A Variational Framework for Supervised Learning

by

Carlos David Brito Pacheco

Thesis submitted in accordance with the requirements
of Universidad Autónoma de Yucatán for the
degree of Bachelor in Computer Science.

June 2018

Contents

Acknowledgements

My dog Chubbs, the best boy there was, is and will be.

List of Figures

List of Tables

Chapter 1

The Supervised Learning Problem

1.1 Overview

The problem of *supervised learning* arises in contexts where a study of a *dependent variable* Y is necessary in terms of an independent variable X. The goal of supervised learning is to predict the values of Y given many instances of the variable X. Generally, we call the instances of X and the values Y takes, the *inputs* and *outputs* respectively. In the *pattern recognition* context we typically may read *features* and *responses* as an alternative terminology. With this in mind, it is worthy to note that we are assuming that there exists a function between X and Y such that $Y = f(X)$. This is akin to saying, for example, we cannot predict the rain solely based on observations if a glass falls off a table in somebody's house. I.e. there *must* be a *relation* between these two variables. Cases in which there is no such function are not of our concern.

From a theoretical perspective, it is necessary to formally define the concepts beforehand. Let $\mathcal{T}$ be a set of N observations on the variable X, then

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N) \mid \mathbf{x}_i \in \Omega, y_i \in \mathcal{Y}\} \tag{1.1}$$

where each $\mathbf{x}_i$ and y_i are instances of X and Y respectively. The set $\Omega \subset R^m$ is called the *feature space* of the variable X and the set $\mathcal{Y}$ is called the *target space*. The target space's cardinality may be either finite or infinite, depending on the problem at hand; if classifying among a discrete number of labels then $|\mathcal{Y}|$ will be finite, but if performing regression then $\mathcal{Y}$ will represent any subset of $\mathbb{R}^m$. The problem of supervised learning is the task of finding the mapping or function $\hat{f}$ that best approximates the real relationship between the two variables. In symbols, we would like to find $\hat{f} : \Omega \to \mathcal{Y}$ such that:

$$\hat{f}(X) \approx f(X), \tag{1.2}$$

where we are given many instances of X with their associated values of Y. Admit-

tedly, the interpretation of supervised learning is to let a model "learn by example". Moreover, the learning and predicting will be done on a computer so as to mimic that the "machine has learnt". As a result we can think supervised learning as a form of *artificial intelligence* where a computer has "learnt" to identify patterns and make inferences based on a set of data. In the following chapters we shall explore how we can use a variational approach to tackle this problem.

1.2 Classification vs. Regression

In addition to defining the problem of supervised learning, we can separate it into two distinct cases based on the nature of the response variable Y, namely Y discrete and Y continuous. If Y takes a discrete form then, in the context of supervised learning, we say that we would like to perform *classification* over *qualitative data*. For example, say that we had an input variable X and a target space $\mathcal{Y} = \{0, 1, 2, 3, 4\}$. We then would say that we want to *classify* instances of X among 5 different classes.

An even more concrete example is to let X be the data of a patient who underwent a new revolutionary experimental treatment. The variable X contains information about the patient's age, blood type, condition, etc. The target variable Y is if the patient either "died" or "survived". If we let the number -1 represent "died" and the number 1 represent "survived" then our target space would look like $\mathcal{Y} = \{-1, 1\}$. We would like to predict if the treatment carries a fatal risk to a one particular patient given her/his data. In our terminology, we would like to classify the patient into one of our two classes. In general we may call the elements in the discrete target space *labels*. When we only have two labels in the target space, we say we are performing *binary classification*. When discussing binary classification most authors will choose $\mathcal{Y} = \{-1, 1\}$. For this purpose we shall use this convention too. In Figure 1.1 we can see how a classifier has attempted to separate both classes by generating a *decision boundary* which dictates the predicted class of a new input point. Notice that for this particular example, the feature space is a subset of $\mathbb{R}^2$ and the target space is $\mathcal{Y} = \{-1, 1\}$.

For a continuous variable, however, we say that the we are working *quantitative data*. As an example, say we have the input variable X that contains information on houses. This may be the area of a house, the year built and whether or not the house has a garden, whether or not it has a pool, location, etc. And our output variable Y is its price in dollars. Then in our context we wish to predict the price of a new house given many previous observations of other houses. The target space for our variable would then be $\mathcal{Y} = [0, \infty)$. In Figure 1.2 we can observe an example of regression being done over a set of data points. Although we could attempt to make the curve pass through most, if not all, the points this would serve little purpose as the relationship between the feature and the target is clearly linear with some added noise. In principle, when performing regression we try to choose the simplest function that best predicts

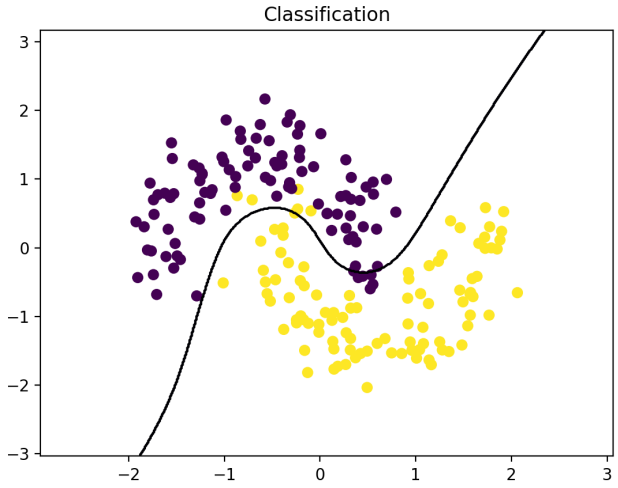

Figure 1.1: The decision boundary (in black) for a binary classifier and the points of a "moon shaped dataset" coloured by their given labels.

the data, which is an analogue action to following *Occam's razor*; the simplest function is the one that should be selected. This also applies to classification as we shall see in the next section.

All in all, when we want to predict qualitative data we say we want to perform classification on the data and when we want to predict quantitative data we say we want to perform regression on the data. While regression might be seen as fitting a continuous function to the data, we can think of classification as separating hyperplanes in the m-dimensional space. As an illustration, when performing binary classification this last action might be done by applying a *threshold* or *activation* function σ on the fitted function, so that their composition is $\sigma(\hat{f}(X)) \in \{0, 1\}$. This is interpreted as the probability of the input X having the response $Y' = (Y + 1)/2 \in \{0, 1\}$ which is a transformation of the original response variable $Y \in \{-1, 1\}$. Going back to our previous example, we may graph a separation in the $\mathbb{R}^2$ plane by plotting each half different colours as seen in Figure 1.3. In general, however, most methods require that this function be differentiable [CITATION NEEDED] and so most times we impose the restriction $\sigma \in C^\infty$ and $\sigma \in (0, 1)$.

1.3 Regularization

When performing regression or classification, it is clear that we want to find the true underlying function which will permit us to perform predictions accurately. It is important, however, to determine which type of model to use; linear or non-linear. I.e. we must specify if the function $\hat{f}(X)$ that we are trying to fit is linear, and if not, must choose an appropriate representation that may approximate non-linear functions. It is clear, however, that not all phenomena will obey a linear pattern. This poses the

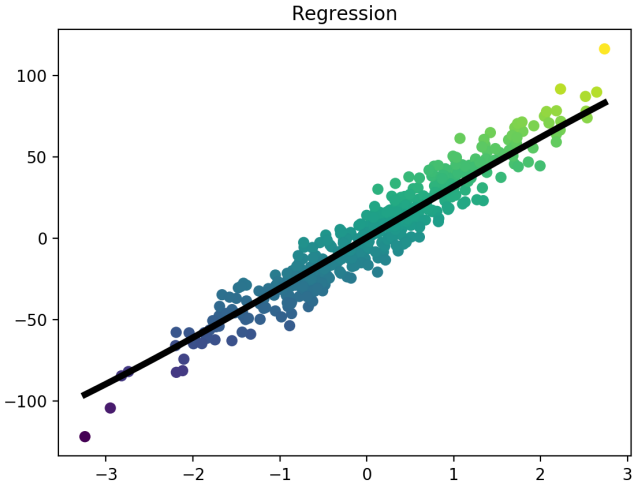

Figure 1.2: A fitted continuous function (black) that predicts new inputs.

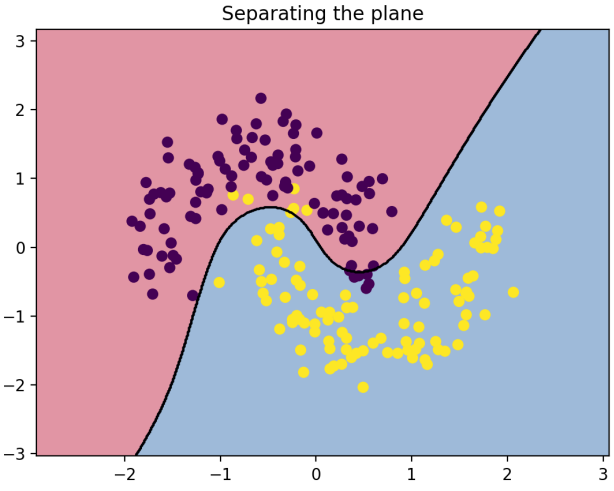

Figure 1.3: Colouring the two halves of the plane different colours.

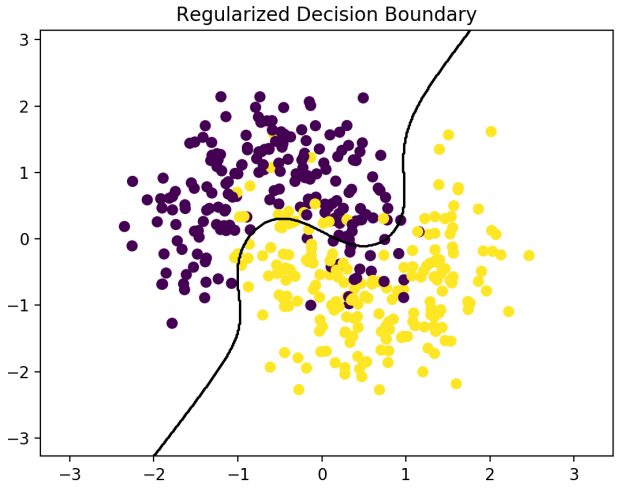

Figure 1.4: A regularized boundary. The rate of misclassification for new inputs should be low.

problem of choosing among a large class of functions and, if not addressed, will lead to *overfitting* the learnt function. The problem of overfitting is well known in the area of statistical inference and has been known since the start of supervised learning. This problem occurs when a given model fits the function, with great degree, solely to the training data so that new inputs that follow a similar trend are predicted incorrectly. Comparatively, we see this phenomenon in humans such that a person might attempt to memorize what he/she understands instead of identifying a common pattern. Another interpretation is that overfitting is equivalent to the fitted function being "too complex". That is, the model finds a highly non-linear function even though the trend might follow a near linear pattern. To illustrate these ideas, we may provide visual aid so a geometric interpretation becomes clear. Take, for example, a similar dataset as the one in the previous section. A nicely fitted function might have a decision boundary such as the one shown in Figure 1.4.

On the contrary, an overfitted function might look like Figure 1.5. Notice that while the decision boundary in Figure 1.4 is smooth and separates and tries to separate the plane, the fitted function in Figure 1.5 tries to fit individual points in both spaces and has sharp edges at the frontier of both class sets.

Unfortunately, the problem of picking a regularization term has no clear "best" solution. I.e. there is not a unique term that best prevents overfitting while not underfitting the data. In this work, however, we present some guides when choosing a regularizer term for our model which lead to stable solutions and prevent overfitting.

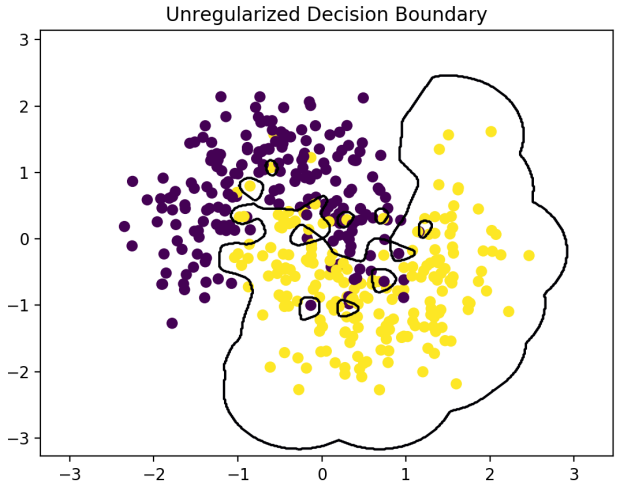

Figure 1.5: An overfitted decision boundary. The rate of misclassification for new inputs would be high, while the rate of misclassification for known inputs is practically zero.

1.4 Other types of learning and their relationship to Supervised Learning

It is well known that supervised learning can be viewed as a subset of machine learning. It is, however, necessary to distinguish between the other types of "learning". The purpose of this section to try an explain the differences and similarities between these.

1.4.1 Statistical Learning

The first type of learning that is discussed, goes by the name of *statistical learning*. The distinction between these two areas come from the fact that statistical learning takes a statistical approach to the task of *machine learning*. That is, in the literature of statistical learning, we will find methods, based or derived from a statistical assumption. It is clear then that we will find common techniques such as statistical inference, bayesian methods, computation and derivation of statistics, and confidence intervals used throughout this area. The reason behind this approach enables to use all of the underlying, well developed and understood statistical, theory and apply it to machine learning methods where it is appropriate. Generalizing, statistical learning can be regarded as *an approach to machine learning through the use of statistical and probabilistic theory*. This can lead to saying that they're nearly equivalent in the sense that they both concern themselves with solving the same problem, through different means. As an example of a problem in the statistical learning context, one might suppose that the data being learned arose from a statistical model

$$Y = f(X) + \epsilon \ , \tag{1.3}$$

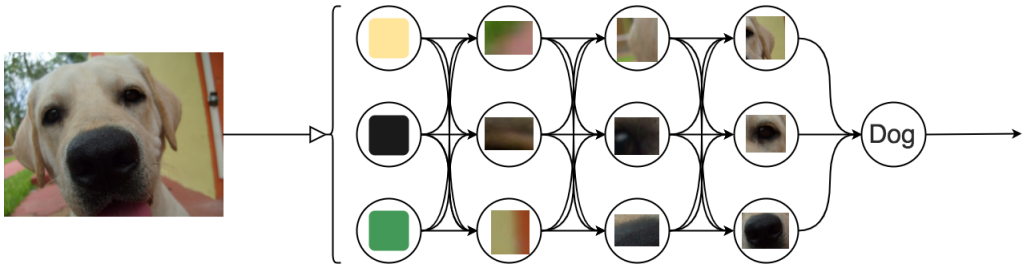

Figure 1.6: A graph representing the concept of *a picture of a dog.*

where the random error ϵ takes a probabillity distribution with expected value $E[\epsilon] = 0$ and is independent of the random variable X. Employing this scheme it is possible to minimize the expected prediction error $E[Y - f(X)]^2$ by obtaining a statistical condition which must be satisfied to attain its minimum when choosing f. Other areas where statistical learning can be used include: *data mining*, *statistical inference* and *dimensionality reduction*. A vast, encyclopaedic collection of material can be found in the book *The Elements of Statistical Learning* by Hastie *et al.* [17].

1.4.2 Deep Learning

Unlike statistical learning, *deep learning* cannot be considered equivalent to supervised learning but rather a subset of it [1]. Through the use of graph based models, such as *neural networks*, to represent concepts which are built on top of each other. The resulting graph is called *deep* because of the many layers it takes to represent a concept. The graphs are then trained based on inputs to labelled targets. One of the principal keystones of deep learning methods, is that they rely heavily on the representation of the data they are given. We may give the example of a graph that will try to model the concept of *a picture of a dog*. Notice that rather than modelling the concept of a dog, we are modelling the *picture of one*. The difference is the representation; a dog is a concept of the animal that has ears, eats, sleeps, has a tongue, stands on four legs, etc. Whereas the picture must contain defining physical characteristics of a dog: a tail, pointy ears, a dog nose, fur, etc. We may illustrate this in Figure 1.6. Here, the graph makes a decision on whether the picture contains a dog or not. Taking the input as raw pixels, each layer builds on the information produced by the previous one, *i.e.* raw pixels form edges, edges form contours and corners, contours and corners form parts of a dog, and finally, dog parts form a dog.

One realization of deep learning is the concept of an *artificial neural network*, which uses *artificial neurons* as the graph's nodes. This is arguably the most popular method among the deep learning community due to its simplicity to understand and ease of use. In Figure 1.7 we can see a sketch of a basic neuron. The function σ is known as the activation function, which represents whether the neuron has activated or not. Note that this is similar to the idea of how a brain neuron works; each neuron fires

[1] This is not entirely true. Please see the Unsupervised Learning section for more details.

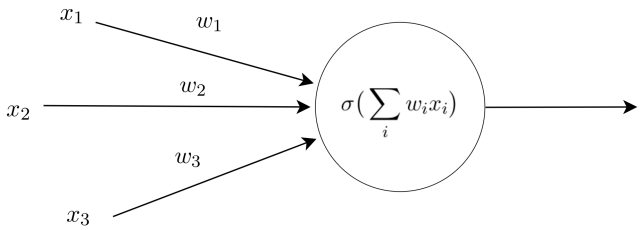

Figure 1.7: An artificial neuron.

if it receives enough charge, eventually triggering other neurons in the network. The activation of a specific combination of neurons will represent a concept, or even an action in the brain. The variables x_i are the features of the input to the neuron, while w_i are the weights of the connections of neurons. The w_i model how strong a connection is between two neurons. The sum inside the activation function has the behaviour of assigning a weight to each input, essentially taking into account how important each feature of the input is.

Unfortunately, even though neural networks are widely used, little is still known about how or why they work. This leaves a considerable amount of work yet to be done. Arguably, the most important theorem at this time of writing is the Universal Approximation Theorem [6], first proved by George Cybenko, which states that a one hidden layer neural network with finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, with very mild assumptions on σ. Later versions of the theorem [7] generalize to multiple layers and characterize neural networks as *universal approximators*. Conversely, there is a considerable amount of work being done on the uses and implementations of neural networks through *ad-hoc* methods in order to try to find the best use cases and optimizations, to improve the performance, and the speed of training. A vast number of articles have been published by many, with documentation and notes being available for consulting on the web and among various books. In particular, the book *Deep Learning* by Ian Goodfellow and Yoshua Bengio, serves both as both a comprehensive reference and an introduction to the field of Deep Learning.

1.4.3 Unsupervised Learning

Whereas supervised learning deals with labelled data, unsupervised learning deals with unlabelled data. Methods in this category include:

- Clustering
- Neural Networks
- Latent Variable Models (*i.e.* expectation-maximization algorithm)

Since no information is given to us by the observed data on the category to which it belongs, this area shifts its focus to the *intrinsic structure* of the data. That is,

unsupervised learning aims to find the properties of the observations by looking at the way it's structured. Essentially, this permits us to observe, otherwise unidentified variables that a human might not see in plain sight.

1.4.4 Reinforcement Learning

1.4.5 One shot learning

Chapter 2

Mathematical Preliminaries

2.1 Normed Linear Spaces

2.2 Convexity

Definition 2.2.1 (Convex Function)**.** *The function f is a convex function if its domain C is a convex set and if for any two points $x, y \in C$, the following property is satisfied:*

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y), \quad \forall \alpha \in [0,1] \tag{2.1}$$

Theorem 2.2.2 (Sum of convex functions)**.** *If f and g are two convex functions with convex sets for domains, then their sum $h = f + g$ is also convex.*

Proof. ■

Theorem 2.2.3. *Let $C \subset \mathbb{R}^m$ be a convex set, and let $f : C \to \mathbb{R}$ be a convex function. Let x^* be a local minimizer of f. Then x^* is a global minimizer.*

Proof. Assume that x^* is a local but not global minimizer. Then there exists z such that $f(z) < f(x^*)$. Consider the line segment from x^* to z,

$$x = \lambda z + (1-\lambda)x^*$$

for $\lambda \in (0, 1]$. Since f is convex, we have

$$f(x) \leq \lambda f(z) + (1-\lambda)f(x^*) < f(x^*)$$

TODO ■

2.3 Constrained Optimization

2.4 Linear Least-Squares

2.5 Levenberg-Marquardt Method

2.6 Radial Basis Function Approximation

Chapter 3

A Variational Approach for the Supervised Learning Problem

3.1 The Problem

The supervised learning problem may be characterized as follows. Given a training set of N observations

$$\mathcal{T} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \mid x_i \in \Omega, y_i \in \mathcal{Y}\} \tag{3.1}$$

we would like to find a function $u = u(x)$ defined by $u : \Omega \to \mathbb{R}$ to predict the associated value y on a, possibly new, input x. We call Ω the feature space. For us to find a suitable function u, it is necessary to suggest a suitable model which best describes the problem. The most widely used framework to solve the supervised learning problem is the minimization of a *loss function* with an added *regularizer term* $S(u)$:

$$\min_u \lambda S(u) + \sum_{i=1}^{N} L(u(x_i), y_i) \ . \tag{3.2}$$

For the continuous case, however, we have

$$\min_u \lambda S(u) + \int_\Omega L(u, y) \ dx \ . \tag{3.3}$$

The idea behind a loss function is to penalize incorrect predictions from the function we are trying to fit, while the regularizer term will prevent the symptom of overfitting and, in passing, allow our model to generalize predictions for new inputs. Usually, we view the loss function as a residual (regression) or the price paid for inaccuracy (classification) between the predictions and the true values. For instance, a regression loss will look like $L(u, y) = ||u - y||$ while a classification loss might look like $L(u, y) = \exp(-yu)$. This work will not be concerned with the extensive analysis of multiple loss functions as such analyses can be found in the modern literature [CITATION NEEDED]. Correspondingly, the regularizer term might be regarded as a term that "controls" the regularized variable. That is, it prevents the regularized variable from

presenting pathologies or overfitting. This being said, it is of interest to find a suitable regularization term which punishes the complexity of u based on some characteristic or property. One might interpret that the previous statement is actually "measuring" the complexity of a characteristic of u and punishing accordingly. This idea can be captured by our model, if we let $R(u) > 0$ be a Lebesgue integrable function and set

$$S(u) = \int_{\Omega} R(u)\ dx \tag{3.4}$$

so that when we substitute into our original equation, we get

$$\min_{u} \int_{\Omega} L(u, y) + \lambda R(u)\ dx\ , \tag{3.5}$$

which will let us shift our focus to the integrand of the problem. It is quite clear that Equation 3.5 is a *functional* depending on u, thus the problem is reduced to a minimization problem that can be solved utilizing the calculus of variations and functional analysis. One way to interpret this last equation, is to see that the empirical risk of choosing a *hypothesis* function $u(x)$ can defined as the expectation of the loss function L:

$$\mathbf{E}[L(u, y)] = \int_{\Omega} L(u, y)\ dx\ . \tag{3.6}$$

Thus, if we want to find a function u that belongs to a space of hypothesis functions X, then we can regard the original problem as the following minimization problem:

$$\min_{u \in X}\ \mathbf{E}[L(u(x), y)]\ . \tag{3.7}$$

Therefore, the probabilistic interpretation of our proposed setup will be to *minimize the regularized empirical risk* by finding an appropriate $\bar{u}$ that minimizes it using a combination of functional and numerical techniques.

3.2 A General Solution to the Problem

Akin to *differential calculus*, we aim to find a condition which a minimizer satisfies by computing the "derivative" of our expression and subsequently equating to zero. The problem then lies in our notion of "derivative of a functional". Fortunately for us, there already exists the well developed theory of Calculus of Variations and Functional Analysis which allow one to reduce various problems of a certain form to a PDE through the use of *variations*. Indeed this PDE will be the optimality condition which we will have to solve in order to solve the problem mentioned in Equation 3.5.

To this end, we should make some arguments as to which form the integrand should take, lest we define an inappropriate model. As a rule in the PDE literature, it is specially important to state explicitly the dependence of derivatives, if any, on a particular function. That is, we must state explicitly if a given function depends explicitly on the derivative of one of its arguments. Therefore, for our case, we must state whether

the loss function or regularization function will depend on the derivative of u. We argue that the former should be able to characterize error we make when choosing a function u, thus, it should depend explicitly on u but it should not rely on any of the partial derivatives of u. This can be seen observed in many, if not all, loss functions utilized in the machine learning literature. On the other hand, we argue that the latter should explicitly depend on the partial derivatives of u but not u itself. The essence of this intuition is that the differential terms of the desired function contain intrinsic information about the structure of $u = u(x)$ at each x. This, in turn, permits the regularization function to measure the complexity of the function and subsequently impose restrictions on it. On the other hand, if the regularization function depended explicitly on u it would add no new information about the complexity u. For this purpose, let us make some definitions precise.

Definition 3.2.1. *Suppose $\Omega \subset \mathbb{R}^n$ is a bounded open set with smooth boundary $\partial\Omega$. Let $L : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ and $R : \mathbb{R}^n \to \mathbb{R}$. Let $\mathfrak{L} : \bar{\Omega} \times \mathbb{R} \times \mathbb{R}^n \to [0, \infty)$ be a function such that it is of the form*

$$\mathfrak{L}(x, z, \xi) := L(z) + \lambda R(\xi) \ . \tag{3.8}$$

*We call $\mathfrak{L}$ the **supervised learning Lagrangian**.*

Remark 3.2.2. *From now on when writing $\mathfrak{L}$ it should be understood as*

$$\mathfrak{L} = \mathfrak{L}(x, z, \xi) = \mathfrak{L}(x_1, \dots, x_n, z, \xi_1, \dots, \xi_n) \ .$$

Remark 3.2.3. *Abusing the notation, when writing $L(z)$ we will really mean $L(z) = L(z, y)$ for a constant function $y = y(x)$.*

Remark 3.2.4. *We note that we reserve the letters L and R to represent the loss and regularization functions respectively. On the other hand, $\mathfrak{L}$ which is not to be confused with the symbol for the loss function, is reserved explicitly for the Lagrangian.*

Remark 3.2.5. *The familiarised reader should have by now realized that we can try to solve the supervised learning problem by deriving the Euler-Lagrange equation associated with an energy functional $I[u]$. This is in fact what most of this manuscript is about.*

Now that we have the function $\mathfrak{L}$, the idea is to attempt to minimize the energy generated all along the feature space Ω. That is, we can regard our setup as minimizing the energy of the loss and of the regularization term which in turn means we find an appropriate solution to our problem. For this purpose, let us define this energy functional.

Definition 3.2.6 (Supervised Learning Energy)**.** *Let $I[u]$ be a functional such that $I : X \to \mathbb{R}$ where X is a vector space, whose exact structure is left unspecified at this moment* [1]*. Then $I[u]$ is the **supervised learning energy** and*

$$I[u] := \int_\Omega \mathfrak{L}(x, u, \nabla u) \ dx = \int_\Omega L(u, y) + \lambda R(\nabla u) \ dx \tag{3.9}$$

[1] In later sections we actually state that this space must be a slightly different version of the usually chosen space $W^{1,p}$

To actually get a precise interpretation of 3.9 we may give a breakdown of the model. The first thing to note is the integral over the feature space Ω. In particular we note that, as opposed to the discrete case, our model looks at *all the possible samples* in the variable's feature space. This means that every [2] possible sample $x \in \Omega$ will be taken into account when deriving a condition for the minimization of 3.9; this makes our definition of the energy functional more general and powerful. Next we note that, rather uninterestingly, we have our loss function $L = L(u, y)$ that follows the basic structure of all the used loss functions in literature. More interestingly, we note that the regularization term depends on u and ∇u. As previously mentioned, we recall that our argument for the dependence of ∇u in $R = R(\nabla u)$ will allow to characterize the *complexity* of the function by obtaining information about its underlying structure. For example, we may opt to choose $R(\nabla u) = ||\nabla u||$, in which case we are minimizing the *total variation* [CITATION NEEDED] of u. In this situation, the "underlying structure" will be the total variation. Trivially this choice of R will lead to a function $u = u(x)$ with low total variation. In the image processing literature, this choice of R is actually called *total variation denoising* [CITATION NEEDED] used for removing noise out of noisy images but it has seen uses elsewhere in the supervised learning context [CITATION NEEDED].

Now that we have a precise definition for the energy which we are trying to minimize, we should accordingly derive an optimality condition for the achievement of a minimum. In the spirit of utilizing the theory behind the calculus of variations and partial differential equations, we will be deriving the associated Euler-Lagrange equation. We will proceed heuristically, not making many assumptions about $\mathcal{L}$ or the class of functions X over which we will perform the choice of u. It is not until in a later chapter which we will prove that a minimizer $\bar{u}$ in fact exists under certain conditions and this minimizer can be calculated using the following proposition. Before proceeding, it is necessary that we clarify the notation employed in Equation 3.10. So then, let

$$L_z := \frac{\partial L}{\partial z}$$
$$R_\xi := (\frac{\partial R}{\partial \xi_1}, \ldots, \frac{\partial R}{\partial \xi_n}) \ .$$

With this in mind let us state our proposition.

Proposition 3.2.7 (Optimality Condition for the Supervised Learning Energy)**.** *Let $I[u]$ be the supervised learning energy. If $F[u]$ attains its minimum at $\bar{u}$, then the following condition is met:*

[2] Since this is just an interpretation, we may turn a blind eye to the actual interpretation which should be *everywhere except in sets of measure zero.*

$$L_z(\bar{u}, y) - \nabla\lambda \cdot R_\xi(\nabla\bar{u}) = 0 \tag{3.10}$$

This type of equation is called an Euler-Lagrange equation.

Our method would then consist in fixing L and R, and subsequently solving for u. For this purpose, let us produce the heuristic argument for the proposition above. Herein let us recall the definition of *first variation* and two of the most important theorems of the calculus of variations.

Definition 3.2.8 (First Variation)**.** *Let $\epsilon \in \mathbb{R}$. Given a vector space X (possibly a Banach space) and a functional $F : X \to \mathbb{R}$, we denote its **first variation** as δF and define it as*

$$\delta F := \lim_{\epsilon \to 0} \frac{F[u + \epsilon\delta u] - F[u]}{\epsilon} = \Big[\frac{\partial}{\partial\epsilon} F[u + \epsilon\delta u]\Big]_{\epsilon=0} \tag{3.11}$$

Remark 3.2.9. *If X is a Banach space or more generally a locally convex topological space, this definition degenerates to the Gâteaux derivative [CITATION NEEDED].*

Correspondingly we may think of the first variation as the first derivative of a functional. In fact, the key concept here is the variation $\epsilon\delta u$ which permits us to "vary" $F[u]$ for small increments $\delta u = \delta u(x)$. We now proceed to give two well known theorems which shall aid us in our search for the minimizer of 3.9.

Theorem 3.2.10 (Minimum of a functional)**.** *Let $F[u]$ be a functional s.t. $F : X \to \mathbb{R}$. Let $u : \Omega \to \mathbb{R}$ be a smooth function. If the first variation of $F[u]$ exists and takes on a minimum at $\bar{u} = \bar{u}(x)$ then*

$$\delta F = 0 \tag{3.12}$$

along $\bar{u} = \bar{u}(x)$.

Remark 3.2.11. *The actual proof is very simple. In fact, the only real trouble is choosing an appropriate space of functions for δu which should clearly be a space of test functions, i.e. $\delta u \in C_c^\infty(\Omega)$ with $\Omega \subset \mathbb{R}^n$.*

Proof. Choose $\delta u \in C_c^\infty(\Omega)$, i.e. a smooth function with compact support. Consider a function $h : \mathbb{R} \to \mathbb{R}$ s.t.

$$h(\epsilon) := F[u + \epsilon\delta u] \ . \tag{3.13}$$

Notice that since $\bar{u}$ is a minimizer of $F[u]$ then h achieves its minimum at $\epsilon = 0$. Clearly then

$$h'(0) = 0$$

iff

$$\Big[\frac{\partial}{\partial\epsilon} F[u + \epsilon\delta u]\Big]_{\epsilon=0} = 0$$

iff

$$\delta F = 0$$

■

Although simple, this result is very powerful when attempting to find minima of a functional $F[u]$. For our particular case, however, we will need the most important result of the calculus of variations.

Theorem 3.2.12 (Fundamental Lemma of the Calculus of Variations)**.** *Let $\Omega \subset \mathbb{R}^n$ be an open set. Let $u \in L^2(\Omega)$ be a continuous s.t.*

$$\int_\Omega u(x)\phi(x)\ dx = 0 \tag{3.14}$$

for all $\phi \in C_c^\infty(\Omega)$, then $u = 0$ a.e. in Ω.

Remark 3.2.13. *This theorem is true for various generalizations, the key being choosing the space of functions where ϕ and u belong to.*

Remark 3.2.14. *We will prove the theorem for quite strong hypotheses but the reader is encouraged to find more general proofs by Dacorogna [CITATION NEEDED] and Adams [CITATION NEEDED]. In particular, the assumption that u is continuous simplifies matters by a lot.*

We will prove the lemma by contradiction; the idea being that if u is continuous then there must exist some neighbourhood around a point x_0 where $u(x_0) \neq 0$ where the sign is constant. This property should then lead us to a contradiction.

Proof. Suppose the statement is not true. Then,

$$u(x_0) \neq 0 \quad \exists x_0 \in \Omega\ .$$

Since u is continuous there exists a neighbourhood $U \subset \mathbb{R}^n$ around x_0 s.t. w.l.o.g. on the sign

$$u(x) > 0 \quad \forall x \in U\ .$$

If we choose $\phi \in C_c^\infty(U)$ s.t. $\phi(x) > 0$, then we have

$$\int_\Omega u(x)\phi(x)\ dx = \int_U u(x)\phi(x)\ dx \neq 0\ ,$$

which clearly violates the hypothesis. ■

Remark 3.2.15. *$\phi \in C_c^\infty(U)$ is another way of saying that ϕ vanishes elsewhere that isn't U.*

With these results in mind, let us derive the aforementioned condition (Equation 3.10) by taking the first variation. Thus,

$$
\begin{aligned}
\delta F &= \Big[\frac{\partial}{\partial \epsilon}\int_\Omega \mathfrak{L}(x, u+\epsilon\delta u, \nabla(u+\epsilon\delta u))\ dx\Big]_{\epsilon=0} \\
&= \Big[\frac{\partial}{\partial \epsilon}\int_\Omega L(x, u+\epsilon\delta u) + \lambda R(x, \nabla(u+\epsilon\delta u))\ dx\Big]_{\epsilon=0} \\
&= \Big[\int_\Omega \frac{\partial}{\partial \epsilon}\big[L(x, u+\epsilon\delta u) + \lambda R(x, \nabla u + \epsilon\nabla\delta u)\big]\ dx\Big]_{\epsilon=0} \\
&= \int_\Omega L_z(x,u)\delta u + \lambda R_\xi(x, \nabla u)\cdot\nabla\delta u\ dx \\
&= \int_\Omega L_z(u)\delta u + \lambda R_\xi(\nabla u)\cdot\nabla\delta u\ dx \\
&= \int_\Omega L_z(u)\delta u\ dx - \lambda\int_\Omega \nabla\cdot R_\xi\delta u\ dx + \int_{\partial\Omega}\delta u R_\xi\cdot\mathbf{n}\ dS \\
&= \int_\Omega \Big[L_z(u) - \lambda\nabla\cdot R_\xi\Big]\delta u\ dx + \int_{\partial\Omega}\delta u R_\xi\cdot\mathbf{n}\ dS\ .
\end{aligned}
$$

By the Fundamental Lemma of the Calculus of Variations and assuming *natural boundary conditions* we arrive at the optimality condition:

$$
L_z(\bar{u}) - \lambda\nabla\cdot R_\xi(\nabla\bar{u}) = 0 \tag{3.15}
$$

$$
R_\xi(\bar{u})\cdot\mathbf{n}\mid_{\partial\Omega} = 0\ , \tag{3.16}
$$

for a minimum $\bar{u}$ of $F[u]$. It is the preceding problem of finding $\bar{u}$ which we are concerned about. The first equation is clearly the associated Euler-Lagrange equation for the supervised learning energy functional.

Remark 3.2.16. *It is important to note that we have omitted any talk about the function space X where u belongs to or the properties of L and R. In particular, the experienced reader in this area, will note that only under certain hypotheses will the solution of the preceding condition be a minimizer of $F[u]$. For this reason, we present a more formal mathematical analysis in the chapter "Supervised Learning Under the PDE Framework ".*

3.3 Interpretation

Now that we have established the problem to solve, it follows that we should attempt to understand and interpret the condition of optimality which we have just derived. Admittedly, although the optimality condition aids immensely in the quest for the search of the desired function, it doesn't help as much in understanding the "physical" meaning of the model. Granted, it is only after we have used our powerful mathematical tools and reduced the problem to one single equation, that we may now try to understand its physical meaning and gain a deeper understanding of the problem. For this purpose, let us assign meanings to the each of the components and clarify any doubts about the model. First, we clarify the meanings of the principal entities of the model. In a similar

manner, the lower half of the interpretations define the more complex components of our scheme.

The Input Vector Let $x \in \Omega \subset \mathbb{R}^n$ be interpreted as the *input or sample* which represents information about an entity that it models. Each column in particular describes a *feature* of this very same entity. The number of inputs in our dataset is finite. Nevertheless, the inputs can belong to a "continuum" and thus, the number of possible inputs can quite clearly be infinite.

The Target Function Let $y : \Omega \to \mathbb{R}$ then $y = y(x)$ is interpreted as the *target function and ground truth* of the underlying relation between x and y. The actual definition or form of the function is unknown and would like to be approximated or found. Fortunately, we are provided with a finite set of pairs which are of the form $(x, y(x))$ which aid us in identifying this relation.

The Sought After Function Let $u : \Omega \to \mathbb{R}$ then $u = u(x)$ is interpreted as the *sought after function*. This function is that we would like to approximate based on the data that has been given to us so as to predict the value of possibly new inputs x. This function needs to generalize appropriately, while making as few errors as possible when predicting the new inputs. The structure of u is unknown beforehand, i.e. we don't know if its linear, polynomial, smooth, etc.

The Structure Component Let $\xi \in \mathbb{R}^n$. This component contains information about the actual *structure of* u and its value changes at each point of u. For our model, we evaluate ξ as ∇u which contains information about the structure of u. In fact, many other problems in the PDE literature which concern themselves with finding a function with certain properties in its structure [CITATION NEEDED] can be formulated in terms of its partial derivatives. Examples include *Dirichlet's principle*, *minimal surfaces*, *Poisson's equation*, and others commonly found in the literature. This structure component is the one that will allow us to characterize and compute the complexity of u.

The Loss Function Let $L : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ be interpreted as the *loss function*. This function shall be the component which is responsible for making sure we make the fewest amount of errors. Generally speaking, this function characterizes the "error" of choosing u by comparing it against the target function y. It shall be this function, along with the complexity of u which will be minimized.

The Complexity / Regularization Function Let $R : \mathbb{R}^n \to [0, \infty)$. We interpret $R = R(u)$ as the *complexity of* u at a point x. It is rather interesting to note that the term "complexity" is actually a very subjective term. In fact, it depends on how one

decides to characterize it and so intuitively there is no "best" general answer for each and every problem. While an individual might define complexity as *the total variation* of the function, another might define it as *the total curvature* of the function. Thus, the choice R should be decided on the desired structure of the sought after function. It is clear the minimization of the complexity is desired according to *Occam's razor*.

The Change in Error for a Choice of u Let $L_z = L_z(u)$. We interpret this term as the *change in error for a choice of* u. This term represents how much the error change for a particular choice of u and thus, intuitively, for every possible choice of u we can calculate how much the loss changes along a "direction" u. Therefore, it should be possible to minimize the loss by finding a $\bar{u}$ at which $L_z(\bar{u}) = 0$, i.e. the change in error among different choices of u has arrived at a minimum with appropriate conditions on the loss function such as convexity. It is interesting to note that when talking about locally convex topological spaces, such as Banach spaces, our derivative takes on the form of the Gâteaux derivative and so our interpretation is well defined. In particular, we say that $L_z(u)$ is the derivative of $L(z)$ with respect to z evaluated at u.

The Change in Complexity with Respect to the Structure Let $R_\xi = R_\xi(\nabla u)$. We interpret this term as the *change in complexity with respect to the structure of* u. In particular, we say that $R_\xi(\nabla u)$ is the derivative of $R(\xi)$ with respect to ξ evaluated at ∇u. This operation is actually defined as the derivative of an scalar with respect to a vector. Clearly then, we are taking the gradient of $R(\xi)$ and subsequently evaluating at ∇u. Like the preceding component, when varying u, then ∇u will behave differently.

The Change in Complexity Generated at x Let $\nabla \cdot R_\xi$. We interpret this term as the *change in complexity generated at a point* x. Since the

Chapter 4

A New Variational Model for Binary Classification

Until now we have not yet specified any choices of L and R to our model or how the Euler-Lagrange equation may be used to find a function which can be used to classify binary data. It is now necessary to show how the previous results may be used. Thus, in this chapter we aim to find the three components that are needed to fully define the model **??**. Namely, these components are: u, L and R.

4.1 Radial Basis Function Approximation

Ordinarily, to approximate the function u it is required that we specify what kind of form it takes. Radial basis function (RBF) approximation relies on the idea that $u(x)$ and can be expressed as a weighted sum of radial basis functions $\{\phi_i(x)\}$, where the weights are the fixed parameter vector $\mathbf{w}$. In particular, the Gaussian RBF kernel is probably the most well known and the most widely used. For this reason only, it shall be our choice all along this work. Thus, bringing the previous ideas together, we write

$$u(x) = \sum_{i=1}^{N} w_i \phi(x_i), \tag{4.1}$$

where $\{\phi_i(x)\}$ will be a set of Gaussian RBF kernels given by

$$\phi_i(x) = e^{-c||x-x_i||^2} \tag{4.2}$$

Maintaining the notation we have been using throughout this work, we say that the $\{x_i\}$ are the observations of the input variable X. In addition, c is a positive constant that we are free to choose, while $||\cdot||$ is the Euclidean norm. We will call c the *fitting degree* of our model. As a small note, notice that by using our approximation of $u(x)$, we are setting the *centers* of the RBFs as the observations. Intuitively, this will allow our model to make predictions by calculating the Euclidean distance between a point of which the class is known, and a new input which its class is unknown while assigning

a "weight" to each point. This last step is akin to saying that some points will be more important than others when making the predictions.

4.2 Loss Function

Up until now, we have discussed the solution for the supervised learning problem without making assumptions about which context we are employing; either regression or classification. For us to correctly employ a classification setting, it is necessary that we specify an appropriate loss function. To this end, we may adopt a probabilistic setting where we make decisions on the class of the object based on the probability that said object belongs to the specified class. Namely, we may define two probabilities; the probability that x belongs to class 1, and the probability that it belongs to class 0. For this purpose, suppose we have the Bernoulli variable $Y \in \{0, 1\}$ which is a function of the random variable X. *I.e.* $Y = u(X)$.

Our goal is to predict the target class y of an input $u(x)$. This setup will allow us to use the previously mentioned probabilities to fit u and to classify x. To make these ideas more precise, take the following definitions. Let

$$Pr(y = 1|\ u(x), \mathbf{w}) = \sigma(u(x))\ , \tag{4.3}$$

be the probability that the output from $u(x)$ is classified as the class $y = 1$. On the other hand, let

$$Pr(y = 0|\ u(x), \mathbf{w}) = 1 - \sigma(u(x)) \tag{4.4}$$

be the probability that $u(x)$ is classified as $y = 0$. The function $\sigma(u)$ is known as the *sigmoid function* which is defined as

$$\sigma(u) = \frac{e^u}{1 + e^u} \tag{4.5}$$

In equations 4.4 and 4.3, $\mathbf{w}$ is a parameter vector on which $u(x)$ depends. Recall that from the previous section this is explicitly the case. Namely, the radial basis function approximation depends on a parameter $\mathbf{w}$ in order to predict an input x. The use of $u(x)$ being an input to the sigmoid is similar to its use in neural networks, where usually the input to the sigmoid might be regarded as $u(x) = \langle x, \mathbf{w} \rangle$ and, likewise, we solve for one set of weights $\mathbf{w}$ [CITATION NEEDED]. The use of $u(x)$ depending on $\mathbf{w}$ might be regarded as a generalization of this particular case. The definitions of 4.4 and (4.3) above will allow us to work towards a useful loss function for binary classification. Granted, we will approach the problem of finding a loss function from a probabilistic point of view while solving using optimization and functional analysis techniques. Consequently it will then be of interest to maximize the likelihood that, given parameters $\mathbf{w}$, the model results in a prediction of the correct class for each input sample with the likelihood being a function of the parameters. In other words we want

to examine the maximum likelihood that the inputs u_i to the sigmoid function produce a correct classification. The likelihood may be written as the joint density function of all the observations parametrized by some parameters $\mathbf{w}$. Observing that the x_i are *i.i.d.* then the joint density will be the product of all the margin densities. This may be written as

$$\mathcal{L}(\mathbf{w}) = \mathcal{L}(\mathbf{w};\ y, u) = Pr(y_1, \ldots, y_N; u_1, \ldots, u_N|\ \mathbf{w}) = \prod_{i=1}^{N} Pr(y_i, u_i|\ \mathbf{w}), \tag{4.6}$$

where $u_i = u(x_i)$. As a result of conditional probability, we have that the joint probability of y_i and u_i is proportional to the probability of y_i given u_i:

$$Pr(y_i, u_i|\ \mathbf{w}) = Pr(y_i|\ u_i, \mathbf{w}) Pr(u_i|\ \mathbf{w}) \tag{4.7}$$

$$\Rightarrow Pr(y_i, u_i|\ \mathbf{w}) \propto Pr(y_i|\ u_i, \mathbf{w}). \tag{4.8}$$

Therefore, we can equivalently solve the maximization problem

$$\arg\max_{\mathbf{w}} \prod_{i=1}^{N} Pr(y_i|\ u_i, \mathbf{w}). \tag{4.9}$$

Maximizing explicitly this function, however, might prove cumbersome and so we, like many others, opt to maximize the log-likelihood. In fact, since the logarithmic function is monotone increasing, maximizing the likelihood is equivalent to maximizing the log-likelihood. Expanding out the log-likelihood, we are left with

$$\ln \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \ln Pr(y_i|\ u_i, \mathbf{w}). \tag{4.10}$$

Recalling that Y is a Bernoulli variable, we will have

$$Pr(y_i|\ u_i, \mathbf{w}) = Pr(y_i = 1|\ u_i, \mathbf{w})^{y_i} \big[1 - Pr(y_i = 1|\ u_i, \mathbf{w})\big]^{1-y_i} \tag{4.11}$$

$$= \sigma(u_i)^{y_i} (1 - \sigma(u_i))^{1-y_i}. \tag{4.12}$$

Undoubtedly, we can see that

$$\ln Pr(y_i|\ u_i, \mathbf{w}) = \ln \left[\sigma(u_i)^{y_i} (1 - \sigma(u_i))^{1-y_i}\right] \tag{4.13}$$

$$= y_i \ln \sigma(u_i) + (1 - y_i) \ln \big(1 - \sigma(u_i)\big). \tag{4.14}$$

Moreover, to maximize the log-likelihood, we can equivalently minimize the negative of the log-likelihood. These results translate over to the problem **??** and one may simply set the loss function as the negative log-likelihood:

$$L(\sigma(u), y) = -\Big[y \ln \sigma(u) + (1 - y) \ln \big(1 - \sigma(u)\big)\Big]. \tag{4.15}$$

We will call Equation 4.15 the *cross entropy* of the distributions of $\sigma(u)$ and y. The

reason why we write $L(\sigma(u), y)$ instead of simply $L(u, y)$ is to express the idea that a loss function used for binary classification will utilize the probability distribution $\sigma(u)$. The loss function would then take as input any value of $u(x)$ and transform it into an approximation of the probability that it belongs to the class. We gain confidence that our model is appropriate by supposing the variable is Bernoulli and formally deriving the loss function. Note that if we were to use this for regression, we would be changing the semantics of the context since a regression variable is definitely not a Bernoulli variable.

In order to solve the variational problem described in Chapter 3, it is necessary to obtain the derivative for the loss function with respect to u. This is a technical matter and so the expression can be found in Appendix A. With this in mind, the derivative of the loss function is

$$\frac{dL}{du} = \sigma(u) - y. \tag{4.16}$$

Before proceeding to give the whole model, we shall give the regularization term that will be used in the next section.

4.3 Laplacian Regularization: $-\Delta u$

Now that we have derived an appropriate loss function for binary classification, we are tasked with the choice of a regularizer term. Much work has been done in the past to find a good regularization term and although there is no clear best choice, we certainly can make do with most suggestions in the literature. In the recent work on this area, Tong Lin *et al.* [CITATION NEEDED] have proposed the square of the gradient vector norm $||\nabla u||$ in the regression context, so that

$$S(u) = \frac{1}{2}\int_{\Omega} ||\nabla u||^2 dx. \tag{4.17}$$

Obtaining an expression for the term that depends on the regularizer in Equation 3.10 becomes a technical matter. Skipping the details (Appendix A), the resulting expression is given by:

$$\frac{d}{du}\frac{||\nabla u||^2}{2} - \nabla \cdot \frac{\partial}{\partial \nabla u}\frac{||\nabla u||^2}{2} = -\Delta u. \tag{4.18}$$

The idea behind this choice of regularization term consists in the punishment of sharp edges of the function u. Another way to look at it is to regard $u(x)$ as scalar field and to notice that taking the norm of a vector is interpreted as calculating its magnitude. Thus, this choice minimizes the slope of the gradient points in the direction of the greatest rate of increase. In part, this will ensure that the u that is found will be smooth and reduce overfitting. As a side note, we can see that the functional in Equation 4.17 is convex and thus our solution will be unique. Equation 4.17 is also known as the Dirichlet Energy functional [CITATION NEEDED], and so we can have a more precise

interpretation of minimizing this functional. Chiefly, the Dirichlet Energy measures how variable a function is and it is a quadratic functional on the Sobolev space $W^{k,2}$. In general, this regularization term has been tried with favourable results [CITATION NEEDED].

One might ask why the term $||\nabla\sigma(u)||^2$ was not utilized instead. To illustrate the reasoning behind our choice, note that $\sigma(\cdot)$ is a fixed function and so the values that $\sigma(u)$ might take will only vary in respect to u. On the contrary, u can vary independently and may behave erratically, thus will require regularization. The convention to regularize explicitly u also permits us to derive simple, as opposed to lengthy and complex expressions.

4.4 The Model

This section and the next can be thought as the climax of the previous sections in the chapter and so, without further ado, we shall give the whole model along with its solution. Let $u : \Omega \subset \mathbb{R}^m \to \mathbb{R}$ be the function we want to fit and let $y \in \{0, 1\}$ be the target values. Let $F[u]$ be a functional depending on u, defined by

$$F[u] = \int_\Omega -[y \ln \sigma(u) + (1 - y) \ln \big(1 - \sigma(u)\big)] + ||\nabla u||^2 \, dx. \tag{4.19}$$

Then the condition of optimality is the elliptic PDE

$$\sigma(u) - y - \Delta u = 0. \tag{4.20}$$

In order to solve the problem numerically, we can isolate the variable y to see that we have reduced the original problem to a simpler sub-problem; one that can be solved numerically. This can be achieved by recalling that $u = u(x)$ depends on a fixed weight vector $\mathbf{w}$ and can be expressed by the sum of the set of RBFs $\{\phi_i(x)\}$. Therefore, it is required to find the weight vector $\mathbf{w}^*$ which appropriately satisfies Equation 4.21. Furthermore, we can regard the problem as a fitting problem. That is, we want to fit the function on the left hand side, to the constant targets which are on the right hand side.

$$\underbrace{\sigma(u) - \Delta u}_{\text{Function to fit}} = \underbrace{y}_{\text{Targets}} \tag{4.21}$$

Since we have N datum pairs (x_i, y_i) we may attempt to utilize these in order to find the function, or equivalently the weights $\mathbf{w}^*$, which satisfy the equality above. It is then that we may write the problem of finding $\mathbf{w}^*$ as a least squares problem (LSQP).

Let $g(x_i, \mathbf{w}) = \sigma(u(x_i, \mathbf{w})) - \Delta u(x_i, \mathbf{w})$, then the LSQP is:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \left[y_i + \Delta u(x_i, \mathbf{w}) - \sigma(u(x_i, \mathbf{w}))\right]^2 \tag{4.22}$$

$$= \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \left[y_i - g(x_i, \mathbf{w})\right]^2 \tag{4.23}$$

$$= \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \left[y_i - g_i(\mathbf{w})\right]^2 \tag{4.24}$$

$$= \arg\min_{\mathbf{w}} S(\mathbf{w}), \tag{4.25}$$

Note we have defined $g_i(\mathbf{w}) = g(x_i, \mathbf{w})$ to obtain the third step. The matter is therefore reduced to picking or designing a numerical method to solve the problem above as we shall see in the next section.

4.5 Training

With great anticipation, we arrive at the training stage. In the previous section, we found out that the condition of optimality expressed the problem of fitting a function to the targets. We argued that finding $u = u(x)$ is achieved by finding a numerical method that could solve the problem for $\mathbf{w}^*$. Due to the non-linearity of the function, we propose that the method be solved by an iterative method; namely the Levenberg-Marquardt (LM) algorithm [CITATION NEEDED]. The LM algorithm consists in estimating $\mathbf{w}^*$ by following a sequence of estimations, each better than the previous one, so that $\mathbf{w}^{(j+1)} = \mathbf{w}^{(j)} + \boldsymbol{\delta}$ and therefore $\mathbf{w}^{(n)} \to \mathbf{w}^*$ as $n \to \infty$. We can determine $\boldsymbol{\delta}$ by looking at the linearisation of $g_i(\mathbf{w} + \boldsymbol{\delta}) = g(x_i, \mathbf{w} + \boldsymbol{\delta})$ which is given by

$$g_i(\mathbf{w} + \boldsymbol{\delta}) \approx g_i(\mathbf{w}) + \nabla g_i(\mathbf{w})\boldsymbol{\delta} \tag{4.26}$$

$$\nabla g_i(\mathbf{w}) = \nabla_{\mathbf{w}} g(x_i, \mathbf{w}). \tag{4.27}$$

Equation 4.27 is the gradient of g with respect to $\mathbf{w}$. By looking at the expression of the linearisation of the function, we can expand it (Appendix B) to get

$$S(\mathbf{w} + \boldsymbol{\delta}) \approx \sum_{i=1}^{N} (y_i - g_i(\mathbf{w}) - \nabla g_i(\mathbf{w})\boldsymbol{\delta}) \tag{4.28}$$

$$= ||\mathbf{y} - \mathbf{g}(\mathbf{w}) - \boldsymbol{J}\boldsymbol{\delta}||^2 \tag{4.29}$$

$$= \dots \tag{4.30}$$

$$= [\mathbf{y} - \mathbf{g}(\mathbf{w})]^T[\mathbf{y} - \mathbf{g}(\mathbf{w})] - 2[\mathbf{y} - \mathbf{g}(\mathbf{w})]^T\mathbf{J}\boldsymbol{\delta} + \boldsymbol{\delta}^T\mathbf{J}^T\mathbf{J}\boldsymbol{\delta}. \tag{4.31}$$

Taking the derivative of 4.31 with respect to $\boldsymbol{\delta}$, we see that

$$(\mathbf{J}^T\mathbf{J})\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{g}(\mathbf{w})] \tag{4.32}$$

must be satisfied. This last line can also be interpreted as "solve for $\boldsymbol{\delta}$". It is desirable, however, that this method converge smoothly to a solution and so LM added a damping factor to the equation above. Therefore the LM algorithm at each step will solve a damped version of the previous equation:

$$[\mathbf{J}^T\mathbf{J} + \eta \operatorname{diag}(\mathbf{J}^T\mathbf{J})]\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{g}(\mathbf{w})], \tag{4.33}$$

where η is a damping parameter which may be fixed or calculated at each iteration. Finally, we can give the algorithm to train our model along with the mathematical expressions used.

Algorithm 1: Training the model (Levenberg-Marquardt)

Data:

- x such that $x = (x_1, \ldots, x_N)^T$ is the matrix of observations
- λ is a regularization parameter
- η is a dampening parameter
- M is the number of iterations

Result: The weight vector $\mathbf{w}^*$

begin

$i \longleftarrow 0$;

$w^{(0)} \longleftarrow (0, \ldots, 0)^T$;

while $i < M$ **do**

Solve $[\mathbf{J}^T\mathbf{J} + \eta \operatorname{diag}(\mathbf{J}^T\mathbf{J})]\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{g}(\mathbf{w})]$ for $\boldsymbol{\delta}$

$w^{(i+1)} \longleftarrow w^{(i)} + \boldsymbol{\delta}$

We now proceed to give the expressions used in the algorithm. Their derivations can be consulted in Appendix B.

$$\begin{aligned}
\mathbf{y} &= [y_1, \ldots, y_N] \\
\mathbf{g}(\mathbf{w}) &= [g_1(\mathbf{w}), \ldots, g_N(\mathbf{w})]^T \\
\mathbf{J} &= [\nabla g_1(\mathbf{w}), \ldots, \nabla g_N(\mathbf{w})]^T \\
\nabla g(\mathbf{w}) &= \nabla_{\mathbf{w}}\sigma(u) + \nabla_{\mathbf{w}}\Delta u \\
\nabla_{\mathbf{w}}\sigma(u) &= \sigma(u)(1 - \sigma(u))[\phi_1, \ldots, \phi_N]^T \\
\nabla_{\mathbf{w}}\Delta u &= [\Delta\phi_1, \ldots, \Delta\phi_N] \\
\Delta\phi_i &= c(c||x - x_i||^2 - m)\phi_i
\end{aligned}$$

4.6 Evaluation Methodology

We tested our model against 9 binary datasets. For each dataset we calculated two metrics:

- Accuracy
- Area under the ROC curve (AUC) [13]

Both of the metrics are calculated over the test set in a 5-*fold* cross validation scheme. There has been minimal preprocessing of the datasets. That is, we have centered to 0 mean and unit variance the features of each dataset. Namely, we have *standarized* the dataset and most notably no dimensionality reduction has been applied.

When training the models, it is necessary to specify the parameters in which they depend on. For our model (LR) we have two choose a parameter triplet (c, λ, η); c is the fitting degree, λ is the regularization parameter and η is a dampening parameter. For a *RBF-kernel* SVM, we choose the parameter pair (C, γ); C is the penalty parameter of the error term while γ is the kernel coefficient. Lastly, we assume that NN is a *multi-layer perceptron* with one hidden layer of 100 nodes which depends on a regularization term α. Both SVM and NN are implementations of the Python library `sklearn` [14]. The parameters for each of our models were chosen by the following methodology:

LR. The parameter triplet (c, λ, η) is searched on a grid:

- c is searched on the interval $(0, 5)$ with step $= \ln 2$.
- λ is searched on the interval $[0, 10]$ with step $= 1$.
- η is fixed to $\eta = 1$ for each and every dataset.

The value of $\eta = 1$ was found empirically to work well with almost any value of λ and c.

SVM. The parameter pair (C, γ) is searched on a grid:

- C is searched on the interval $(0, 5)$ with step $= \ln 2$.
- γ is fixed to $\gamma = 1/m$, where m is the number of features of the dataset.

Originally γ was searched on the interval $[0, 10]$ with step $= 1$. It was soon found empirically, that fixing γ to $\gamma = 1/m$ resulted in better performance.

NN. The parameter α was fixed to $\alpha = 0.001$.

Table 4.1: The accuracy (%) of each method is outlined in this table. The second to last column indicates which ranking (1st, 2nd or 3rd) LR obtained; higher is better. On the other hand, the last column indicates the absolute value of the residual between LR and the 1st place. In bold, the best accuracy.

Data	Dim	N	LR	SVM	NN	Place	Dist.
Australian	14	690	86.6667	86.8116	**87.8261**	3rd	1.1594
Blood Transfusion	4	748	**78.2246**	78.2237	77.2859	1st	0.0
Breast Cancer	30	569	97.7146	**98.7425**	98.2673	3rd	1.0279
Bupa	6	345	**72.4638**	**72.4638**	71.0145	1st	0.0
German	24	1000	76.0000	76.6000	**78.3000**	3rd	2.3
Haberman	3	306	73.5431	73.8710	**74.5267**	3rd	0.9836
Heart	13	270	82.2222	**84.8741**	84.0148	3rd	2.6519
Sonar	60	208	88.4321	**88.9199**	87.4681	2nd	0.4878
Vertebral Column	6	310	**86.7742**	85.4839	83.8710	1st	0.0
					Average distance from 1st:		0.9567

4.7 Results and Analysis

This section presents the results in Tables 4.1 through 4.3. For us to comprehend more easily the results obtained, we have arranged and summarized the results into various tables. In terms of accuracy, LR outperformed SVM and NN on 3 datasets. Specifically, LR outperformed NN on 4 datasets and outperformed SVM on 2, tying on the *Breast Cancer* dataset. To fully grasp how much better or worse our method has performed we calculated the absolute value of the residual between LR and the top performer for each dataset. On average we see that our method was down by 0.9567 %. Although not shown in Table 4.1, the average distance from the top performer for SVM is 0.5229 % and for NN it is 0.8975 %. These scores lend to the interpretation of "which method got closer to the real solution"; looking at who got the top score we might say that the top performer was the best *method suited for that particular type of dataset.* It is of interest then to evaluate a method which in average should perform well for most types of datasets. Interpreting the results, we can confidently say that **SVM is the best method, while NN is second and LR comes a close third in terms of accuracy**.

The AUC score may also be used to further determine the performance of a classifier [15]. Proceeding in a similar manner as before, we calculate the average residual between the 1st place and LR. We find that LR is on average down from 0.0126 units from the top performer for each dataset. The same is calculated for SVM and NN; respectively 0.0088 and 0.0161. Surprisingly, even though NN outperformed SVM and LR on 4 different datasets while LR only outperformed the others on one, **on average LR will perform better than NN.** Unsurprisingly, SVM will still perform better than NN and LR. These results may be observed in Table 4.2.

Table 4.2: The area under the ROC curve has been calculated for each method over each dataset Also, we have added a grade next to the scores to see how they perform against each other more easily; A being excellent performance, while F is catalogued as a fail. Likewise (Table 4.1), the second to last column indicates which ranking (1st, 2nd or 3rd) LR obtained; higher is better. On the other hand, the last column indicates the absolute value of the residual between LR and the 1st place. In bold, the best AUC.

Data	LR	SVM	NN	Place	Dist.
Australian	0.8643 (B)	0.8701(B)	**0.8777** (B)	3rd	0.0134
Blood Transfusion	0.5844 (F)	**0.6144** (D)	0.5502 (F)	2nd	0.03
Breast Cancer	0.9729 (A)	0.9800 (A)	**0.9858** (A)	3rd	0.0129
Bupa	0.7024 (C)	**0.7059** (C)	0.6866 (D)	2nd	0.0035
German	0.6821 (D)	0.6920 (D)	**0.7058** (C)	3rd	0.0237
Haberman	**0.5560** (F)	0.5559 (F)	0.5463 (F)	1st	0.0
Heart	0.8180 (B)	0.8452 (B)	**0.8322** (B)	3rd	0.0142
Sonar	0.8857 (B)	**0.8906** (B)	0.8812 (B)	2nd	0.0049
Vertebral Column	0.8292 (B)	**0.8404** (B)	0.7978 (C)	2nd	0.0112
				Average distance from 1st:	0.0126

Our analysis has been very exact so it now time to present a more intuitive analysis based on the AUC. Furthermore, this analysis is more *robust* than the previous. The analysis consists in assigning a "grade" to a classifier by specifying the following grading scheme:

$$\text{Grade} = \begin{cases} \text{Excellent } (A) & 0.9 \leq \text{AUC} \leq 1 \\ \text{Good } (B) & 0.8 \leq \text{AUC} < 0.9 \\ \text{Fair } (C) & 0.7 \leq \text{AUC} < 0.8 \\ \text{Poor } (D) & 0.6 \leq \text{AUC} < 0.7 \\ \text{Fail } (F) & 0.5 \leq \text{AUC} < 0.6 \end{cases}$$

The "robustness" comes from the fact that we are partioning discretely the interval $[0, 1]$ and assigning each a grade. Small variations within the sub-intervals will be neglected. Looking at Table 4.2 we see that each of the AUC scores has a letter assigned to it. This is interpreted as the grade which the method received on that particular dataset. In order to summarize the grades, we arrange the number of times a method received a particular grade in Table 4.3. Simply by looking at Tables 4.2 and 4.3 we can get the sense that all of the models performed similarly. In order to obtain a quantitative measure, we may assign each grade a value. Namely, A = 1, B = 2, ..., F = 5. Obtaining the weighted total of the grades will then let us asses directly which method is better by looking at the lowest total. The results are presented in Table 4.3. Immediately, we see that SVM once again obtained the best score. This time, however, LR came a close second with only a 1 point difference. On the contrary NN was down by 4 points from SVM and 3 points from LR. Summarizing, **LR outperforms NN**

Table 4.3: The columns LR, SVM and NN indicate the numbers of times each of the methods got the grade on the leftmost column. Letting A = 1, B = 2, C = 3, D = 4, F = 5, we can calculate a weighted final grade for our classifiers and see how each of them performed. Clearly a lower grade is better. The weighted grade is just the weighted total of the grades each of these methods obtained.

Grade	LR	SVM	NN
A	1	1	1
B	4	4	3
C	1	1	2
D	1	2	1
F	2	1	2
Weighted grade (lower is better):	26	**25**	29

again while SVM remains overall the best method, indicating that NN is very prone to overfitting while LR and SVM are not.

Chapter 5

Towards Higher Order Derivatives in Regularization Terms

Most times, it is of much interest to explore higher order terms in equations or formulas in order to obtain a deeper understanding of the underlying structure. Our general intuition allows us to reason that if we allow more complex terms to take part in our model it will perform better since it should be able to admit more complex characteristics. Thus, it is of interest to see what the optimality condition derived in Chapter 3 would look like if we allowed higher order derivatives in the regularization term. In this chapter we aim to find the associated Euler-Lagrange equation, which will pave the way for the choice of loss and regularization functions.

5.1 Regularization Terms using Higher Order Derivatives

Similarly to the first derivation in Chapter 3, we would like to approximate a function $u : \Omega \subset \mathbb{R}^m \to \mathbb{R}$ using a loss functional $L(u)$ and a regularization functional $R(u)$. In order to deal with n dimensions and derivatives of higher order, we will have to be careful about how we write out the problem. Due to the dimensionality and order it will make sense to employ *multi-index notation* which, for the most part, will be adapted from the book *Partial Differential Equations (Appendix A: Notation)* by Lawrence C. Evans in concordance with the notation we have used throughout this manuscript.

To this end, let $u : \Omega \subset \mathbb{R}^n \to \mathbb{R}$. Let α be a *multi-index* defined by a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \geq 0$ of *order*

$$|\alpha| := \sum_{i=1}^{n} \alpha_i \ . \tag{5.1}$$

Furthermore, for a multi-index α, define

$$D^\alpha u(x) := \frac{\partial^{|\alpha|} u(x)}{\partial x_1^{\alpha_1} \ldots \partial x_n^{\alpha_n}} = \frac{\partial}{\partial x_1^{\alpha_1}} \cdots \frac{\partial}{\partial x_n^{\alpha_n}} u(x) \tag{5.2}$$

Finally, define

$$\nabla^k u(x) := \{D^\alpha u(x) \ : | \ |\alpha| = k, \ k \geq 0\} \ . \tag{5.3}$$

By assigning an ordering to $\nabla^k u(x)$, we may regard it as a point of $\mathbb{R}^{n^k}$. We may quickly gain some insight as to the choice of the ordering by specifying our own; that is we may visualize the particular cases of $k = 0, 1, 2$. The first case corresponds to the identity of u itself

$$\nabla^0 u = u, \tag{5.4}$$

while the next case clearly corresponds to a vector

$$\nabla^1 u = \text{grad } u = \nabla u = (\frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_m}) \ , \tag{5.5}$$

which, unsurprisingly, is the gradient. Similarly, for $k = 2$, by regarding it as a matrix of sorts

$$\nabla^2 u = \mathbf{H_u} = \begin{bmatrix} \dfrac{\partial^2 u}{\partial x_1^2} & \dfrac{\partial^2 u}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 u}{\partial x_1 \partial x_m} \\ \dfrac{\partial^2 u}{\partial x_2 \partial x_1} & \dfrac{\partial^2 u}{\partial x_2^2} & \cdots & \dfrac{\partial^2 u}{\partial x_2 \partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 u}{\partial x_m \partial x_1} & \dfrac{\partial^2 u}{\partial x_m \partial x_2} & \cdots & \dfrac{\partial^2 u}{\partial x_m^2} \end{bmatrix} \ , \tag{5.6}$$

we find that it corresponds to the Hessian matrix. Higher order terms are harder to visualize, thus, we may facilitate the conceptualization of ∇^k if we regard it as a container of all the k-th order partial derivatives of the function u.

5.2 The Original Model for Higher-Order Derivatives

Now that we have defined the meaning of the differential operators that will be used, we may state the problem in its entirety. Recall that we would like to find the minimum of a functional $F[u]$ of the form

$$F[u] = \int_\Omega f(\mathbf{x}, u, \nabla u, \nabla^2 u, \ldots, \nabla^k u) \ d\mathbf{x} \ . \tag{5.7}$$

where f is called the Lagrangian. In our case, however, the Lagrangian takes the form

$$f = \underbrace{L(u)}_{\text{Punishes errors}} + \underbrace{R(u, \nabla u, \ldots, \nabla^k u)}_{\text{Controls the complexity of u}} \ . \tag{5.8}$$

As we saw in previous chapters , the intuition behind this choice of model lies on the two basic components of most, if not all, supervised learning frameworks. That is, we specify a penalty term which "punishes errors" (the loss function) while we specify another function which will "control the complexity" of the desired function (the regularization term). This may prove advantageous for building specific classifiers which satisfy certain hypotheses where our regularization term serves as a proxy to

implement these hypotheses. For example, we may recall the basic model we built in the previous chapters where we opted to choose

$$R(\mathbf{x}, u, \nabla u) = \frac{1}{2}||\nabla u||^2 \ . \tag{5.9}$$

Substituting this choice in equation 3.5 leads to a specific version of the problem

$$\min_u \int_\Omega L(u, y) d\mathbf{x} + \lambda \frac{1}{2} \int_\Omega ||\nabla u||^2 d\mathbf{x} \ , \tag{5.10}$$

where if we focus solely on the right-most side of the equation we find that it is the Dirichlet Energy which we are minimizing alongside the loss of the model [1]. If we take the general interpretation of this integral as quantifying "how variable a function is", it is not surprising that we are in fact imposing a restriction over u; preventing sharp edges. Thus, it makes sense that we should eventually want to impose more complex restrictions on our desired functions. For one thing, if there were some knowing way to determine *a priori* how our u will behave we could choose a particular regularization term to impose on our desired function. Unfortunately, we shan't extend further upon this topic but we feel that it is important to mention. Instead we will add it to our list of possible future works.

5.3 Euler-Lagrange Equation for Higher Order Derivatives

Speculation aside, let us get back to the task at hand; deriving the Euler-Lagrange equation associated to the supervised problem. In this manner, we shall proceed heuristically without making too many assumptions on Ω or the domain of $F[u]$, lest we deviate too far from our goal and delve into the details of functional analysis. To this end, let us recall once again the definition of a functional derivative. That is,

$$\int \frac{\delta F}{\delta u}(\mathbf{x}) \delta u(\mathbf{x}) \ d\mathbf{x} = \lim_{\epsilon \to 0} \frac{F[u + \epsilon \delta u] - F[u]}{\epsilon} = \Big[\frac{d}{d\epsilon} F[u] + \epsilon \delta u] \Big]_{\epsilon = 0} \tag{5.11}$$

Without saying too much about the space where u belongs to, we can begin deriving an appropriate condition at which $F[u]$ attains its minimum for higher order derivatives. Similar to previous sections, we let

$$F[u] = \int_\Omega f(\mathbf{x}, u, \nabla u, \ldots, \nabla^k u) \ d\mathbf{x} \tag{5.12}$$

and seek that the following condition is met

$$\frac{\delta F}{\delta u} = 0 \tag{5.13}$$

[1] For the more formally inclined, we would need to fix $u = u_0$ for some u_0 on the boundary $\partial\Omega$

This time around, however, we are met with the difficulty that the Lagrangian contains derivatives of up to order k. Proceeding rather formally, we first apply the definition of functional derivative to our functional. That is, we calculate the functional derivative of $F[u]$:

$$\int_\Omega \frac{\delta F}{\delta u}(\mathbf{x})\delta u(\mathbf{x})\ d\mathbf{x} = \Big[\frac{d}{d\epsilon}\int_\Omega f(\mathbf{x}, u+\epsilon\delta u, \nabla u + \epsilon\nabla\delta u, \ldots, \nabla^k u + \epsilon\nabla^k\delta u)d\mathbf{x}\Big]_{\epsilon=0} \tag{5.14}$$

$$= \int_\Omega \frac{\partial f}{\partial u}\delta u + \frac{\partial f}{\partial \nabla u}\nabla\delta u + \ldots + \frac{\partial f}{\partial \nabla^k u}\nabla^k\delta u\ d\mathbf{x} \tag{5.15}$$

Bibliography

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville: *Deep learning.* MIT press, 2016.

[2] A. Singh, N. Thakur and A. Sharma, "A review of supervised machine learning algorithms," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, 2016, pp. 1310-1315.

[3] Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems." J. Mach. Learn. Res 15.1 (2014): 3133-3181.

[4] Kotsiantis, Sotiris B., I. Zaharakis, and P. Pintelas: "Supervised machine learning: A review of classification techniques." (2007): 3-24.

[5] Caruana, Rich, and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms." Proceedings of the 23rd international conference on Machine learning. ACM, 2006.

[6] Gybenko, G. "Approximation by superposition of sigmoidal functions." Mathematics of Control, Signals and Systems 2.4 (1989): 303-314.

[7] Hornik, Kurt. "Approximation capabilities of multilayer feedforward networks." Neural networks 4.2 (1991): 251-257.

[8] Shore, John, and Rodney Johnson. "Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy." IEEE Transactions on information theory 26.1 (1980): 26-37.

[9] Evans, Lawrence. "Partial differential equations." (2010).

[10] Lin, Tong, et al. "Supervised learning via Euler's Elastica models." Journal of Machine Learning Research 16 (2015): 3637-3686.

[11] Moré, Jorge J. "The Levenberg-Marquardt algorithm: implementation and theory." Numerical analysis. Springer, Berlin, Heidelberg, 1978. 105-116.

[12] Press, William H., et al. Numerical recipes in C. Vol. 2. Cambridge: Cambridge university press, 1996.

[13] Hanley, James A., and Barbara J. McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." Radiology 143.1 (1982): 29-36.

[14] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of Machine Learning Research 12.Oct (2011): 2825-2830.

[15] Fawcett, Tom. "An introduction to ROC analysis." Pattern recognition letters 27.8 (2006): 861-874.

[16] Belkin, Mikhail, Partha Niyogi, and Vikas Sindhwani. "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples." Journal of machine learning research 7.Nov (2006): 2399-2434.

[17] Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning. Vol. 1. New York: Springer series in statistics, 2001.

[18] Parr, Robert G. "Density functional theory of atoms and molecules." Horizons of Quantum Chemistry. Springer, Dordrecht, 1980. 5-15.

Appendix A

Derivation of Model Components

This Appendix concerns itself with the derivation of some useful expressions of the models. They are mostly presented to satisfy technical curiosity and rigour. This chapter, however, will not contain the derivations of the numerical algorithms. Instead, they are left until Appendix B.

A.1 Derivative of Binomial Cross Entropy Loss

The cross entropy loss between *the probability distribution of a random Bernoulli variable* $Y \in \{0, 1\}$ and *the probability of finding the output* $y = 1$ *on an input* $u(x)$ *given by* $\sigma(u)$ is

$$L(y, \sigma(u)) = -\Big[y \ln \sigma(u) + (1 - y) \ln \big(1 - \sigma(u)\big)\Big] \ , \tag{A.1}$$

where σ is the logistic function.

For us to utilize this in our model, we wish to find the expression $\frac{dL}{du}$. To this end, and to facilitate understanding, let us read the following derivation:

$$\begin{aligned}
\frac{dL}{du} &= -\frac{d}{du}\Big[y \ln \sigma(u) + (1 - y) \ln(1 - \sigma(u))\Big] && \text{(A.2)} \\
&= -y\frac{\sigma'(u)}{\sigma(u)} + (1 - y)\frac{\sigma'(u)}{\sigma(u)} && \text{(A.3)} \\
&= -y\frac{(1 - \sigma(u))\sigma(u)}{\sigma(u)} + (1 - y)\frac{(1 - \sigma(u))\sigma(u)}{1 - \sigma(u)} && \text{(A.4)} \\
&= -y(1 - \sigma(u)) + (1 - y)\sigma(u) && \text{(A.5)} \\
&= \sigma(u) - y \ . && \text{(A.6)}
\end{aligned}$$

Firstly, we substitute. Secondly, we expand and differentiate. Furthermore, we utilize the fact that $\frac{d}{du}\sigma(u) = (1 - \sigma(u))\sigma(u)$. Finally, we expand and cancel out the terms.

A.2 Derivative of Multinomial Cross Entropy Loss

Bibliography