biology and Technology*, 158:110981, December 2019.

[57] Bernhard Schölkopf. The kernel trick for distances. *Advances in neural information processing systems*, 13, 2000.

[58] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[59] Ira Shavitt and Eran Segal. Regularization learning networks: Deep learning for tabular datasets. In *NeurIPS*, 2018.

[60] Guanghui Shen, Matthieu Lesnoff, Vincent Baeten, Pierre Dardenne, Fabrice Davrieux, Hernan Ceballos, John Belalcazar, Dominique Dufour, Zengling Yang, Lujia Han, and Juan Antonio Fernández Pierna. Local partial least squares based on global pls scores. *Journal of Chemometrics*, 33(5):e3117, 2019.

[61] John S. Shenk, Mark O. Westerhaus, and Paolo Berzaghi. Investigation of a local calibration procedure for near infrared instruments. *Journal of Near Infrared Spectroscopy*, 5(4):223–232, October 1997. Publisher: SAGE Publications Ltd STM.

[62] Amel Sifaoui, Afef Abdelkrim, and Mohamed Benrejeb. On the use of neural network as a universal approximator. *Int. J. Sci. Tech. Control Comput. Eng*, 2:386–399, 2008.

[63] SN Sivanandam and SN Deepa. Genetic algorithm optimization problems. In *Introduction to genetic algorithms*, pages 165–209. Springer, 2008.

[64] Leslie N. Smith. Cyclical learning rates for training neural networks. *CoRR*, abs/1506.01186, 2015.

[65] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

[66] P.P. Subedi, K.B. Walsh, and G. Owens. Prediction of mango eating quality at harvest using short-wave near infrared spectrometry. *Postharvest Biology and Technology*, 43(3):326–334, March 2007.

[67] J.A.K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, June 1999.

[68] Samuele Tosatto, Riad Akrour, and Jan Peters. An upper bound of the bias of nadaraya-watson kernel regression under lipschitz assumptions. *Stats*, 4(1):1–17, 2021.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[70] N. Vlachos, Y. Skopelitis, M. Psaroudaki, V. Konstantinidou, A. Chatzilazarou, and E. Tegou. Applications of fourier transform-infrared spectroscopy to edible oils. *Analytica Chimica Acta*, 573-574:459–465, July 2006.

[71] K.B. Walsh, V.A. McGlone, and D.H. Han. The uses of near infra-red spectroscopy in postharvest decision support: A review. *Postharvest Biology and Technology*, 163:111139, May 2020.

[72] Kerry B. Walsh, José Blasco, Manuela Zude-Sasse, and Xudong Sun. Visible-nir 'point' spectroscopy in postharvest fruit and vegetable assessment: The science behind three decades of commercial use. *Postharvest Biology and Technology*, 168:111246, October 2020.

[73] Hailong Wang, Jiyu Peng, Chuanqi Xie, Yidan Bao, and Yong He. Fruit quality evaluation using spectroscopy technology: A review. *Sensors*, 15(5):11889–11927, May 2015.

[74] Hongyu Wang, Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Mayo. *A Comparison of Machine Learning Methods for Cross-Domain Few-Shot Learning*, pages 445–457. 11 2020.

[75] B.B. Wedding, C. Wright, S. Grauf, R.D. White, B. Tilse, and P. Gadek. Effects of seasonal variability on ft-nir prediction of dry matter content for whole hass avocado fruit. *Postharvest Biology and Technology*, 75:9–16, January 2013.

[76] Jacob Wegelin. A survey of partial least squares (pls) methods, with emphasis on the two-block case. *Technical report*, 04 2000.

[77] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016.

[78] Svante Wold, Michael Sjöström, and Lennart Eriksson. Pls-regression: a basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, 58(2):109–130, October 2001.

[79] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.

[80] Hui Yan and Heinz W Siesler. Hand-held near-infrared spectrometers: State-of-the-art instrumentation and practical applications. *NIR news*, 29(7):8–12, 2018.

[81] Eduardo Zamora Rojas, Ana Garrido-Varo, Frans Berg, Jose Guerrero Ginel, and D.C. Perez-Martin. Evaluation of a new local modelling approach for large and heterogeneous nirs data sets. *Chemometrics and Intelligent Laboratory Systems*, 101:87–94, 04 2010.