Foundations of Machine Learning (CS5590)

Instructor: Saketh

December 20, 2022

Contents

Notations

$\mathbb{R}, \mathbb{R}_+, \mathbb{R}_{++}$	Space of reals, non-negative reals, positive reals.
$\mathbb{R}^n$	Euclidean vector space of dimensionality n
$\mathcal{X}$	Space in which input samples lie
$\mathcal{Y}$	Space in which output samples lie
m	Number of training samples
$p^*(x, y)$	(Unknown) joint likelihood of input-output pairs (x, y)
$p^*(y\|x)$	(Unknown) posterior likelihood of output y conditioned on input x
$\mathcal{D}$	Training data
(x_i, y_i)	i^{th} Pair of input-label sample in $\mathcal{D}$
(X_i, Y_i)	Random variable corresponding to (x_i, y_i)
$\mathcal{F}$	Model/Hypothesis class/Inductive bias
l	Loss function $l : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}_+$, $l : \mathcal{F} \times \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}_+$
$R[f]$	(True) Risk (of f)
f^*	Bayes Optimal (minimizer of Risk)
$f^*_{\restriction \mathcal{F}}$	Bayes optimal restricted to $\mathcal{F}$
$\hat{f}_m$	Estimate for the Bayes Optimal obtained with training with m samples
$\hat{R}_m[f]$	Empirical Risk (with f)
$\hat{f}_m^{\mathrm{ERM}}(\mathcal{F})$	Empirical Risk Minimizer in $\mathcal{F}$
$\mathcal{C}(l, \mathcal{F})$	Complexity/capacity of the loss-model combination
$\mathcal{L}_{n,\phi}$	Linear model in n-dimensional feature space of ϕ
ϕ	Feature map
k	Kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$
G	Gram matrix
Ω	Regularizer function
W	Regularization upper-bound
V_i	matrix where j^{th} column is the vector of incoming edge weights from nodes in layer
$\mathcal{N}_{n,d,q,\sigma}$	Feed Forward Neural Network (FFNN) when input $x \in \mathbb{R}^n$, depth d, width q
${}^k\mathcal{N}_d$	K-nearest network model

f_S	Labeling function in nearest neighbor, given a set S
σ	ReLU activation function
$\mathcal{B}$	Bernoulli model
$\mathcal{M}$	Multinoulli model
$\mathcal{G}_n$	Gaussian model over $\mathbb{R}^n$
$\{a_1, a_2, \ldots a_k\}$	Set of possible actions in k-arm bandits
A_t	Action taken at instant t
R_t	Reward for action at instant t
$q(a_i)$	expected reward when action a_i is taken
$Q_t(a_i)$	empirical estimate of reward in t trials when action a_i is taken
$Ti(n)$	random variable denoting the no. of times arm a_i was pulled in first n instant.

Chapter 1

Introduction

1.1 Mimicing human Learning is ML

Refer:sections 1.1 , 1.2 in Sha Shwartz Ben-David [201

We began by noting the grand goal of Machine Learning (ML): "To develop computer programs that can mimic high-level cognitive abilities in humans for solving complex problems (e.g., those in Astrophysics, Biology, Environment etc.)"[1]. The important keyword is mimic (rather than copy/replicate). This is because today's science is very far from understanding how humans acquire their cognitive abilities. Hence we merely wish to mimic/imitate cognition via sophisticated mathematical models.

To understand this goal, and to know where machines stand as of today, we present the following list of cognitive abilities:[2]:

smṛti : Memorizing concepts and recognizing objects etc. Today's computers/devices can store large amounts of data efficiently. Deep learning techniques achieve human-level performance in tasks like speech recognition[3], face recognition[4] etc. So today's machines seemingly can mimic smṛti well in multiple settings.

medha : Knowledge (know-how) acquisition/representation, summarization/deductions, etc. For example, today's computers surpass human performance in arithmetic and logical computations. Modern Information Retrieval (IR) systems and generative models seem to mimic medha in some sense.

buddhi : Comprehensive understanding and long-term (visionary) decision-making,

[1] Also read section 1.2 in Shalev-Shwartz and Ben-David [2014].
[2] The source for the Indian terminology used here is "DevīBhāgavataṃ".
[3] For example, voice-typing on your mobiles
[4] For example, Windows Hello, Apple Face ID etc.

insights/intuition etc. In a very restrictive sense, programs like AlphaGo, and techniques for domain generalization seem to mimic buddhi.

pratibha : Creativity and originality. Barring few attempts to create artwork/articles etc., today's machines do not seem to mimic pratibha.

sphuraṇa : Impromptu flash of abstract thought leading to creative and original work. Machines seems to be completely absent here!

Learning in humans is their ability to improve these cognitive abilities from experience. This important aspect is imitated in ML via so-called "training from data". Also, humans seem to have inherent biases, which in-turn determine whether their abilities are useful or not[5] (for a given task). ML models do encode "biases" with a similar effect.

Interested students may read section 1.4 in Shalev-Shwartz and Ben-David [2014] to know the relationship between ML and other subjects. Also, popular ML applications are listed in: section 1.2 in Mohri et al. [2012].

1.2 Some ML jargon

In ML, a mathematical model is what mimics a human. The process of learning in humans is imitated by so-called "training" of the model. In other words, an untrained-model becomes a trained-model through the process of training. The feedback/supervision (i.e., experience) that enables human learning is often dubbed as the training data in ML.

Humans typically have a motivation when they learn something. Likewise, training in ML is often performed with a specific goal/objective. This goal is sometimes referred to as the learning task. At times, prior information relevant to the task at hand is available. This is also called as the Domain/background knowledge or simply as the prior. The inherent "bias" in a model may aid the current task (e.g., bias is aligned with the prior), in which case the trained-model is expected to perform well in the application. Hence in this course we shall study different models, with different inherent biases, which may be useful in diverse applications.

A typical ML pipeline is presented below to better understand the jargon:

[5] While section 1.1 in Shalev-Shwartz and Ben-David [2014] shows how learning in pigeons may not be useful, `https://pubmed.ncbi.nlm.nih.gov/21965161/` argues that pigeons can solve complicated optimization problems.

1. The task (the final objective or end goal), of the to-be-performed learning, must be first well-defined formally.

2. The second and the most crucial step is the choice of the model. The prior, if available, can be used to choose an appropriate model or to tweak the model's bias accordingly. If the bias of the model is misaligned with the task, then no matter how efficient the subsequent stages are, the performance might be poor.

3. The next important step is training the model to accomplish the task. Here, the (untrained) model is somehow updated with the (statistical) information buried in the training data, leading to a trained model. The trained model must in some sense be optimal for the defined task.

4. The trained model can then be employed to perform various inferences in real-world deployment. In this course, we shall be restricting ourselves to inferences that merely can evaluate the empirical performance of the trained model wrt. the task at hand.

Students may also refer section 1.4 in Mohri et al. [2012] for related details.

Chapter 2

Abstract ML Settings

Since even merely imitating human cognition in a holistic manner is itself pretty overwhelming, ML is attempted in a few basic restricted settings. We begin with a popular ML setup.

2.1 Setup-1: Batch Supervised Function Induction

We describe the setup by providing details of each main stage in the ML pipeline. We begin with details of the learning task in the standard setup. This section is an expansion of section 2.1 in Shalev-Shwartz and Ben-David [2014].

2.1.1 Task: Function Induction

Function induction is a generic class of learning tasks. Here, the goal is to find a function $f^* : \mathcal{X} \mapsto \mathcal{Y}$ that induces the "best" label, $y_0 \in \mathcal{Y}$, for any[1] input, $x_0 \in \mathcal{X}$, i.e., $f^*(x_0) = y_0$. For example, consider the ML application where the input space is the set of all visual perceptions of various objects and the label space is the set of all nouns in a language. Such a learning task would be appropriate for imitating humans recognizing objects with their names.

In order to unambiguously define the notion of "best", we will need to first assume the existence of some ideal conceptualization of the input-label relationship.

[1] In contrast, if the labels are to be found only for a finite subset of inputs, then the task is known as transduction. The term induction highlights that the function must be defined for any input in the input space rather than over few inputs.

A simple, yet mostly naive, conceptualization is that of a functional relationship i.e., assuming there exists a single true label for any given input. Indeed functional relationship is very restrictive: recall the object recognition example from the preceding para. Assuming a functional relationship essentially means that objects are exactly recognizable given any of the object's visual perception, which is absurd. All of us remember the embarrassing situations where we failed to recognize our dear relative/friend because of partial darkness or she/he is wearing a mask etc. Hence, typically one assumes the existence of a true, but unknown, conditional likelihood[2]: $p^*(y/x)$. This conditional likelihood is known as the posterior likelihood of labels as it encodes the uncertainty in the label after observing the input.

Based on the label space such tasks are further categorized as[3]:

Classification: If the label space is a finite set and no structure that may relate the labels is available. Further, if there are only two labels, it is known as a Binary Classification task. E.g., ML applications like email spam filtering, product sentiment analysis, Windows-Hello/Apple-Face-ID typically involve a Binary classification task. If there are more than two labels, it is often qualified as a multi-class classification task. E.g., ML applications like online credit-card/loan approval, Google News involve a multi-class classification problem.

Regression: If the label space is a Euclidean space. Depending on the dimensionalty of the space, it is either scalar-regression or vector-valued/multi regression. E.g., quantitative prediction.

Structured Prediction: If the label space is discrete and is structured. E.g., speech recognition (sequence structure), image segmentation (grid structure), Hierarchical document classification (tree structure) etc.

Intuitively, $f^*(x)$ must be as "close" as possible to the posterior, $p^*(y/x)$, for any input x. In other words, the label $f^*(x)$ must be such that the expected error in approximating the random label following the posterior law, $p^*(\cdot/x)$, by $f^*(x)$ is the least possible:

$$f^*(x) \equiv \arg\min_{y \in \mathcal{Y}} \mathbb{E}_{Y/x \sim p^*(\cdot/x)} \left[l\left(y, Y\right) /x \right]. \tag{2.1}$$

[2]If label space is uncountable, then conditional likelihood is a density function (over labels for any given input) and is a mass function (over labels for any given input) if the label space is countable.

[3]Interested students may refer to section 1.3 in Mohri et al. [2012] for related details.

Here, l is a given loss function that evaluates the approximation loss incurred by replacing the labels in it's arguments with one another. Standard examples of loss functions are[4]:

0-1 loss: Defined by:

$$l(y, y') \equiv \begin{cases} 1 & \text{if } y \neq y' \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

squared loss: Defined by:

$$l(y, y') \equiv \left\| y - y' \right\|^2 . \tag{2.3}$$

Here, $\mathcal{Y} \subseteq \mathbb{R}^n$ and $\| \cdot \|$ denotes the corresponding Euclidean norm.

f^* is known as the Bayes optimal corresponding to the posterior, p^*, and loss l. Infact, (2.1) can be equivalently written as[5]: section 2.4.2 in et al. [2012].

$$f^* = \arg \min_{f: \mathcal{X} \mapsto \mathcal{Y}} R[f] \equiv \mathbb{E}_{(X,Y) \sim p^*} \left[l\left(f(X), Y \right) \right], \tag{2.4}$$

where $R[f]$ is known as the (true) Risk (of f), and p^* is the joint likelihood of input-label pairs[6]. Hence Risk minimization is the quintessential task in function induction. The minimum achievable risk, which is the risk of the Bayes optimal, is known as the Bayes (optimal) risk/error.

Given the learning task defined in (2.1,2.4), one can think about two very different modelling paradigms. The ones those explicitly model p^* – the Probabilistic Models (see section 3.2); and those that directly model f^* (see section 3.1). In the latter paradigm, if $\mathcal{F}$ is the model, then the Bayes optimal's restriction to $\mathcal{F}$ is defined by:

$$f^*_{\restriction \mathcal{F}} \equiv \arg \min_{f \in \mathcal{F}} \mathbb{E}_{(X,Y) \sim p^*} \left[l\left(f(X), Y \right) \right]. \tag{2.5}$$

We define the model error or approximation error of $\mathcal{F}$ as:

$$R[f^*_{\restriction \mathcal{F}}] - R[f^*] \tag{2.6}$$

[4] Refer pg 27, in Shalev-Shwartz and Ben-David [2014] for related details.

[5] In the

[6] The posterior $p^*(y/x)$ in (2.1) is same as the posterior induced by $p^*(x, y)$ in (2.4). In other words, $p^*(x, y) = p^*(y/x)p^*(x)$. Also, equivalence holds as long as $p^*(x) > 0 \ \forall \ x \in \mathcal{X}$.

2.1.2 Supervised Learning

In order that the trainnng data is useful for the defined task, (2.1,2.4), one must assume there is some relationship between the training data and the underlying truth, p^*. The most common assumption is that the training data is a collection of iid samples from $p^*(x, y)$. This is known as the Supervised learning setting.

In case of probabilistic models, supervised learning essentially is the problem of Likelihood Estimation i.e., estimating the likelihood p^* (or some likelihood related to either the posterior or the joint), from its samples. This problem has orginally been studied in statistics and is detailed in section ??. ML has advanced this area primarily via seminal results in Probabilistic Graphical Models [Koller and Friedman, 2009] and Deep Generative Models [Murphy, 2023] (Unit IV).

In case of function-based models, supervised learning essentially is the problem of estimating $f^*_{\restriction\mathcal{F}}$ defined in (2.4) (or finding a related function) using samples from the joint $p^*(x, y)$. Such optimization problems are known as the Stochastic Optimization problems. Algorithms for solving such problems fall under two paradigms: Sample Average Approximation (SAA) and Stochastic Gradient Descent (SGD). SAA is popularly known as Empirical Risk Minimization in Supervised Learning terminology and is detailed in section 4.1. SGD is detailed in section 4.2.

After training with m samples, one obtains an estimate for the (restricted) Bayes optimal, say $\hat{f}_m$. This is sometimes called as the trained model or as the prediction function. We define estimation error as:

$$R[\hat{f}_m(\mathcal{F})] - R[f^*_{\restriction\mathcal{F}}] \tag{2.7}$$

Theorems 4.1.1, under mild conditions, guarantee that the estimation error decays to zero asymptotically as number of samples grows to ∞. On the other hand, the model error can be decreased only by improved modelling. Typically, better modelling is achieved by closely interacting with domain experts in the ML application, collecting all possible domain knowledge, and carefully encoding this domain knowledge into an appropriate model. The sum of approximation and estimation errors is defined as the generalization error. Needless to say, the key objective in ML is to somehow ensure the generalization error is low. Please refer section 5.2 in Shalev-Shwartz and Ben-David [2014] for related details.

2.1.3 Evaluation: Batch

We begin by noting that, empirically evaluating a trained model, $\hat{f}_m(\mathcal{F})$, is same as estimating $R[\hat{f}_m(\mathcal{F})]$. This is typically performed by computing the average loss

of $\hat{f}_m(\mathcal{F})$ over an evaluation/test dataset. This is analogous to mimicing an exam where humans are expected to answer questions related to the concept taught. Needless to say, the evaluation is effective when the questions in exam are related yet different from those answered in lectures. Likewise, the test data is assumed to be:

1. iid samples from p^* (questions in exams must be related to the concept taught).
2. (statistically) independent of those in the training set (exam questions must not be repeated from lectures).
3. Most importantly, the test set MUST never to available to the learner (at learning stage). Otherwise we would say the exam paper leaked ;)

The above training and evaluation process is categorised as so-called batch learning. This is because, the entire training dataset (batch) is available for learning. And evaluation begins ONLY after the entire training data (batch) is processed (learned).

At times, a proxy/surrogate for the loss may be used at evaluation stage that is different (yet related) to that in the training. For example, in classification problems, it is common to employ the hinge/logistic loss during training and employ the 0-1 loss during evaluation[7]. Or some other times, estimate of $R[\hat{f}_m(\mathcal{F})]$ is presented in a more meaningful way: for example, in regression problems, people report the explained variance[8]. Explained variance basically tells how better/worse is the test set estimate of $R[\hat{f}_m(\mathcal{F})]$ (with squared loss), when compared to that of a constant prediction function that evaluates to the empirical mean of training labels (prior mean of labels) for any input.

2.2 Setup-2: Unsupervised Likelihood Estimation

Here, it is assumed that there is an unknown (yet fixed) likelihood p^* that is of interest. And the learning task is to find it.

Based on domain/prior knowledge (or convenience), we may wish to model p^* by an appropriate probabilistic model, $\mathcal{Q}$ (see section 3.2). In such a case,

[7] Popular evaluation metrics for classification are implemented here: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics.

[8] Popular evaluation metrics for regression are implemented here: https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics.

formally the learning task can be written as:

$$\arg\min_{q \in \mathcal{Q}} l(p^*, q), \tag{2.8}$$

where l is a loss function that provides the loss of replacing p^* by p. The most popular loss function[9] is the so-called KL divergence/Relative-entropy,

$$l(p, q) \equiv KL(p \parallel q) \equiv \mathbb{E}_{X \sim p}\left[\log\left(\frac{p(X)}{q(X)}\right)\right] \tag{2.9}$$

Refer section 6.2 in Murphy [2022] for details about KL divergence. We note the most important properties of this loss below[10]:

1. $KL(p \parallel q) \geq 0$ for any pair of likelihoods p, q.
2. $KL(p \parallel q) = 0 \iff p = q$.
3. Let $\{q_n\}$ be any converging sequence of likelihoods. Then, $KL(p, q_n) = 0 \iff \{q_n\} \to p$.

Using this loss, the task (2.8) can be re-written as:

$$\arg\min_{q \in \mathcal{Q}} \mathbb{E}_{X \sim p^*}\left[\log\left(\frac{p^*(X)}{q(X)}\right)\right], \tag{2.10}$$

which is also same as maximizing the so-called cross-entropy:

$$\arg\max_{q \in \mathcal{Q}} \mathbb{E}_{X \sim p^*}\left[\log\left(q(X)\right)\right], \tag{2.11}$$

To enable learning, let's assume that some training data is given. The crucial assumption we make is that the training data is a set of iid samples from p^*. This is known as the Unsupervised Learning assumption. This is indeed not supervised learning because in that case samples of the form $(z, p^*(z))$ need to be given for learning the function p^* (and this is impractical).

We define unsupervised likelihood estimation, as solving (2.8) given samples from p^*. In the special case of KL loss, this is same as solving the stochastic optimization problem (2.11), using samples, $\mathcal{D}$, from p^*. As mentioned earlier, this can be solved either using SAA (see section 4.3) or SGD (see section 4.2).

[9] Interested students may refer section 2.7 in Murphy [2023] for various other notions of loss functions. Recall that some of them were proposed by our students in the lecture itself!

[10] Additionally, it can be interpreted as a generalization of squared Euclidean distance. Interested students may see section 5.1.8 in Murphy [2023].

2.3 Setup-3: Unsupervised Conditional Likelihood Estimation

Here, it is assumed that there is an unknown (yet fixed) likelihood $p^*(x, y)$ that is of relevance. And the learning task essentially is to find $p^*(y/x)$, without finding $p^*(x, y)$.

Based on domain/prior knowledge (or convenience), we may wish to model $p^*(y/x)$ by an appropriate probabilistic (discriminative) model, $\mathcal{Q}$ (see section 3.2.2). In such a case, formally the learning task can be written as:

$$\arg\min_{q \in \mathcal{Q}} \mathbb{E}_{X \sim p^*(x)} \left[l(p^*(y/X), q(y/X)] \right., \tag{2.12}$$

where l is a loss function that provides the loss of replacing p^* by p. Using the popular loss function, KL divergence/Relative-entropy, as the loss:

$$\arg\min_{q \in \mathcal{Q}} \mathbb{E}_{X \sim p^*(x)} \left[\mathbb{E}_{Y/X \sim p^*(y/X)} \left[\log \left(\frac{p^*(Y/X)}{q(Y/X)} \right) \right] \right]. \tag{2.13}$$

From the total expectation rule[11], this is same as maximizing the so-called conditional likelihood of the training data:

$$\arg\max_{q \in \mathcal{Q}} \mathbb{E}_{(X,Y) \sim p^*(x,y)} \left[\log \left(q(Y/X) \right) \right]. \tag{2.14}$$

To enable learning, let's assume that some training data is given. We make the unsupervised learning assumption that the training data is a set of iid samples from $p^*(x, y)$.

We define unsupervised conditional likelihood estimation, as solving (??) given samples from $p^*(x, y)$. With a generic loss function, this objective is quite challenging as (2.12) involves conditionals, whereas samples are only available from the joint.

However, interestingly, in the special case of KL loss, this is same as solving the stochastic optimization problem (2.14), using samples, $\mathcal{D}$, from $p^*(x, y)$. As mentioned earlier, this can be solved either using SAA (see section 4.4) or SGD (see section 4.2).

[11]

2.4 Setup-4: Unsupervised Representation Learning

Here, it is assumed that there is an underlying likelihood, p^*, that generates the training data, $\mathcal{D} = \{x_1, \ldots, x_m\}$. And, the goal is to find a "good" representation of $x \in \mathcal{X}$. Since we wish to employ the paradigm of unsupervised learning, accordingly the training data does not include any examples/samples of representations of x. Recall that in deep learning classification/regression we implicitly learn a representation; however in those cases, though there is no supervision in terms of representations of x, there is supervision in terms of a linear function over the representations and hence one may say it is representation learning with "*weak*" supervision. What we envision in this section is a completely unsupervised learning.

Internet definition for weakly-supervised learning is different, it is similar to semi-supervised learning.

In order to formally define our goal, we employ a reconstruction loss, l, and define the representation learning task as:

$$\arg \min_{f \in \mathcal{F}, g \in \mathcal{G}} \mathbb{E}\left[l(X, g(f(X)))\right], \tag{2.15}$$

where f is an encoding/embedding function, $f(x)$ is the representation/embedding of x, g is a decoding/reconstructing function. $\mathcal{F}, \mathcal{G}$ are models for encoding and decoding functions, l is the reconstruction loss function, which quantifies how different it's arguments are. Popularly, we employ the square loss as the reconstruction loss. Solving (2.15) given samples from X defines the Unsupervised Representation Learning problem. Needless to say, the encoder, decoder models must be compatible in the sense that that co-domain (range) of the encoder functions must be the domain of the decoder functions.

In the special case, $f : \mathbb{R}^n \mapsto \mathbb{R}^d$, $g : \mathbb{R}^d \mapsto \mathbb{R}^n$, where $d \ll n$, the task of representation learning is known as dimensionality reduction. Motivations for such a reduction are listed in the beginning of chapter 23 in Shalev-Shwartz and Ben-David [2014].

2.5 Setup-5: Unsupervised Support Estimation

Formally, this is the task of learning/estimating the support of a likelihood given it's samples. Support of p is $\{x \mid p(x) > 0\}$. In practice, we look for points where $p(x) > \epsilon > 0$. Outlier/intrusion/novelty detection are example applications, where ϵ is set to a low value.

A straight-forward strategy is to first solve the likelihood estimation problem

and then appropriately threshold to estimate the support. However, one can estimate the support using function-induction models too (refer section 5.18).

2.5.1 Setup-6: Clustering

In a popular variant of this task, one needs to find so-called clusters in the data. Clusters are defined as contiguous regions in the domain, where the datapoints within have high likelihood (density). Again, one straight-forward strategy is to first solve the support estimation problem (with appropriately high threshold, ϵ) and then find contiguous subsets of it. There are many popular alternate (approximate) definitions for our definition. Interested students may refer section 22.2 in Shalev-Shwartz and Ben-David [2014].

2.6 Setup-7: Online Learning

In this section we study an interesting alternative to batch supervised function induction set-up. Unlike in a batch setting, humans are typically evaluated, or tend to utilize their predictions, even while learning. Accordingly, we defined the online supervised learning framework, where characteristics of such a form of learning are encoded nicely.

Here examples $(x_1, y_1), (x_2, y_2) \ldots, (x_m, y_m), \ldots$ arrive sequentially and can be "seen" only once (like in a stream). When the first example (x_1, y_1) arrives, only x_1 is shown to the learner. Then the learner makes a prediction about y_1 as say, $\hat{y}_1$. Then, y_1 is shown to the learner. If $y_1 \neq \hat{y}_1$, then it is counted as a mistake. In general, one instead compute a loss $l(y_1, \hat{y}_1)$. Based on this loss/feedback the learner "learns" something related to input-label relations. Then, the second example (x_2, y_2) arrives, ..., and so on for m rounds. The goal in online learning is to minimize the number of mistakes, or more generally, minimize the total loss in the m rounds.

Since the goal in online learning only concerned with happenings in m rounds, rather than "generalization" to unseen inputs, one usually does not assume anything about the generation process of inputs, i.e., $p^*(x)$. Only assumptions about the relationship between inputs-labels are assumed, i.e., assumptions like existence of $p^*(y/x)$ or a function f^* such that $f^*(x) = y$ etc.

As in case of batch learning, no learning seems possible even in online learning without any restriction on candidates for, say, f^* i.e., $\mathcal{F}$ is the set of candidates

(model). For example, we repeat the discussion in section 21.1[12] in Shalev-Shwartz and Ben-David [2014] and mainly noted the Halving algorithm for online binary classification and it's mistake bound (upper bound on no. mistakes in m rounds) as $\log(|\mathcal{F}|)$. So interesting online learning happens iff $|\mathcal{F}| < o(2^m)$. This seems fine because, an unrestricted class $\mathcal{F}$ would allow all possible labelling functions over m inputs, which is 2^m. Interested students may read section 21.1.1 to know more about characterizing cases where online learning is possible in the set-up of section 21.1.

2.7 Setup-8: Reinforcement Learning

Using the example of "how infants learn to roll over?", we motivated the need for a new set-up/model in machine learning. This is because humans do not seem to acquire such fundamental skills through examples (either supervised or unsupervised). Accordingly, instead of example-based learning, we introduced the notion of *trial and error* based learning, which will later be formalized as Reinforcement Learning (RL). Informally, we summarized the key characteristics of a reinforcement learning set-up:

1. It is not an example-based learning. Hence, fundamentally very different from both Supervised and Unsupervised Learning.

2. Learning happens by trying various possible actions, which result in rewards (loss or error) i.e., **trail and error**.

3. The motivation for learning is to maximize reward (or minimize loss). In this sense it is similar to supervised learning. Also, rewards are assumed to be stochastic like in supervised learning.

4. Again, like in unsupervised learning, typically RL algorithms directly/indirectly estimation some likelihoods. This is the similarity with unsupervised learning. However, the underlying likelihoods may also change in a full-fledged RL problem. But most simple case is the stationary assumption, where the underlying likelihoods are assumed to be the same at every iteration/step.

5. Like online learning, in RL too the learning agent needs to make sequential decisions and the notion of regret is most natural.

[12] You may skip section 21.1.1.

6. However, strikingly different from online supervised learning, the decisions cannot simply try to maximize reward, because perhaps other better rewarding actions are not explored enough. For example, if infant stops trying something different after rolling over, then perhaps it would never learn to crawl! This brings to the most important aspect of RL, which is the trade-off between **exploration-exploitation**.

7. In general RL set-ups, the notion of delayed rewards exists. for e.g., a wrong move in chess will result in negative reward only after many more moves. Such a concept of delayed loss/reward is not present in online supervised learning. The simplest case is that of immediate rewards, which makes it similar to online supervised learning.

8. Another aspect that makes RL fundamentally different from Supervised and Unsupervised Learning is the concept of states. Actions not only result in rewards but also lead to changes in the state of the learning agent (like position of infant).

We then began formally studying restricted versions of the RL set-up, which are easier to analyze. We began with single state, stationary, immediate reward RL, which is formalized as the multi-arm bandit problem. Please refer sections 2-2.1 in Sutton and Barto [2018] for details.

2.7.1 Multi-arm Bandit Set-up

We began formally studying the Bandit problem. Let the set of possible actions be $\{a_1, \ldots, a_k\}$. Let A_t denote the action taken at instant t and R_t be the corresponding reward[13].

We define the value of an action/arm, a_i, as $\mathbb{E}[R_t / A_t = a_i]$ and denote it by $q(a_i)$. Note that by the stationary assumption, the value is independent of the instant, t. We denote the empirical estimate[14] of $q(a_i)$ at instant t by $Q_t(a_i)$. We denote the best arm as the one(s) with the maximum value: $a^* \in \arg\max_{a \in a_1, \ldots, a_k} q(a)$. See also section 2.2 inSutton and Barto [2018].

An algorithm, $\mathcal{A}$, is a data-dependent scheme/policy for choosing the arms to be pulled at every instant based on the Q function. We then defined the regret with an algorithm $\mathcal{A}$ as:

$$regret_{\mathcal{A}}(n) \equiv nq(a^*) - \sum_{i=1}^{k} T_i(n) q(a_i), \tag{2.16}$$

[13] Both A_t, R_t are random variables.

[14] One way to estimate is given in equation (2.1) in Sutton and Barto [2018].

where $T_i(n)$ is the random variable denoting the number of times arm a_i was pulled in first n instants. Obviously, $T_1(n) + \ldots T_k(n) = n$. Typically one is interested in upper bounding the expected regret[15].

Needless to say, if $T_{i^*}(n) = n$, where $a_{i^*} = a^*$, then regret is minimized. However, the identity of this arm is not known and hence learning is relevant.

[15]If expected regret grows slower than n, then such algorithms are guaranteed to figure out the optimal arm asymptotically.

Chapter 3

ML Models

3.1 Models for Function Induction

3.1.1 Linear Models

These are the simplest models in ML. Informally, they are the collection of all linear functions defined over an input space. However, if the input-space has no predefined mathematical structure, then it is not clear what linear functions are. Hence, we assume that a feature map, $\phi : \mathcal{X} \mapsto \mathbb{R}^n$, is available. Basically, the feature map, ϕ, provides a Euclidean representation/embedding for the inputs. The co-domain/range of ϕ, which is $\mathbb{R}^n$, is known as the (input) feature space and the entries of $\phi(x)$ i.e., $\phi_j(x), j = 1, \ldots, n$ are known as the (input) features.

We define the linear model as:

$$\mathcal{L}_{n,\phi} \equiv \left\{ f \mid f(x) = w^\top \phi(x) \; \forall \; x \in \mathcal{X}, \text{ for some } w \in \mathbb{R}^n \right\}, \tag{3.1}$$

where n is the dimensionality of the range of the feature map. Here, w is the parameter for the model. In other words, linear model is nothing but the set of all linear functions in the input feature space.

Infact, (3.1) defines a family of models, one for each ϕ. The feature map is usually *designed* using domain knowledge[1]. The final performance of the trained model will in general be strongly dependent of the feature map and hence needs to be designed very carefully. Typical estimation error bounds with this model turn out to be: $O\left(\frac{n \log(m)}{m}\right)$. Though the good news is that as $m \to \infty$, the estimation

[1] For example, see `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction` for example codes that compute feature maps relevant for vision and NLP applications.

error decays to 0, the flip-side is that enormous number of training samples may be needed in high-dimensional feature spaces!

3.1.2 Regularized Linear Models

In linear models, restricting the form of the function via feature map design is the way in which inductive bias is introduced. This is fine for applications like Voltage-Amperage (Ohm's law experiment), where it is known that the expected/ideal input-label relationship has a particular functional form. However, many times the kind of domain knowledge or prior information available may be more implicit. For e.g, given the input features, it might be intuitive in some applications that actually only few of the features are sufficient for predicting the labels (Mechanism is simple!). One way of introducing appropriate bias is to include a constraint of the form $\Omega(w) \leq W$, where Ω is some function that correlates with sparsity in w, the model parameters. Lower the W, sparser functions are the candidates in the model, hence sparser is the ERM solution, and hence fewer features are actually used for prediction (as parameters corresponding to more features are zero). An example of such an Ω is the 1-norm i.e., $\Omega(w) = \|w\|_1$[2]. In some applications, like recommender systems, one may wish to look for parameter matrices having a low rank. The so-called nuclear norm (or trace norm) of the parameter matrix, $\|w\|_*$, is an example function that correlates with rank[3] i.e., $\Omega(w) = \|w\|_*$.

Such functions that control characteristics of the model parameter, which are known from domain knowledge apriori, are called as regularizers. The parameter, W, which controls the value of this regularizer is called as the regularization hyperparameter. Below is the formal definition of the family of regularized linear models:

$$\mathcal{L}_{n,\phi,\Omega,W} \equiv \left\{ f \mid f(x) = w^\top \phi(x) \; \forall \; x \in \mathcal{X}, \text{ for some } w \in \mathbb{R}^n, \Omega(w) \leq W \right\}, \tag{3.2}$$

Note that, apart from the feature map, given a regularizer and a regularization hyperparameter, the model (inductive bias) is fixed. If one changes the regularizer, Ω, and/or, changes the regularization hyperparameter, then the model (inductive bias) changes. If one choose a generously large feature map, then the inductive

[2]This is because the optimal conditions say that the gradient of the objective must belong to the cone of (outward) normals for the feasibility set at an optimal solution. And since for the feasibility set, $\|w\|_1 \leq W$, the set of normals at the points on axis (where w is sparse) is far bigger than at other points, the chance that an arbitrary objective function's gradient matches the normal directions of the feasibility set is far higher at the points where w is sparse. This was explained in the lecture with intuitive illustrations.

[3]Interested students may refer `https://www.di.ens.fr/~fbach/mlss08_fbach.pdf` for more details on sparsity inducing regularizers like 1-norm/trace-norm.

bias is controlled mainly by the regularization. In this course, we will closely study only one regularizer, which is the 2-norm based one:

3.1.2.1 $l2$ regularized linear models

Here we focus on the special family of models where $\Omega(w) \equiv \|w\|_2$, which is the default Euclidean norm (so-called $l2$-norm or 2-norm):

$$\mathcal{L}_{n,\phi,W} \equiv \left\{ f \mid f(x) = w^\top \phi(x) \; \forall \; x \in \mathcal{X}, \text{ for some } w \in \mathbb{R}^n, \|w\|_2 \leq W \right\}. \tag{3.3}$$

Typical estimation error bounds with this model turn out to be $O\left(\frac{WR}{\sqrt{m}}\right)$, where $\max_{x \in \mathcal{X}} \|\phi(x)\| \leq R$. The important observation is that this bound is completely independent of the dimensionality of the feature space.

3.1.3 Kernel based Models

Recall that if one wants to deal with a feature map in linear models that induces all possible r degree monomials over the input factors in $\mathbb{R}^d$, then $n = O(d^r)$. In such cases, or essentially cases where n is high, solving the ERM problem may be computationally challenging. We then made the observation that the optimal solution of the ERM problem with $l2$ regularized linear models:

$$\min_{w \in \mathbb{R}^n} \quad \frac{1}{2}\|w\|_2^2 + C \sum_{i=1}^{m} l\left(w^\top \phi(x_i), y_i\right) \tag{3.4}$$

will always be a linear combination of the feature maps of the training inputs. This is called as the representer theorem:

Theorem 3.1.1. *Any optimal solution, $\hat{w}$, of (3.4) will be of the form $\hat{w} = \sum_{i=1}^{m} \alpha_i \phi(x_i)$. Hence (3.4) is equivalent to:*

$$\min_{\alpha \in \mathbb{R}^m} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k\left(x_i, x_j\right) + C \sum_{i=1}^{m} l\left(y_i, \sum_{j=1}^{m} \alpha_j k\left(x_i, x_j\right)\right), \tag{3.5}$$

where $k\left(x_i, x_j\right) \equiv \phi(x_i)^\top \phi(x_j)$. Also, $\hat{w}^\top \phi(x) = \sum_{i=1}^{m} \alpha_i k\left(x_i, x\right) \; \forall \; x \in \mathcal{X}$.

Refer section 4.2 in Scholkopf and Smola [2001] or theorem 5.4 in Mohri et al. [2012] for details of the proof.

From the theorem it is clear that as long as the computation of the dot product, i.e., evaluation of the function k, is computationally efficient, things are fine.

Interestingly, using Multinomial theorem, we noted that $k(x,z) = \left(1 + x^\top z\right)^r = \phi(x)^\top \phi(z)$, where $\phi(x)$ has all possible monomials in x upto degree r. Hence computing $k(x,z)$, in this case, is only $O(d+r)$ rather than $O(d^r)$.

There are many other examples where the function k can be evaluated efficiently as opposed to naively evaluating the corresponding dot-product (see `https://www.jmlr.org/papers/volume2/lodhi02a/lodhi02a.pdf` for example with strings). Encouraged by such results we asked if there are any common properties that characterize such functions k, which evaluate the dot product in some feature space. The answer is the so-called kernels.

3.1.3.1 Kernels

Given a domain $\mathcal{X}$, a function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a kernel iff (a) k is symmetric, i.e., $k(x,z) = k(z,x)\ \forall\ x,z \in \mathcal{X}$ (b) for any $m \in \mathbb{N}$ and for any $x_1, \ldots, x_m \in \mathcal{X}$, the so-called gram-matrix, G, whose i,j^{th} entry is $k(x_i, x_j)$, must be positive-semidefinite.

Theorem 3.1.2. *If k is a (valid) kernel over the domain $\mathcal{X}$, then there exists a Hilbert space, $\mathcal{H}$, and a feature map, $\phi : \mathcal{X} \mapsto \mathcal{H}$, such that $k(x,z) = \langle \phi(x), \phi(z) \rangle\ \forall\ x,z \in \mathcal{X}$. $\langle \cdot, \cdot \rangle$ is the (default) inner-product in the Hilbert space $\mathcal{H}$. Conversely, if $k(x,z) = \langle \phi(x), \phi(z) \rangle\ \forall\ x,z \in \mathcal{X}$ for some Hilbert space, $\mathcal{H}$, and feature map, ϕ, then, the function k must be a (valid) kernel. Also, if k_1 and k_2 are kernels, then:*

1. *The function $k \equiv k_1 + k_2$ will be a (valid) kernel.*
2. *The function $k \equiv k_1 k_2$ will be a (valid) kernel.*

Moreover, if $k_1, \ldots, k_n, \ldots$ is an infinite sequence of (valid) kernels, and if the (point-wise) limit $\lim_{n\infty} k_n$ exists, say k, then, k will be a (valid) kernel.

Using this theorem we proved the following are valid kernels over $\mathbb{R}^n$:

Linear Kernel: $k(x,z) \equiv x^\top z$.

Polynomial Kernel: $k(x,z) \equiv (1 + x^\top z)^r$. r is the hyperparameter.

Gaussian/RBF Kernel: $k(x,z) \equiv e^{-\frac{1}{2\sigma^2}\|x-z\|_2^2}$. σ is the hyperparameter.

Now, we are ready to define kernelized models: given an input space, $\mathcal{X}$, we first design a kernel over $\mathcal{X}$. Then the model (equivalent to (3.2)) can be described

as:

(3.6)

$$\mathcal{L}_{k,W} \equiv \left\{ f \mid \exists\ m \in \mathbb{N}, x_1, \ldots, x_m \in \mathcal{X}, \alpha \in \mathbb{R}^m,\ \ni\ \ f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x), \alpha^\top G \alpha \leq W^2 \right\},$$

where G is the gram-matrix with k over $x_1, \ldots, x_m$. And the corresponding ERM/prediction function are as given in thoerem 3.1.1. Also, note that (3.6) is a non-parametric description. Moreover, when the kernel is Gaussian, the prediction function is linear combination of Gaussians and hence is a universal approximator[4]. Hence, the modelling error is expected to be low (for large enough W) and the earlier estimation error bound further simplifies[5] to $O(\frac{W}{\sqrt{m}})$. Further, the ERM problem (3.5) is always a convex problem as long as the loss function, l, is convex. So the numerical optimization error is also expected to be low (with reasonable computational effort). These three important characteristics made these Gaussian kernel-like models default choice for machine learning (until 2010, when deep learning era started).

In summary, kernelized models can be understood as non-parametric models (3.6) or as linear models (3.1) with a kernel-induced feature map (i.e., a feature map in theorem 3.1.1). In general, any model/algorithm can be "kernelized" by replacing geometric operations by kernel evaluations. For example, . This is usually called as the kernel trick. However, if some operations in the algorithm cannot be described by (Euclidean) geometric operations, then those can't be kernelized. For more details, please refer chapter 16 in Shalev-Shwartz and Ben-David [2014] and chapter 6 upto section 6.3 in Mohri et al. [2012].

3.1.4 Artificial Neural Networks (Deep Learning Models)

In linear modelling (whether regularized or not; whether kernelized or not) the goal was to (directly) model the Bayes optimal function. However, since mimicing cognitive abilities of humans was our primary goal, a powerful modelling strategy would be to model "functions" that encoded neural processing in humans, instead[6]. Observe that, the kind of inductive biases encoded by a linear model would be very different from those encoded by such human neural models.

Though this looks like an attractive (and obvious) strategy, the main difficulty is that we understand very little about human brain and neural processing.

[4] Interested students may refer `https://jmlr.csail.mit.edu/papers/volume7/micchelli06a/micchelli06a.pdf` for more details.

[5] Because $R = \max_{x \in \mathcal{X}} \|\phi(x)\| = \max_{x \in \mathcal{X}} \sqrt{k(x,x)} = 1$ (for Gaussian kernel).

[6] Ofcourse, since humans excel in these abilities, the Bayes optimal may also be well approximated by these functions.

Nevertheless, for the purpose of achieving specific tasks, perhaps a rough/coarse model of how neural processing happens in humans may also suffice. One such naive, yet useful (non-trivial), model of neural processing is stimuli following through a network of neurons.

More specifically, it is believed that few neurons in a network of neurons perceive input stimuli and each neuron processes it's own input: performs a simple 'gating' style computation i.e., some weak/uninteresting aggregated signals are ignored; while responding to strong/interesting aggregated signals. Thus an input stimulus get processed as it passes through the neuron network. Then final decision/action is taken based on the processed input. Also, the connectivity, strength of connections between neurons changes with time, which essentially amounts to learning/adaptation.

The above simple model of neural processing is aptly copied in mathematical models called Artificial Neural Network (ANN) models. These models are defined by a directed acyclic graph, where nodes represent 'artificial' neurons and edges denote the connectivity among them. Edges have weights to represent the strength of connectivity. In this course, we will only study the most basic kind of artificial neural networks, where the nodes are partitioned into layers depending on their hop-distance to input stimuli, x, i.e., the in the first layer are those that directly receive input from, x, and those in second layer are those that receive stimuli from the first layer neurons only and so-on. Moreover, there are directed edges from every node in layer i to every node in layer $i+1$. Such special ANNs are called as Fully-connected Feed-forward Neural Networks (FFNN) or MultiLayer Perceptron (MLP). The number of layers is called the depth of the FFNN and the (common) number of nodes in each layer is called the width of the FFNN. Since biological neural networks are believed to be deep, we expected deep ANNs to generalize well in practice. Hence these models are popularly dubbed as deep learning models.

Let V_i denote the matrix where the j^{th} column is the vector of incoming edge weights from nodes in layer $i-1$ to the j^{th} node in the i^{th} layer. At each artificial neuron, the stimuli from it's inputs are aggregated using a weighted sum operation. For example, $V_1^\top x$ is the vector of weighted sum stimuli at the first layer nodes. The main processing that happens in each node (artificial neuron) is a simple gating-style activation over this aggregated stimulus. For example, $\sigma(V_1^\top x)$ is the activated response of the nodes in the first layer. The function σ, which acts element-wise on it's input is called the activation function. The most popular example of activation function is the so-called REctified Linear Unit (ReLU): $\sigma(x) \equiv \max(0, x)$. Formally, if input, $x \in \mathbb{R}^n$, depth is d, and width is q, then the FFNN model can be mathematically expressed as:

$$\mathcal{N}_{n,d,q,\sigma} \equiv \left\{ f \mid \exists V_1 \in \mathbb{R}^{n\times q}, V_2 \in \mathbb{R}^{q\times q} \ldots, V_d \in \mathbb{R}^{q\times q}, w \in \mathbb{R}^q \ni f(x) = w^\top \left(\sigma\left(V_d^\top \left(\sigma\left(V_{d-1}^\top \ldots \sigma\left(V_1^\top x\right)\right)\right)\right)\right)\right\} \tag{3.7}$$

The parameters of the model are: $V \equiv [V_1, \ldots, V_d], w$. The hyperparameters are d, q and σ needs to be designed carefully. In general, for ANNs, designing the DAG, aggregation, activation, and clever parameter tying are the crucial modelling steps. Like linear models, (universal) kernel based models, FFNNs are also universal approximators[7]. Please refer https://d2l.ai/chapter_multilayer-perceptrons/mlp.html for more details.

3.1.5 Non-parametric Models

Here, the basic idea is to not restrict the choice of candidate functions in (2.4) apriori using a (parametric) model. But rather somehow (by definition itself) ensure that the (trained) model is a close approximation of an ERM solution computed over all possible functions (unrestricted). As mentioned earlier, such an ERM solution will achieve zero loss on the training data (memorizing training data). However, the performance on unseen data might be arbitrarily bad.

The idea in non-parameteric function models is to somehow restrict the trained function values on unseen data such that it will generalize well. Typically, in such models, different candidate functions are described using different (number of) parameters, hence the qualifier non-parametric is employed for such models.

3.1.5.1 Nearest Neighbour models

Like how the linear model is defined given a feature map over the input space, a Nearest Neighbour (NN) model is defined given a (valid) distance[8] function over the input space: $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$.

1-NN Classification model

Each candidate function in this model is then described using a finite set, $\mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$. Let the i^{th} pair in $\mathcal{S}$ be $(x_i^{'}, y_i^{'})$. Given the set $\mathcal{S}$, the corresponding function, $f_{\mathcal{S}} : \mathcal{X} \mapsto \mathcal{Y}$, is defined by: $f_{\mathcal{S}}(x) = y_{i^*(x)}^{'}$, where $i^*(x) \in \arg\min_{i=1,\ldots,|\mathcal{S}|} d(x, x_i^{'})$. In other words, the function's value at any input seen in $\mathcal{S}$ is the corresponding label and at any unseen input is that of the nearest input, $x_{i^*(x)}^{'}$ (ties broken arbitrarily). See figure 16.1 in Murphy [2022] for an illustration of this model in

[7] Interested students may refer section 20.3 in Shalev-Shwartz and Ben-David [2014] for details.

[8] Refer https://en.wikipedia.org/wiki/Metric_space for properties of distance like non-negativity, symmetry and triangle inequality. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.DistanceMetric.html?highlight=di is a good collection of popular distance metrics.

2-d space. Formally, 1-NN model is:

(3.8)

$$^{1}\mathcal{N}_d \equiv \left\{ f \mid \exists\ (x_i^{'}, y_i^{'}) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, s,\ s \in \mathbb{N},\ \ni\ f(x_i^{'}) = y_i^{'}, \forall i = 1, \ldots, s, f(x) = y^{'}_{i^*(x)}, i^*(x) \in \arg\min_{i=1,\ldots,|\mathcal{S}|} d(x, x_i^{'}) \right\}$$

The set, $\mathcal{S}$, could be called as the parameter. But the parameters for different candidate functions may be different, hence qualifying this model as a non-parametric model is well-justified.

Theorem 19.3 in Shalev-Shwartz and Ben-David [2014] provides a formal justification of why/when this 1-NN classification generalizes well. Note that the justification is a consequence of the key assumption that the label posterior given an input is same (close) as the label posterior given it's nearest neighbour (asymptotically).

So under the assumptions of this theorem, asymptotically, the true posterior, $p^*(y/x)$, at an $x \in \mathcal{X}$ and the true posterior at it's nearest neighbour, $p^*(y/x_{i*(x)})$, will essentially be the same. Hence the label of the nearest neighbour is nothing but a sample from $p^*(y/x)$. In 1-NN, this label is simply copied as the label of x.

Now, if $k > 1$ nearest neighbours are considered then under appropriate assumptions their labels will again be samples from $p^*(y/x)$. Labeling x using all these k labels (samples) seems an improved strategy. This leads to the idea of k-NN classification:

k-NN Classification model

Each candidate function in this model is described using a finite set, $\mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$. Let the i^{th} pair in $\mathcal{S}$ be $(x_i^{'}, y_i^{'})$. Given the set $\mathcal{S}$, the corresponding function, $f_{\mathcal{S}} : \mathcal{X} \mapsto \mathcal{Y}$, is defined by: $f_{\mathcal{S}}(x)$ is the majority label among those of the k nearest neighbours of x. We call this model as the k-NN classification model and denote it by $^{k}\mathcal{N}_d$. Majority label is appropriate because it is nothing but the mode of the empirical likelihood formed with labels of the k neighbours (which are samples from $p^*(y/x)$ as hinted above). And posterior mode is the Bayes optimal with the 0-1 loss.

k-NN Regression model

Exactly same as the above except that the predicted label of x is the average of the labels of the k neighbours. Average label is appropriate because it is nothing but the expectation/mean of the empirical likelihood formed with labels of the k neighbours (which are samples from $p^*(y/x)$ as hinted above). And posterior mean is the Bayes optimal with the squared loss.

3.2 Probabilistic Models

Here, the goal is to model the underlying likelihood p^*. In Generative modelling (refer section 3.2.1), the aim is to model the (unknown) likelihood that generates the data i.e., the likelihood from which the training data is sampled (e.g., the joint likelihood, $p^*(x, y)$, in case of classification tasks). Whereas, in discriminative modelling (refer section 3.2.2), the aim is to model the posterior likelihood, $p^*(y/x)$, of labels (given the input). Either way, one can compute the Bayes optimal classifier corresponding to the estimated $\hat{p}$ and used it for prediction.

3.2.1 Generative Models

Because these model the likelihood generating the training data, they are useful for both Supervised and Unsupervised Learning.

3.2.1.1 Bernoulli Model

Bernoulli model, $\mathcal{B}$, is nothing but the set of all possible Bernoulli likelihoods i.e., likelihoods of the form: $p_\theta(x) = \theta^x(1-\theta)^{1-x} \ \forall \ x \in \{0, 1\}$. Here, $\theta \in [0, 1]$, the probability of 1 (heads), is the parameter for the model. Please see section 2.4.1 in Murphy [2022].

3.2.1.2 Multinoulli Model

Multinoulli (Categorical) model, $\mathcal{M}$, is nothing but the set of all possible Multinoulli likelihoods i.e., likelihoods of the form: $p_\theta(x) = \theta_x, \ \forall \ x \in 1, \ldots, c$. Here, $\theta \in \Delta_c$, is the parameter for the model[9]. Please see section 2.5.1 in Murphy [2022].

3.2.1.3 Gaussian Model

Gaussian model, $\mathcal{G}_n$ is nothing but the set of all possible Gaussian likelihoods over $\mathbb{R}^n$, i.e., likelihoods of the form: $p_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)} \ \forall \ x \in \mathbb{R}^n$. Here, the mean vector, $\mu \in \mathbb{R}^n$, and the covariance matrix, $\Sigma \succ 0 \in \mathbb{S}_n$, are the parameters. Please see section 3.2-3.2.4 in Murphy [2022].

[9] $\Delta_n = \left\{x \in \mathbb{R}^n \mid x \geq 0, \sum_{i=1}^n x_i = 1\right\}$, is the n-dimensional simplex (of probabilities).

3.2.1.4 Gaussian Mixture Model (GMM)

This model has likelihood functions that are convex combinations of a fixed number, k, of component likelihoods, which are nothing but n-dimensional Gaussian likelihood functions. If the parameters of these k Gaussians/components are (μ_k, Σ_k) and the coefficients are the convex combination are $\theta_1, \ldots, \theta_k$, then the likelihoods in this model are given by

$$p(x) = \sum_{i=1}^{k} \frac{\theta_i}{(2\pi)^{n/2}|\Sigma_i|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_i)^\top \Sigma_i^{-1}(x-\mu_i)}, \tag{3.9}$$

where $\mu_i \in \mathbb{R}^n, \Sigma_i \succ 0 \in \mathbb{R}^{n\times n}, \theta \in \Delta_k$. ($\Delta_k$ is the k-dimensional simplex).

This family of models is known to be universal approximators for likelihood functions[10] and are popularly used in clustering tasks.

3.2.2 Discriminative Models

In discriminative models, the goal is to model a conditional likelihood of labels given the observed inputs i.e., $p^*(y/x)$. The general idea here is to start with a generative model over labels and make the parameters a function of inputs. The parameter functions can themselves be modelled using any of the function-based models (see section 3.1) like linear models etc.

3.2.2.1 Discriminative Multinoulli Model

Motivated by (5.10), this model is a multinoulli over labels, $\{1, \ldots, c\}$, where the parameter functions are given by softmax:

$$p_W(y/x) = \frac{e^{w_y^\top \phi(x)}}{\sum_{y' \in \mathcal{Y}} e^{w_{y'}^\top \phi(x)}} \;\forall\; y \in \{1, \ldots, c\},\; \forall\; x \in \mathcal{X}, \tag{3.10}$$

where $W = [w_1 \ldots w_c]$ is the parameter.

3.2.2.2 Discriminative Gaussian Model

This is a Gaussian over labels, where the mean function is modelled by the linear model (and vairance is a known constant):

$$p_w(y/x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\left(y - w^\top \phi(x)\right)^2} \;\forall\; y \in \mathbb{R},\; x \in \mathcal{X}. \tag{3.11}$$

[10] Interested students may read theorem 6.6 in Scott [1992] for details.

3.2.3 Non-parametric Likelihood Models

Here, the basic idea is to not restrict the choice of candidate likelihoods in (2.8) apriori using a (parametric) model. But rather somehow (by definition itself) ensure that the (trained) likelihood is a close approximation of the empirical likelihood with the training data. Recall that empirical likelihood is the solution of (4.4) with $\mathcal{Q}$ unrestricted. As mentioned earlier, while the empirical likelihood is guaranteed to asymptotically converge to p^*, it is not a valid density and hence useless when the likelihood $p*$ is a density.

The idea in non-parametric likelihood (density) models is to somehow smoothen the empirical likelihood so that unseen datapoints also get some non-zero likelihood (density).

3.2.3.1 Kernel Density Estimation (KDE) model

We exactly follow sections 16.3-16.3.3 in Murphy [2022]. The KDE model is defined using a so-called density/smoothing kernel, which is nothing by a density (over $\mathbb{R}^n$) symmetric wrt. the origin. Popular smoothing kernels are in table 16.1 in Murphy [2022][11].

[11] See also `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html#sklearn.neighbors.KernelDensity`.

Chapter 4

Training Algorithms

4.1 Empirical Risk Minimization

Refer:sections 2.2, 2.3 in Sha Shwartz Ben-David [201

Recall that the goal is to solve the stochastic optimization problem (2.4) given the training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. Here, $\mathcal{D}$ is assumed to be a set of m iid samples from $p^*(x, y)$. Let (X_i, Y_i) denote the random variable corresponding to (x_i, y_i) i.e., (x_i, y_i) is the value of (instantiation) of (X_i, Y_i).

Since the key difficulty in solving (2.4) seems to be in computing the (true) risk, one obvious idea is to perhaps approximate the (true) risk with average loss computed over training set: $\hat{R}_m[f] \equiv \frac{1}{m}\sum_{i=1}^{m} l(f(X_i), Y_i)$, which we call as the Empirical Risk (with f). The motivation for this is the law of large numbers that guarantees that the average (loss with g) over the training set will be arbitrarily close to the expectation (of loss with g), with arbitrarily high probability, provided m is big enough! Please refer for details.

This motivates the idea of Empirical Risk Minimization:

$$\hat{f}_m^{\text{ERM}}(\mathcal{F}) \equiv \arg\min_{f \in \mathcal{F}} \hat{R}_m[f] \tag{4.1}$$

as a proxy for true risk minimization, (2.4). Now, how bad can the ERM solution be compared to the (restricted) Bayes optimal? This is answered by the fundamental theorem 26.5(3) (and results like lemma 26.10) in Shalev-Shwartz and Ben-David [2014], which is summarized below using our notation:

Theorem 4.1.1. *With probability* $1 - \delta$,

$$R\left[\hat{f}_m^{\text{ERM}}(\mathcal{F})\right] \leq R[f^*_{|\mathcal{F}}] + O\left(\sqrt{\frac{\mathcal{C}(l, \mathcal{F})}{m}}, \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{m}}\right),$$

where $C(l, \mathcal{F})$ is the "complexity/capacity"[1] of the loss-model combination.

The key take-home from the above theorem is: Estimation error will be arbitrarily small, with arbitrarily high probability, provided m is large enough, and complexity of model $C(l, \mathcal{F})$ is finite. So in machine learning we always design models with finite capacity[2] Theorem 6.6 in Shalev-Shwartz and Ben-David [2014] is one formal result that confirms that infinitely complex models ('bias-free'/'unrestricted' models) don't learn (see also section 2.2.1 in Shalev-Shwartz and Ben-David [2014]). Bounds as those in theorem 4.1.1 are referred to as learning/high-confidence/Probably-Approximately-Correct/PAC bounds.

4.1.1 ERM solver: Gradient Descent

ERM problems are typically solved using the gradient descent algorithm (or its variants). See also, section 8.2 in Murphy [2022], sections 6.1, 6.2 in Murphy [2023] for related details. Whenever the ERM problem is convex (in the parameters), it's convergence is guaranteed.[3] This is formalized in theorem 3.2.2 in Nesterov [2014] (step size as in eqn. (3.2.10) in same book). From this theorem, the approximation error of a GD solution is atmost $O(1/\sqrt{T})$, where T is number of iterations. It is amazing that in this case, irrespective of initialization, dimesionality of feature space, etc., we can compute as accurate an ERM solution as we want by simply running GD for enough number of iterations.

However, in the non-convex case, one is happy if a stationary point can be found, i.e., a parameter where gradient of the ERM objective is (near) zero. The classical results are given in propositions 1.2.1-1.2.4 in Bertsekas [1999]. Additionally, if the objective is a smooth function, i.e., the gradient is L-Lipschitz, the convergence rates can also be derived: for example, see proposition 3.3.1 in `https://www2.isye.gatech.edu/~nemirovs/Lect_OptII.pdf`.

[1] Actually this is a term related to formally defined Complexity/Capacity like VC-dimension/Rademacher etc.

[2] Please do not confuse finite capacity as finite cardinalty. There are many models that allow uncountably infinite members but have finite capacity. See for e.g., section 6.1 in Shalev-Shwartz and Ben-David [2014]!

[3] Even if the function is not differentiable, because of convexity, one-sided instantaneous slopes (subgradients) exist. Any such "subgradient" can be used in place of gradient. For example, the hinge loss $h(z) \equiv \max(0, 1 - z)$, is not differentiable at 0. But 0,-1 are instantaneous slopes from the positive and negative sides at 0. Any number between 0 and 1 happens to be a valid "subgradient".

4.1.1.1 BackProp

In some cases, like in artificial neural network models, one needs to compute the gradient of functions that are 'compositional' in nature. These can be computed efficiently by state-of-the-art automatic differentiation codes, which are efficient GPU-based optimized codes for applying chain rule of differentiation over a computational graph. Automatic differentiation specialized to ANNs is popularly known as back propagation (BackProp). Please refer section 20.6 in Shalev-Shwartz and Ben-David [2014] for more details.

4.1.2 ERM with non-parametric models

In the special case of non-parametric models (section 3.1.5), solving the ERM problem is trivial leading to the candidate (predictive) function parametrized by the training set, $\mathcal{D}$. For example:

$$\hat{f}_m^{ERM}({}^k\mathcal{N}_d)(x_i) \equiv y_i, \tag{4.2}$$

$$\hat{f}_m^{ERM}({}^k\mathcal{N}_d)(x) = \text{mode}(y_{i_1^*(x)}, \ldots, y_{i_k^*(x)}), \tag{4.3}$$

where $i_j^*(x)$ is the j^{th} nearest input in the training set to x as per the distance, d. Hence, training is same as efficiently memorizing the training set $\mathcal{D}$ (so that nearest neighbour queries at inference stage are efficient). The most promising and generic approaches for this are based on the so-called Locality Sensitive Hashing (LSH) [Gionis et al., 1999, Shrivastava and Li, 2014].

4.2 Stochastic Gradient Descent a.k.a. SGD

Refer
tion 14.5 in Sha
Shwartz
Ben-David [201

It so happens that there is a fundamentally different way to solve Stochastic Optimization problems, like (2.4), (2.11), which is by using SGD. The general algorithm, which is not limited to solving those like (2.4), (2.11), is presented in Shalev-Shwartz and Ben-David [2014] on page 157. A particular variant for solving (2.4), (2.11) is presented on page 163 in the same book. Equation (7) in Shalev-Shwartz et al. [2009][4], which proves the correctness of SGD, is summarized below:

Theorem 4.2.1. *With slight abuse of notation let us denote the (true) risk with function corresponding to the parameter w as $R[w]$. If $R[w]$ is a convex*

[4] In this paper it is also shown that if estimation error decays to zero with ERM, then so does with SGD. Interestingly, there are models/inductive-biases/hypothesis-classes where estimation error decays to zeros with SGD, whereas not the same with ERM!

and a L-Lipschitz continuous function in w, then, with probability atleast $1-\delta$, we have: Estimation error with the SGD solution is atmost $O\left(BL\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{m}}\right)$. Here, $B = ||w^0 - w^*||$, is the goodness of the initialization[5].

The key-take home from this theorem is: estimation error of SGD solution will be arbitrarily small, with arbitrarily high probability, provided no. samples m are large enough. L, the Lipschitz constant, is now a measure of "complexity" of loss+model combination.

If the risk is not a convex function of the parameter, then one typically assumes L-Smoothness and bounded gradient variance, leading to $O(1/m^{1/4})$ bound on estimation error Ghadimi and Lan [2013].

We then discussed some practical aspects of applying SGD[6]. Since gradient descent for solving ERM problem uses gradient of loss wrt. all examples at every iteration, and SGD uses one per iteration, a trade-off is achieved by using a few samples per iteration, called as mini-batch. Here, the SGD direction is gradient of the average loss over the samples in the minibatch. Also, the classical SGD scans the entire training set only once. A pass through training set is called an epoch. In practice, SGD is run for multiple epochs with different random order of samples. Both these tricks lead to the so-called mini-batch SGD, which is the default variant used in practice. Also, though the theorems give a default value for the step-size etc., the hyperparameters like step-size, no. epochs, mini-batch size etc. are typically tuned using validation set procedure[7]. While classical SGD (one epoch) is not expected to overfit, the minibatch SGD run for multiple variants may overfiit. So it is important to control the number of epochs appropriately (like using validation).

4.2.1 Online Gradient Descent

Consider the online learning setup in section 2.6. If the model, $\mathcal{F}$, is parametrized by a parameter, say w, then one can simply run the SGD algorithm and use the SGD iterates to predict labels at the respective iteration. This is because vanilla SGD (1 epoch) also happens to employ exactly one input at every iteration. However, one may wonder if such Online Gradient Descent will have any meaningful

[5] R can be assumed to be same as the regularization hyperparameter, W, in $l2$ regularized linear models if initial parameter is zero. And in this case, one can show that the complexity of the model involves RL term even in the ERM case!

[6] Interested students may read the seminal work in Bottou [2010] for other details.

[7] Interested students may refer `http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture2.pdf` for more details.

upper bounds on the cumulative loss. Interestingly, theorem 21.25 in Shalev-Shwartz and Ben-David [2014] shows that the excess cumulative loss compared to the best possible candidate in $\mathcal{F}$, known as the regret, is upper bounded by $O(WL\sqrt{m})$, whenever the loss is a convex function in terms of the parameter, and is L-lipschitz, and $\|w\| \leq W$ (regularized model). Hence, SGD is a useful algorithm both in batch learning as well as in online learning.

4.3 Maximum Likelihood Estimation (MLE)

Consider the SAA strategy, where (2.11) is approximated as:

$$\arg\max_{q \in \mathcal{Q}} \frac{1}{m} \sum_{i=1}^{m} \log\left(q(x_i)\right), \tag{4.4}$$

Since $\sum_{i=1}^{m} \log\left(q(x_i)\right) = \log\left(\Pi_{i=1}^{m} q(x_i)\right)$ and log is monotonic, (4.4) is sometimes[8] interpreted as "maximizing the likelihood of the training data". Hence, (4.4) is popularly known as Maximum Likelihood Estimation (MLE). The MLE problem (4.4) can be solved using GD (see section 4.1.1) or its variants. Please refer sections 4.2.1-4.2.2, 6.2.5 in Murphy [2022] for other details on MLE.

If the model is "bias-free", i.e., consists of all possible likelihoods, then an optimal solution of MLE turns out to be the so-called empirical likelihood with $\mathcal{D}$, defined as $\hat{p}_{\mathcal{D}}(z) \equiv \begin{cases} \frac{1}{m} & \text{if } z \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases}$. Though the law of large numbers shows asymptotic convergence of $\hat{p}_{\mathcal{D}}$ to p^*; with finite samples, $\hat{p}_{\mathcal{D}}$ seems pretty useless for any ML task. Hence the bias inherent in a probabilistic model plays an important role in MLE/SAA (see also the last para in section 4.1).

For some simple cases, MLE has analytical solutions: for e.g., see sections 4.2.3, 4.2.4, 4.2.5, 4.2.6 in Murphy [2022] for solving problem MLE with Bernoulli, Multinoulli, and the Gaussian models.

4.4 Maximum Conditional Likelihood Estimation (MCLE)

Consider the SAA strategy, where (2.14) is approximated as:

$$\arg\max_{q \in \mathcal{Q}} \frac{1}{m} \sum_{i=1}^{m} \log\left(q(y_i/x_i)\right). \tag{4.5}$$

[8] Assuming, samples in $\mathcal{D}$ remain independent wrt. every likelihood in the model.

Solving this to obtain the parameters of a discriminative models is known as Maximum Conditional Likelihood Estimation (MCLE). The MCLE problem (4.5) can be solved using GD (see section 4.1.1) or its variants.

4.5 Multi-arm Bandit Algorithms

ϵ greedy algorithm: Here, the arm with highest Q_t is picked with probability $1-\epsilon$, while at random with probability ϵ.

Another way to have low regret (2.16) is to have $T_i \propto q(a_i)$. Accordingly, we have the so-called soft-max algorithm: where the probability of picking the i^{th} arm is equal to $\frac{e^{Q_t(a_i)}}{\sum_{j=1}^{k} e^{Q_t(a_j)}}$. Both these algorithms have nice regret bounds, but the one that is more popular and insightful is the so-called Upper Confidence Bound (UCB) algorithm.

4.5.1 UCB Algorithm

In order to motivate the algorithm, we derived the basic bounds that govern the mismatch between Q_t and q: With probability atleast $1-\delta$, we have that[9]

$$q(a_i) \leq Q_s(a_i) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2T_i(s)}}. \tag{4.6}$$

We define the RHS in the above as the Upper Confidence Bound (UCB)[10]. Similarly, with probability atleast $1-\delta$, we have that $q(a_i) \geq Q_s(a_i) - \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2T_i(s)}}$. The RHS in this may be refered to as the Lower Confidence Bound (LCB).

Since we know that with high probability, the true mean lies within the LCB and UCB, one may propose branch-and-bound style algorithms that improve over soft-max by avoiding arms with too low quality. However, there is a more popular and elegant algorithm based on these bounds, called the UCB algorithm: at any round, pull the arm with highest UCB. Such an algorithm would perform both

[9] W.l.o.g., we assume that the rewards are between 0 and 1.

[10] The proof technique used for deriving the bound is called the Chernoff bounding technique. This is summarized in https://en.wikipedia.org/wiki/Chernoff_bound#The_generic_bound. This bound is in terms of moment-generating-functions (mgfs). By the Hoeffding lemma (See https://en.wikipedia.org/wiki/Hoeffding%27s_lemma), such finitely supported random variables are sub-Gaussian(Interested students may refer https://ocw.mit.edu/courses/mathematics/18-s997-high-dimensional-statistics-spring-2015/lecture-notes/MIT18_S997S15_Chapter1.pdf), which provides the bound on the mgf. This leads to 4.6

exploitation (first term in UCB) and exploration (second term in UCB). This is because UCB will be higher if either the average is higher or the arm is chosen a very few times.

Given that UCB is a good strategy, the one question is the choice of δ in (4.6). It is clear that during initial iterations, high delta would simple mean arbitrary choice for arms because the variance in (4.6) will be too high. Accordingly, we choose $\delta = \frac{1}{t}$, in the t^{th} round. Please refer fig1 in this paper for details of the UCB algorithm. Please refer theorem 1 in Auer et al. [2002] for details of regret bound with the UCB algorithm or this handwritten-notes: see "TechnicalDerivations− >UCB regret bound" in `https://1drv.ms/o/s!Au6Zdrbq2x4ph7JQUbky8SW6ygknAg`[11].

The most interesting take homes from this regret analysis are:

1. The regret after m rounds is bounded by $O(\log(m))$, which is in some sense is a negligible increase. Less interestingly, this also shows that UCB figures out the best arm(s) asymptotically.

2. The bound on regret is lower, if the margin between the value of the best arm(s) and the rest are higher. This is indeed insightful and illustrates the goodness of this bound. Accordingly, Multi-arm Bandit problems where margin is high are easier.

Though Bandits do not expose all aspects of a full RL problem, it does highlight the important explore-exploit trade-off, which makes RL a unique and interesting problem to study. Interested students may study Sutton and Barto [2018], which is an excellent book on basics of RL.

[11] Same as the version in `https://www.cse.iitb.ac.in/~shivaram/teaching/old/cs747-a2018/resources/ucb-regret.pdf`.

Chapter 5

Named ML settings

This chapter is list of instantiations of settings in chapter 2. In coding lingo, if chapter 2 is a list of `classes`, then this chapter is a list of `objects` instantianted from those `classes`.

5.1 Linear Regression

The set-up of linear regression is detailed below:

1. The task is that of regression. More specifically, the label space, $\mathcal{Y} = \mathbb{R}$.
2. Loss function is the squared loss (2.3).
3. Model is the linear model (3.1). The feature map, $\phi : \mathcal{X} \mapsto \mathbb{R}^n$, is assumed to be known (designed using domain knowledge).
4. Assume the supervised learning assumption (see section 2.1.2). Let training set be $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$.

Now, the stochastic optimization, (2.4), in this case can be solved using SGD (section 4.2) or using ERM (section 4.1). In the latter case, the ERM problem can be written as:

$$\min_{w \in \mathbb{R}^n} \quad \frac{1}{m} \sum_{i=1}^{m} \left(w^\top \phi(x_i) - y_i \right)^2 . \tag{5.1}$$

Please refer section 9.2 in Shalev-Shwartz and Ben-David [2014] for details of linear regression[1]. The ERM solutions turn out to be those of the Normal equation

[1] Also see section 11.3.1 in Mohri et al. [2012], .

corresponding to the set of linear equations given by: $X^\top w = y$, where X is the data matrix with columns as the training input vectors, and y has entries as the training labels. In the special case $XX^\top$ is invertible, ERM has a unique solution given by the left inverse of $X^\top$ (least square solution).

Once the ERM solution, $\hat{w}_m$, is obtained, the label for any input, $x \in \mathcal{X}$, shall be predicted as: $\hat{w}_m^\top \phi(x)$. The performance of this ERM regressor may be evaluated using explained variance over appropriate test set (see section 2.1.3).

5.2 Linear Regression (Generative Modelling)

The set-up of generative linear regression[2] is detailed below:

1. The task is that of regression. More specifically, the label space, $\mathcal{Y} = \mathbb{R}$ or in case of so-called multiple regression, it is $\mathcal{Y} = \mathbb{R}^d$. $d = 1$ gives back usual regression.

2. Loss function is the squared loss (2.3).

3. Assume the input space, $\mathcal{X} \in \mathbb{R}^n$. $\phi(x) = x$ is the feature map.

4. Generative modelling is employed (see section 3.2.1). More specifically, the underlying (joint) likelihood $p^*(x, y)$ is modelled using the ($n + d$ dimensional) Gaussian model (see section 3.2.1.3).

5. Supervised learning assumption (for estimating the Bayes optimal, $f^*_{|\mathcal{G}}$), namely, training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, is a set of iid samples from the underlying likelihood p^*.

Since, the supervised learning assumption made above is exactly same as the unsupervised learning assumption for estimating p^* (see section 2.2), relative/cross entropy loss is employed for likelihood estimation, MLE (see section 4.3) is employed to train the Gaussian model to obtain the trained model $\mathcal{N}(\hat{\mu}(\mathcal{D}), \hat{\Sigma}(\mathcal{D}))$, where $\hat{\mu}(\mathcal{D}), \hat{\Sigma}(\mathcal{D})$ are the sample mean and covariance of the training data, $\mathcal{D}$.

Using equation (3.28) in Murphy [2022] we obtain the bayes optimal predictor (posterior mean) corresponding to this trained model as: $\hat{f}_m(x) = \hat{\mu}_2 +$

[2]This terminology is not popular, but we use it here as we feel it's appropriate: in the sense that it's the generative version of the (discriminative) linear regression of section 5.3. However, there is atleast one other alternative which could be given the same name: p(y) and p(x/y) are individually modelled using Gaussians. This is discussed in section 3.3 in Murphy [2022], and goes by the name linear Gaussian system.

$\hat{\Sigma}_{21}\hat{\Sigma}_{11}^{-1}(x-\hat{\mu}_1)$, where $\hat{\mu}(\mathcal{D}) = \begin{bmatrix} \hat{\mu}_1 \\ \hat{\mu}_2 \end{bmatrix}$ and $\hat{\Sigma}(\mathcal{D}) = \begin{bmatrix} \hat{\Sigma}_{11} & \hat{\Sigma}_{12} \\ \hat{\Sigma}_{21} & \hat{\Sigma}_{22} \end{bmatrix}$, defines the appropriate blocking of the sample mean and covariance. The performance of this regressor may be evaluated using explained variance over appropriate test set (see section 2.1.3).

5.3 Linear Regression (Discriminative Modelling)

Refer: s
tion 11.2 in M
phy [2022].

The set-up of discriminative linear regression is detailed below:

1. The task is that of regression. More specifically, the label space, $\mathcal{Y} = \mathbb{R}$.
2. Loss function is the squared loss (2.3).
3. Assume a feature map $\phi : \mathcal{X} \mapsto \mathbb{R}^n$ is given.
4. Dsicriminative (probabilistic) modelling is employed (see section 3.2.2). More specifically, the posterior likelihood $p^*(y/x)$ is modelled using the (discriminative) Gaussian model (see section 3.2.2.2).
5. Assumes supervised learning (for estimating the Bayes optimal, $f^*_{|\mathcal{G}}$), namely, training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, is a set of iid samples from the underlying likelihood $p^*(x, y)$.

Using the training data, MCLE is performed to obtain the parameters. Interestingly, the MCLE turns out to be exactly same as (5.1). Hence this is the discriminative version of the linear regression model. Once the MCLE solution, $\hat{w}_m$, is obtained, the label for any input, $x \in \mathcal{X}$, shall be predicted as: $\hat{w}_m^\top \phi(x)$. The performance of this ERM regressor may be evaluated using explained variance over appropriate test set (see section 2.1.3).

5.4 Ridge Regression

The set-up of ridge regression is detailed below:

1. The task is that of regression. More specifically, the label space, $\mathcal{Y} = \mathbb{R}$.
2. Loss function is the squared loss (2.3).

3. Model is the $l2$-regularized linear model (3.3). The feature map, $\phi : \mathcal{X} \mapsto \mathbb{R}^n$, the regularization hyperparameter, W, are assumed to be known (designed using domain knowledge or hyperparameter cross-validated).

4. Assume the supervised learning assumption (see section 2.1.2). Let training set be $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$.

Now, the stochastic optimization, (2.4), in this case can be solved using SGD (section 4.2) or using ERM (section 4.1). In the latter case, the ERM problem can be written as:

$$\begin{aligned} \min_{w \in \mathbb{R}^n} \quad & \tfrac{1}{m} \textstyle\sum_{i=1}^m \left(w^\top \phi(x_i) - y_i \right)^2, \\ \text{s.t.} \quad & \|w\|_2 \leq W. \end{aligned} \tag{5.2}$$

Sometimes one prefers solving the following equivalent form, with λ, the so-called ridge hyperparameter appropriately designed/cross-validated:

$$\min_{w \in \mathbb{R}^n} \quad \frac{\lambda}{2} \|w\|_2^2 + \sum_{i=1}^m \left(w^\top \phi(x_i) - y_i \right)^2 \tag{5.3}$$

Please refer section 13.1.1 in Shalev-Shwartz and Ben-David [2014] for details of ridge regression[3]. It can be easily shown that the solution of (5.3) is always unique[4] and as $\lambda \to 0$, the solution turns out to be exactly same[5] as the Moore-Penrose (pseudo) inverse based solution: $w = \left(X^\top \right)^\dagger y$ (i.e., the min-norm least squares solution), where X is the data-matrix whose columns are $\phi(x_1), \ldots, \phi(x_m)$. Interested students may refer sections 11.3.1.1-11.3.1.2 in Murphy [2022] for numerically solving (5.3) and to know why such algorithms are numerically stable.

Once the ERM solution, $\hat{w}_m$, is obtained, the label for any input, $x \in \mathcal{X}$, shall be predicted as: $\hat{w}_m^\top \phi(x)$. The performance of this ERM regressor may be evaluated using explained variance over appropriate test set (see section 2.1.3).

5.4.1 Kernelized Ridge Regression

This is exactly same as Ridge regression except that the feature map is implicitly specified by a (well-designed) kernel. So the model is given by (3.6) and the ERM (3.5) simplifies as:

$$\min_{\alpha \in \mathbb{R}^m} \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j k\left(x_i, x_j\right) + C \sum_{i=1}^m \left(y_i - \sum_{j=1}^m \alpha_j k\left(x_i, x_j\right) \right)^2, \tag{5.4}$$

[3] Also see section 11.3.1 in Mohri et al. [2012], .

[4] Please refer equations 11.12-11.14 in Mohri et al. [2012].

[5] For e.g., see `https://ds12.github.io/talks/lectures/lecture4.html#slide40`.

Once optimal α^* is obtained, the prediction function is $\hat{f}(x) = \sum_{i=1}^{m} \alpha_i^* k(x_i, x)$. Refer section 11.3.2 in Mohri et al. [2012], section 17.3.9 in Murphy [2022], for details.

5.5 Support Vector Regression

The set-up of support vector regression is detailed below:

1. The task is that of regression. More specifically, the label space, $\mathcal{Y} = \mathbb{R}$.

2. Loss function is the ϵ-insensitive loss[6]: $l_\epsilon(f(x), y) \equiv \max(0, |y - f(x)| - \epsilon)$. Unlike the square-loss, this loss is expected to be zero for most training samples. This is the defining characteristic of "support vector" methods. More specifically, the exact locations of training inputs where $|y_i - f(x_i)| \geq \epsilon$ are critical in the sense that if even we perturb their location by an infinitesimal amount, the loss value changes. Hence, these are called as Support Vectors (corresponding to f). Whereas, even if locations of training points where $|y_i - f(x_i)| < \epsilon$ change by an infinitesimal amount, the loss value will never change.

3. Model is the $l2$-regularized linear model (3.3). The feature map, $\phi : \mathcal{X} \mapsto \mathbb{R}^n$, the regularization hyperparameter, W (or equivalently C), are assumed to be known (designed using domain knowledge or hyperparameter cross-validated).

4. Assume the supervised learning assumption (see section 2.1.2). Let training set be $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$.

Now, the stochastic optimization, (2.4), in this case can be solved using SGD (section 4.2) or using ERM (section 4.1). In the latter case, the ERM problem can be equivalently written as:

$$\min_{w \in \mathbb{R}^n} \quad \frac{1}{2}||w||_2^2 + C \sum_{i=1}^{m} \max(0, |y_i - w^\top \phi(x_i)| - \epsilon) \tag{5.5}$$

Interested students may refer section 11.3.3 in Mohri et al. [2012] for more details.

[6] Please see figure 11.5 in Mohri et al. [2012] for an illustration of the loss function

5.5.1 Kernelized SVR

This is exactly same as SVR except that the feature map is implicitly specified by a (well-designed) kernel. So the model is given by (3.6) and the ERM (3.5) simplifies as:

$$\min_{\alpha \in \mathbb{R}^m} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k\left(x_i, x_j\right) + C \sum_{i=1}^{m} \max\left(0, |y_i - \sum_{j=1}^{m} \alpha_j k\left(x_i, x_j\right)| - \epsilon\right). \tag{5.6}$$

Once optimal α^* is obtained, the prediction function is $\hat{f}(x) = \sum_{i=1}^{m} \alpha_i^* k(x_i, x)$.

5.6 Neural Network based Regressors

In case of each of the above regression set-up, one can replace the linear model with a ANN model. i.e., replace $f(x) = w^\top \phi(x)$ by $f(x) = w^\top \left(\sigma\left(V_d^\top \left(\sigma\left(V_{d-1}^\top \ldots \sigma\left(V_1^\top x\right)\right)\right)\right)\right)$. SGD/ERM (along with backprop) can be used to find optimal parameters V, w. Interested students may refer section 13.2.4.4 in Murphy [2022] for more details.

5.7 Linear Classification

The set-up of linear (binary) classification is detailed below:

1. The task is that of binary classification. More specifically, the label space has two elements: $|\mathcal{Y}| = 2$.
2. Loss function is the 0-1 loss (2.2).
3. Model is the linear model (3.1), composed with the `sign` function. The feature map, $\phi : \mathcal{X} \mapsto \mathbb{R}^n$, is assumed to be known (designed using domain knowledge). This model is also known as Halfspace model or the Linear classifiers model:

   $$\mathcal{H}_{n,\phi} \equiv \left\{ f \mid f(x) = sign\left(w^\top \phi(x)\right) \ \forall\ x \in \mathcal{X}, \text{ for some } w \in \mathbb{R}^n \right\}, \tag{5.7}$$

   The composition with `sign` function ensures the functions are binary-valued. One of the class labels is (arbitrarily) associated with the function value 1 and the other is associated with the function value -1. Thus, elements of $\mathcal{H}$ can be understood as functions from input-space to label-space. Refer section 9.1 in Shalev-Shwartz and Ben-David [2014] for more details.

4. Assume the supervised learning assumption (see section 2.1.2). Let training set be $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. Here, $y_i \in \{-1, 1\}$ (refer label mapping described above).

Now, the stochastic optimization, (2.4), in this case can be solved using ERM (section 4.1):

$$\min_{w \in \mathbb{R}^n} \quad \frac{1}{m} \sum_{i=1}^{m} 1_{y_i w^\top \phi(x_i) < 0}. \tag{5.8}$$

This problem turns out to be a computationally hard problem[7]. However, in the special case where the training data is linearly separable, i.e., $\exists w \ni y_i w^\top x_i \geq 0 \ \forall \ i = 1, \ldots, m$, it can either be posed as a Linear Program (refer section 9.1.1 in Shalev-Shwartz and Ben-David [2014]), or solved (exactly) using the so-called Perceptron algorithm that converges in finite no. iterations (refer section 9.1.2 in Shalev-Shwartz and Ben-David [2014]).

A popular idea to mitigate the computational complexity is to use a surrogate loss function that are easier to optimize (like hinge-loss, logistic loss etc.).

5.7.1 Logistic Regression

Refer: s
tion 9.3 in Sha
Shwartz and B
David [2014].

This is a setup exactly same as that in linear (binary) classification, but the loss function is the so-called logistic loss: $l(f(x), y) \equiv \log\left(1 + e^{-yf(x)}\right) \ \forall \ y \in \{-1, 1\}, \ x \in \mathcal{X}$, instead of the 0-1 loss. Observe that, through this definition, we generalized our notion of loss functions (logistic loss no longer compares two labels; rather it compares the label with a *score* for the other). Accordingly, we generalize the notion of loss as a function $l : \mathcal{F} \times \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}_+$. The ERM problem (4.1) with logistic regression setup turns out to be:

$$\min_{w \in \mathbb{R}^n} \quad \frac{1}{m} \sum_{i=1}^{m} \log\left(1 + e^{-y_i w^\top \phi(x_i)}\right). \tag{5.9}$$

The above is an unconstrained convex program[8] with a differentiable objective. The (global) optimal solution, say w^{ERM}, of (5.9) can be found efficiently by running the Gradient Descent algorithm[9]. We noted how the update rule is a soft version of that in the perceptron algorithm.

[7] Infact a more comprehensive statement can be made: refer Feldman et al. [2009] for details.

[8] Interested students may refer section 10.2.3.4 in Murphy [2022] to understand why the objective is convex.

[9] Interested students make lookup `numpy.logaddexp`, `numpy.log1p` to know how to handle numerical overflow, underflow issues while running gradient descent with logistic loss.

However, we proved in lecture that in some cases (5.9) does not have any optimal solution. It is the special case where the training data is linearly separable. So logistic regression and the perceptron algorithm can be used in complementary scenarios, enabling linear classification in all situations.

5.8 (Naive) Bayes Classifier

er: sec- n 9.2 in Mur- [2022].

The set-up of Bayes classifier or Gaussian/Quadratic Discriminant Analysis (GDA/QDA) is detailed below:

1. The task is that of (multi-class) classification. More specifically, the label space has c elements: $|\mathcal{Y}| = c$.
2. Loss function is the 0-1 loss (2.2).
3. Assume the input space, $\mathcal{X} \in \mathbb{R}^n$. $\phi(x) = x$ is the feature map.
4. Generative modelling is employed (see section 3.2.1). More specifically, the underlying (joint) likelihood $p^*(x, y) = p^*(x/y)p^*(y)$ is modelled. The factors $p^*(x/y)$, for each $y \in \mathcal{Y}$, are modelled using the (n dimensional) Gaussian model (see section 3.2.1.3). Let the parameters for these c Gaussians be $(\mu_y, \Sigma_y), y \in \mathcal{Y}$. The factor $p^*(y)$ is modelled using a Multinoulli (see section 3.2.1.2), with parameter $\pi \in \Delta_c$.
5. Supervised learning assumption (for estimating the Bayes optimal, $f^*_{|\mathcal{G}}$), namely, training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, is a set of iid samples from the underlying likelihood p^*.

Estimation of the parameters $\pi \in \Delta_c$ can be done as follows. Create the dataset $\mathcal{D}_2 = \{y_1, \ldots, y_m\}$. The supervised learning assumption (above), implies that $\mathcal{D}_2$ are iid from $p^*(y)$. In other words, π can be estimated using the unsupervised learning paradigm (the MLE estimate $\hat{\pi}_y$ will simply be the fraction of times y appeared in $\mathcal{D}_2$).

Estimation of (μ_y, Σ_y) can be done as follows: create a dataset of inputs from the training set $\mathcal{D}$ that are labelled as y i.e., $\mathcal{D}_y \equiv \{x \mid (x, y) \in \mathcal{D}\}$. Then (μ_y, Σ_y) can be estimated from $\mathcal{D}_y$ using MLE (sample mean and sample covariance of datapoints in $\mathcal{D}_y$).

Using this trained model, $\hat{p}(x/y)\hat{p}(y)$, (with parameters as $\hat{\pi}$, $(\hat{\mu}_y, \hat{\Sigma}_y)\ \forall\ y \in \mathcal{Y}$), one can obtain $\hat{p}(y/x)$ using Bayes rule (refer (9.4) in Murphy [2022]). And, then predict using the mode of the this posterior (because mode is the Bayes optimal

with 0-1 loss). This final predictor is popularly known as the Bayes Classifier. In general, the decision boundaries with Bayes classifiers will be quadratic surfaces. For e.g., see figure 9.2(a) in Murphy [2022].

In the special case of tied parameters i.e., $\Sigma_y = \Sigma \; \forall \; y \in \mathcal{Y}$, the posterior has a simpler form as given in (9.8) in Murphy [2022], which is reproduced below:

$$p_\theta(y/x) = \frac{e^{w_y^\top \phi(x)}}{\sum_{y' \in \mathcal{Y}} e^{w_{y'}^\top \phi(x)}}, \tag{5.10}$$

where the w_ys are appropriate functions of the parameters of the Bayes classifier (see equations (9.5-9.8) in Murphy [2022] for details). The functional form in 5.10 is referred to as the softmax function (see section 2.5.2 in Murphy [2022] for details). The decision boundaries in this special case will be linear surfaces. For e.g., see figure 9.2(b) in Murphy [2022]. Hence this special case is known as Linear Discriminant Analysis (LDA).

The special case where the covariances, Σ_y, are all diagonal, the resulting classifier is known as the Naive Bayes Classifier. Covariances of the class-conditionals, $p(x/y)$, being diagonal is same as saying the input features are uncorrelated given the class label. Since with Gaussian models uncorrelation is same as statistical independence, this assumption is same as the so-called Naive Bayes assumption: $p(x/y) = \Pi_{i=1}^{n} p(x^i/y)$, where x^i is the i^{th} feature/dimension of x.

5.9 Logistic Regression (Discriminative Modelling)

Refer:sec10.2, sec 10.3 in M phy [2022].

The set-up of Discriminative Logistic Regression is detailed below:

1. The task is that of (multi-class) classification. More specifically, the label space has c elements: $|\mathcal{Y}| = c$.
2. Loss function is the 0-1 loss (2.2).
3. Assume a feature map $\phi : \mathcal{X} \mapsto \mathbb{R}^n$ is given.
4. Discriminative (probabilistic) modelling is employed (see section 3.2.2). More specifically, the posterior likelihood $p^*(y/x)$ is modelled using the discriminative multinoulli model (see section 3.2.2.1).
5. Assume supervised learning (for estimating the Bayes optimal, $f^*_{|\mathcal{G}}$), namely, training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, is a set of iid samples from the underlying likelihood $p^*(x, y)$.

Using the training data MCLE can be performed. The MCLE (4.5) in this case simplifies as:

$$\arg \min_{w_1,\ldots,w_c \in \mathbb{R}^n} \sum_{i=1}^{m} \left\{ \log \left(\sum_{y \in \mathcal{Y}} e^{w_y^\top \phi(x_i)} \right) - w_{y_i}^\top \phi(x_i) \right\}. \tag{5.11}$$

This can be solved using gradient descent[10]. In the special case $c = 2$, by change of variables, $w \equiv w_1 - w_2$, (5.11), is exactly same as the logistic regression ERM, (5.9). Once the MCLE solution, $(\hat{w}_1, \ldots, \hat{w}_c)$, is obtained, the optimal Bayes (prediction) function, which is the mode of posterior, can be computed using:

$$\hat{f}(x) \equiv \arg \max_{y \in \{1,\ldots,c\}} \hat{w}_y^\top \phi(x). \tag{5.12}$$

Hence the discriminating surfaces are linear (like in Bayes classifiers). Bayes classifier and discriminative logistic regression are our first examples where the inference (Prediction) problem is non-trivial and perhaps may be computationally costly for high c. Typically, one uses so-called LSH (locality sensitive hashing) Gionis et al. [1999], Shrivastava and Li [2014] techniques to efficiently solve the inference problem. This problem, (5.12), is popular as the Maximum Inner-Product Search (MIPS) problem and is closely related to the nearest neighbour search problem.

5.10 $l2$ Regularized Logistic Regression

This set-up is exactly same as Logistic Regression's set-up with the only difference that the $l2$ regularized linear model is employed instead of the linear model. Please refer section 13.8 in Mohri et al. [2012] for more details. Apart from the better estimation error bounds, the other advantage over (unregularized) logistic regression is that numerical algorithms are much more efficient and stable. Also, the solution exists even for linearly separable datasets as now parameters with high magnitude are penalized in the objective.

5.10.1 Kernelized Logistic Regression

This set-up is exactly same as $l2$ regularized logistic regression, except that the feature map is induced by a (well-designed) kernel. So the model is given by (3.6) and the ERM (3.5) simplifies as in the binary classification case:

$$\min_{\alpha \in \mathbb{R}^m} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) + C \sum_{i=1}^{m} \log \left(1 + e^{-y_i \sum_{j=1}^{m} \alpha_j k(x_i, x_j)} \right), \tag{5.13}$$

[10] Interested students may refer section 10.3.3 in Murphy [2022] for details.

Once optimal α^* is obtained, the prediction function is $\hat{f}(x) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i^* k(x_i, x)\right)$.

5.11 (Discriminative) Nearest Neighbour Classifier/Regressor

The set-up of Nearest Neighbour Classifier/Regressor is detailed below:

1. The task is that of (multi-class) classification or Regression.
2. Loss function is the 0-1 for classification and squared loss for regression.
3. Assume a distance function over input space, $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$ is given.
4. The nearest neighbour classification model, ${}^1\mathcal{N}_d$ from (3.8), or it generalization ${}^k\mathcal{N}_d$, is employed. In case of regression, instead of majority label, mean label is to be employed. The respective discriminative models are modelled using KDE. Details appear below.
5. Assume supervised learning (for estimating the Bayes optimal, $f^*_{\lceil \mathcal{G}}$), namely, training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, is a set of iid samples from the underlying likelihood $p^*(x, y)$.

As detailed in section (4.1.2), training is essentially storing the training data, $\mathcal{D}$ and the prediction function (for classification) is given by (4.2)-(4.3). In case of discriminative NN classification, the prediction function remains the same. And the estimate of $p^*(y/x)$ is simply the fraction of labels of the k neighbours belonging to y^{th} class (this is same as KDE with kernel as dirac delta). In case of discriminative NN regression, the estimate of $p^*(y/x)$ is simply the KDE with samples as labels of the k neighbours i.e., $\hat{p}(y/x) = \frac{1}{k}\sum_{i=1}^{k} \kappa_h(y - y_i^{'})$, where κ_h is a smoothing kernel with smoothness-parameter h, and $y_1^{'}, \ldots, y_k^{'}$ are the labels of the k nearest neighbours to x. And the corresponding prediction function will be the average label of these labels, $\frac{1}{k}\sum_{i=1}^{k} y_i^{'}$.

If during training $\mathcal{D}$ is stored in rudimentary ways, then one may need sophisticated nearest neighbour searching codes like `https://github.com/facebookresearch/faiss` during the inference stage. Else, if during training $\mathcal{D}$ is stored using sophisticated LSH data structures [Gionis et al., 1999, Shrivastava and Li, 2014], then inference is trivial (querying the data structure[11].).

[11] Ball tree is an alternative data structure `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html#sklearn.neighbors.BallTree`

Theorem 19.3 in Shalev-Shwartz and Ben-David [2014] provides a formal justification of why/when this 1-NN classification generalizes well. The encouraging part of the result is that as $m \to \infty$, the true risk with 1-NN classification is atmost twice that of the (unrestricted) Bayes optimal. However, unfortunately, the convergence rate (known as sample complexity) is plagued with the curse of dimensionality i.e., the bound is meaningless/useless when dimensionality of input space is high. Though k-NN ($k > 1$) often works better than 1-NN in practice (with appropriate k), in terms of sample complexity bounds it is the same (apart from the constants involved).

5.11.1 Kernelized Nearest Neighbors

Though distance functions are more general than inner-products (and hence, kernels), sometimes a well-designed kernel may be readily available for a domain; whereas a distance function is not available readily. Then one can use the kernel induced distance: More specifically, $d(x,z) \equiv \|\phi(x) - \phi(z)\| = \sqrt{\langle \phi(x), \phi(x) \rangle - 2\langle \phi(x), \phi(y) \rangle + \langle \phi(y), \phi(y$ $\sqrt{k(x,x) + k(y,y) - 2k(x,y)}$. Such a d will be a valid distance over $\mathcal{X}$, except that it may not satisfy the positive-definiteness (positivity) property[12].

5.12 Generative KDE Regression

er:sections
3.5.1 in Mur-
[2022]

The set-up of generative KDE regression is detailed below:

1. The task is that of regression. More specifically, the label space, $\mathcal{Y} = \mathbb{R}$ or in case of so-called multiple regression, it is $\mathcal{Y} = \mathbb{R}^d$. $d = 1$ gives back usual regression.

2. Loss function is the squared loss (2.3).

3. Assume a distance function over input space, $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$ is given.

4. $p^*(x,y)$ is modeled using KDE. The smoothing kernel, κ, over $\mathcal{X} \times \mathcal{Y}$ is assumed to be the product of two kernels, κ_1, κ_2, over $\mathcal{X}, \mathcal{Y}$ i.e., $\kappa(x,y) = \kappa_1(x)\kappa_2(y)$. κ_1, κ_2 (along with their bandwidths) are fixed apriori.

[12] See `https://en.wikipedia.org/wiki/Metric_space`. One can show that this will be a valid distance, satisfying the positive definitess property also, with a Gaussian kernel. Further, in this case, the distance simplifies as $\sqrt{2 - 2k(x,y)}$.

5. Supervised learning assumption (for estimating the Bayes optimal, $f^*_{|\mathcal{G}}$), namely, training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, is a set of iid samples from the underlying likelihood p^*.

The (trained) model is $\hat{p}(x, y) \equiv \frac{1}{m} \sum_{i=1}^{m} \kappa_1(x - x_i)\kappa_2(y - y_i)$. The corresponding marginal, $\hat{p}(x) = \frac{1}{m} \sum_{i=1}^{m} \kappa_1(x - x_i)$ and $\hat{p}(y/x) = \frac{\sum_{i=1}^{m} \kappa_1(x-x_i)\kappa_2(y-y_i)}{\sum_{i=1}^{m} \kappa_1(x-x_i)}$. The corresponding Bayes optimal (prediction function) is $\hat{f}(x) = \sum_{i=1}^{m} \rho_i(x) y_i$, where $\rho_i(x) \equiv \frac{\kappa_1(x-x_i)}{\sum_{j=1}^{m} \kappa_1(x-x_j)}$. So the prediction is simply a weighted interpolation of the training set labels. With Gaussian smoothing kernel, these weights are inversely proportional to the distance of the corresponding inputs to x. In this sense, it is a smoothed version of KDE regression and discriminative KDE regression in section 5.11.

5.13 Generative KDE Classification

The set-up of generative KDE classification is detailed below:

1. The task is that of classification.
2. Loss function is the 0-1 loss.
3. Assume a distance function over input space, $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$ is given.
4. $p^*(x/y)$ is modeled using KDE for every class. The smoothing kernel, κ, over $\mathcal{X}$ is assumed same for all classes. $p^*(y)$ is modelled using KDE with dirac-delta kernel.
5. Supervised learning assumption (for estimating the Bayes optimal, $f^*_{|\mathcal{G}}$), namely, training data, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, is a set of iid samples from the underlying likelihood p^*.

The (trained) class-conditional model $\hat{p}(x/y) = \frac{1}{m_y} \sum_{i:y_i=y} \kappa(x - x_i)$. And trained label prior $\hat{p}(y) = \frac{m_y}{m}$. Hence mode of posterior is $\hat{f}(x) = \arg\max_{y \in \mathcal{Y}} \sum_{i:y_i=y} \kappa(x - x_i)$. If the kernel is the so-called, balloon kernel[13] (see section 16.3.4 in Murphy [2022]), then this is same as k-NN classification. Hence the generative KDE classification may be understood as smoothened (discriminative) k-NN classification in section 5.11.

[13] Kernel evaluates to $1/k$ for the k nearest neighbours and zero otherwise.

5.14 SVM Classification

The set-up of Support Vector Machine (SVM) (binary) classification is detailed below:

1. The task is that of binary classification. More specifically, the label space has two elements: $|\mathcal{Y}| = 2$.

2. Loss function is the hinge loss: $l(f(x), y) = \max(0, 1 - yf(x))$. Unlike, logistic loss, we expected this loss to be zero for most of the training samples. This is infact the defining characteristic for support vector methods. More specifically, the exact locations of training inputs where $y_i f(x_i) \leq 1$ are critical in the sense that if even we perturb their location by an infinitesimal amount, their loss value changes. Hence, these are called as Support Vectors (corresponding to f). Whereas, even if locations of training points where $y_i f(x_i) > 1$ change by an infinitesimal amount, their loss value will remain 0.

3. Model is the $l2$ regularized linear model (3.3), composed with the sign function. The feature map, $\phi : \mathcal{X} \mapsto \mathbb{R}^n, W > 0$, are assumed to be known (designed using domain knowledge or cross-validated for). This model is also known as Large/Fat margin classifier model or gap-tolerant classifier model:
(5.14)
$$\mathcal{M}_{n,\phi,W} \equiv \left\{ f \mid f(x) = sign\left(w^\top \phi(x)\right) \ \forall\ x \in \mathcal{X}, \text{ for some } w \in \mathbb{R}^n, \|w\|_2 \leq W \right\},$$

   Visually/Geometrically, these classifiers can be understood as hyperplane classifiers, such that the distance (margin/gap) between the corresponding supporting hyperplanes is atleast W. The hyperplanes defined by $w^\top \phi(x) = \pm 1$ are known as the supporting hyperplanes corresponding to the hyperplane $w^\top \phi(x) = 0$. Recall that an input incurs zero hinge loss if and only if $|w^\top \phi(x)| \geq 1$, hence the threshold scores $w^\top \phi(x) = \pm 1$ are critical and worth being named. Geometrically, the support vectors, defined above, will either lie on the supporting hyperplanes or in the halfspace below (above) in case of positively (negatively) labelled points.

4. Assume the supervised learning assumption (see section 2.1.2). Let training set be $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. Here, $y_i \in \{-1, 1\}$ (refer label mapping described above).

Now, the stochastic optimization, (2.4), in this case can be solved using SGD[14]. Alternatively the ERM problem is:

$$\begin{aligned} \min_{w\in\mathbb{R}^n} \quad & \tfrac{1}{m}\textstyle\sum_{i=1}^m \max\left(0, 1 - y_i w^\top \phi(x_i)\right), \\ \text{s.t.} \quad & \|w\|_2 \leq W. \end{aligned} \tag{5.15}$$

Sometimes one prefers solving the following equivalent form, with C, the hyperparameter appropriately designed/cross-validated:

$$\min_{w\in\mathbb{R}^n} \quad \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^m \max\left(0, 1 - y_i w^\top \phi(x_i)\right) \tag{5.16}$$

Or equivalently as:

$$\begin{aligned} \min_{w\in\mathbb{R}^n} \quad & \tfrac{1}{2}\|w\|_2^2, \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^m \max\left(0, 1 - y_i w^\top \phi(x_i)\right) \leq B, \end{aligned} \tag{5.17}$$

where B is the hyperparameter. In the special case $B = 0$, the above can be simplified as:

$$\begin{aligned} \min_{w\in\mathbb{R}^n} \quad & \tfrac{1}{2}\|w\|_2^2, \\ \text{s.t.} \quad & y_i w^\top \phi(x_i) \geq 1 \;\forall\; i = 1, \ldots, m. \end{aligned} \tag{5.18}$$

The resultant (optimal) classifier in this special case is qualified as the hard-margin SVM classifier. This is because all the training datapoints lie outside the margin of the optimal classifier. And of all classifiers in the model, this optimal classifier has the maximum margin between the supporting hyperplane. This is because the margin between $w^\top\phi(x) = 1$ and $w^\top\phi(x) = -1$ turns out to be $\frac{2}{\|w\|_2}$. Likewise, the interpretation of (5.16) is maximally separating the positive and negative points while allowing some slack. Sometimes the optimal classifier corresponding to (5.16) is hence qualified as soft-margin SVM. Interested students may further read chapter 5 in Mohri et al. [2012] and chapter 15 in Shalev-Shwartz and Ben-David [2014].

5.14.1 Kernelized SVM

This is exactly same as the setup of SVM with feature map implicitly defined by a well-designed kernel. The ERM (3.5) simplifies as:

$$\min_{\alpha\in\mathbb{R}^m} \frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j k\left(x_i, x_j\right) + C\sum_{i=1}^m \max\left(0, 1 - y_i\sum_{j=1}^m \alpha_j k\left(x_i, x_j\right)\right), \tag{5.19}$$

[14] Please refer section 15.5 in Shalev-Shwartz and Ben-David [2014]. Minibatch versions of this algorithm with appropriate step-size schemes give state-of-the-art computational efficiency.

Once optimal α^* is obtained, the prediction function is $\hat{f}(x) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i^* k(x_i, x)\right)$.

5.15 Neural Network based Classifiers

One can simply replace the linear model with a FFNN model in binary logistic regression/SVM setups. i.e., replace $f(x) = w^\top \phi(x)$ by $f_{w,V}(x) = w^\top \left(\sigma\left(V_d^\top \left(\sigma\left(V_{d-1}^\top \ldots \sigma\left(V_1^\top x\right)\right)\right)\right)\right)$. SGD/ERM (along with backprop) can be used to find optimal parameters V^*, w^*. Any input x can be classified using $\text{sign}(f_{w^*,V^*}(x))$.

One can even perform discriminative logistic regression using ANNs: the idea is to replace the linear models employed for the parameters of the label posterior by FFNNs. More specifically, $p(y/x) \propto e^{w_y^\top \left(\sigma\left(V_{y,d_y}^\top \left(\sigma\left(V_{y,d_y-1}^\top \cdots \sigma\left(V_{y,1}^\top x\right)\right)\right)\right)\right)}$. Hence, if it is a c class problem, we would employ c FFNNs. Infact, it is common to tie the hidden node parameters across all the c FFNNs, so that the effective feature map over x is the same, exactly like in linear model based logistic regression i.e., $p(y/x) \propto e^{w_y^\top \left(\sigma\left(V_d^\top \left(\sigma\left(V_{d-1}^\top \cdots \sigma\left(V_1^\top x\right)\right)\right)\right)\right)}$. Because the parameters are tied (and hence depth/width are same for all FFNN), it is convenient to understand this model as one FFNN with c outputs. The nodes representing these outputs are said to form the output layer. Further it is popular to understand the outputs as directly $\frac{e^{w_y^\top \left(\sigma\left(V_d^\top \left(\sigma\left(V_{d-1}^\top \cdots \sigma\left(V_1^\top x\right)\right)\right)\right)\right)}}{\sum_{y'=1}^{c} e^{w_{y'}^\top \left(\sigma\left(V_d^\top \left(\sigma\left(V_{d-1}^\top \cdots \sigma\left(V_1^\top x\right)\right)\right)\right)\right)}}$ rather than as $w_y^\top \left(\sigma\left(V_d^\top \left(\sigma\left(V_{d-1}^\top \ldots \sigma\left(V_1^\top x\right)\right)\right)\right)\right)$. In such a case, since the outputs are obtained using the soft-max function, it is refered to as soft-max layer. As usual, parameters can be learnt using SGD/ERM (via backprop). Refer sections 13.2.4.1-13.2.4.3 in Murphy [2022] for more details.

5.16 Auto-Encoder

The set-up of an Auto-encoder/Autoassociative Neural Network is as follows:

1. Input-space is $\mathbb{R}^n$.
2. Reconstruction loss is the squared loss.
3. The encoder and decoder models are both FFNNs (section 2.4).

Training is performed using minbatch SGD to obtain optimal parameters for encoder network, w^*. The representation of x is $f_{w^*}(x)$, computed by forward-pass

through the learned encoder network. Please refer section 12.4.2 in Bishop [2006] for details.

5.17 Principal Component Analysis (PCA)

The set-up of Principal Component Analysis (PCA) is as follows (special case of auto-encoder):

1. Input-space is $\mathbb{R}^n$.
2. Reconstruction loss is the squared loss.
3. The encoder model is all linear functions $\mathbb{R}^n \mapsto \mathbb{R}^d$ and the decoder model is all linear functions $\mathbb{R}^d \mapsto \mathbb{R}^n$, $d \ll n$ (section 2.4). Hence parameters are the matrices $U \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{d \times n}$ that parametrize the linear models.

Training this model (learning the parameters) can be done via finding top d eigenvectors of the sample correlation matrix. This is purely a linear algebra result. Please refer section 23.1 in Shalev-Shwartz and Ben-David [2014] for details.

5.17.1 Kernelized PCA

Please refer section 15.2 in Mohri et al. [2012] and section 23.1.1 in Shalev-Shwartz and Ben-David [2014] for details.

5.18 1-class SVM

This is a set-up useful for support estimation:

1. There is an underlying/generating likelihood, $p^*(x)$, and the training set is a set of m iid samples from it. Task is to estimate the support of p^*.
2. Motivated by exponential family models, here the idea is to indirectly model the underlying/generating likelihood, p^*, by modeling a potential function, $f^*(x)$, such that $p^*(x) \propto e^{f^*(x)}$. One may use a regularized linear model or a kernel-based model or a FFNN model for $f^*(x)$. In the following, for simplicitly, the formulae are given with a (regularized) linear model: $f(x) \equiv w^\top \phi(x)$.

3. Hingeloss is employed to insist that training samples ought to having high likelihood accordingly to p^* (i.e., high value of $f^*(x)$): $l(w, x_i, 1) \equiv \max(0, 1 - w^\top \phi(x_i))$. Here, 1 denotes the (arbitrary) high value of the potential. Such a loss would promote $f(x_i) = w^\top \phi(x_i) \geq 1$.

The corresponding ERM problem simplifies as:

$$\min_{w \in \mathbb{R}^n} \quad \frac{1}{2}\|w\|_2^2 + C \sum_{i=1}^{m} \max\left(0, 1 - w^\top \phi(x_i)\right) \tag{5.20}$$

Since this is same as (5.16) when there is only one class i.e., $y_i = 1 \; \forall \; i$, this set-up is popular as 1-class SVM. Once this is sovled for optimal $\hat{w}$, support is estimated as $\mathcal{S} \equiv \{x \in \mathcal{X} \mid \hat{w}^\top \phi(x) \geq 1\}$. If a sample $x \notin \mathcal{S}$, it is called as an outlier/novel point.

5.18.1 Support Vector Clustering

The idea is to use post-process the high/low density information provided by a trained 1-class SVM to form clusters[15]:

1. Let $\mathcal{I}$ be the index set of those training points labeled "high" by the one-class SVM.
2. We initialize the adjacency matrix, A, representing cluster information with $A_{ij} = 0 \; \forall \; (i, j) \in \mathcal{I} \times \mathcal{I}$.
3. For each pair in $(i, j) \in \mathcal{I} \times \mathcal{I}$, we verify if the points in the line segment between the pair are also labeled "high". If so, then $A_{ij} = 1$.
4. Clusters are defined as the connected components in A. Note that such a procedure would indeed retrieve contiguous regions of high density (as per the model). Given a new datapoint, one can again check high/low density and connectivity to these clusters, in order to infer the cluster-id.

The key advantages of such a clustering, henceforth referred to as Support Vector Clustering (SVC), are that the number of clusters need not be known aprior and no strong assmptions on the shapes of clusters need to be made. The limitations are: i) tuning the hyperparameters[16] like C and kernel/network parameters is critical ii) the algorithm for cluster assignment needs to computationally scale well to large datasets.

[15] Refer `http://www.jmlr.org/papers/volume2/horn01a/horn01a.pdf` for details.

[16] One can still do something like CV but with disagreement over various folds wrt. clusters as the CV error.

5.19 GMM-based clustering

The set-up of GMM-based clustering is as follows:

1. Input space is $\mathcal{X} \in \mathbb{R}^n$. Cluster-ids are from the set $\mathcal{Y} = \{1, \ldots, k\}$.
2. Assumed there is an underlying joint likelihood, $p^*(x, y)$, relating inputs, $x \in \mathcal{X}$, to cluster-ids, $y \in \mathcal{Y}$.
3. The idea is to model the factors $p^*(x/y), p^*(y)$, by Gaussians and Multinoulli. Let the parameters of $p(x/y)$ be (μ_y, Σ_y) and the parameters of the Multinoulli be $\theta_1, \ldots, \theta_k$.
4. The training set is m iid samples from the marginal $p^*(x) = \sum_{y=1}^{k} p^*(x/y)p^*(y)$. Note that, as a consequence of the above modelling assumption, $p^*(x)$ is modelled using a GMM.

One can estimate the parameters of the (marginal) GMM using vanilla MLE, as samples from it are those in the training set. By our modelling assumptions, these parameters will be the parameters for the joint too. Let us denote this MLE based estimate of the joint be $\hat{p}(x, y)$. Clusters can then be inferred as follows:

1. Points, $x \in \mathcal{X}$, where $\hat{p}(x) < \epsilon$ can be declared as outliers (not belonging to any clusters). $\epsilon > 0$ is a hyperparameter. For the others:
2. Cluster of x can be declared as $\arg\max_{y \in \mathcal{Y}} \hat{p}(y/x)$.

One popular algorithm (more popular compared to SGD) for solving this MLE problem is the so-called EM algorithm, described below. The advantages of GMM-clustering over SVC are that the number of hyper-parameters are less (only k) and the inference is far more computationally easier. Moreover, using the posterior $\hat{p}(y/x)$, one can do "soft-clustering", whereas such an option is not readily available in SVC. However, the main limitation is that the cluster shapes are always elliptical (for high enough ϵ); whereas with SVC, there is no such restriction.

5.19.1 EM algorithm: Solving MLE with GMM

The MLE problem in case of GMMs turns out to be non-convex and computationally challenging to find the (global) optimal. EM (Expectation Maximization) algorithm is a very intuitive algorithm that at every iteration combines (processed) assumptions along with empirical evidence in the training samples to converge to

a meaningful solution in reasonable time. The convergence details are beyond the scope of this course and we will here be noting only the algorithm. Please refer equations (8.160, 8.165, 8.166) in Murphy [2022] for the EM algorithm for this case.

5.19.1.1 k-means algorithm

This is an simpler special case of the EM algorithm that is presented in section 21.3.1 in Murphy [2022] (see also section 21.4.1.1 in the same book). Interested students may refer this paper for related theoretical guarantees.

Chapter 6

Modelling and Model Selection

The goal here is to detail basic principles in ML modelling.

Models we studied in this course induce a bias in learning. For example, when the input space is Euclidean and the feature map is identity, the linear model introduces a (hard) bias towards linear functions in the sense that the Bayes optimal for the learning task is assumed to be a linear function (over the Euclidean input space). Likewise, the Gaussian model implicitly assumes that the underlying likelihood is Gaussian etc. The stronger the bias the restrictive/smaller the model is and vice-versa.

Models that introduce a strong bias may sometimes generalize very well. Infact, consider a model that only has one function that is the (unknown) Bayes optimal for the current learning task. Such a model is guaranteed to have 0 generalization error! In this sense, this is the best model possible for the current learning task. However, the issue is that while it works best for the current learning task, it will fail miserably on another learning task, whose Bayes optimal is very far/different from the current one. More importantly, designing such "best" models requires precise domain knowledge, which if available, one would not resort to ML anyways!

On the other extreme, there are models which are unbiased. For example, when the input space is binary, the Bernoulli model is completely unbiased. This is because all possible mass functions over binary values are included in the Bernoulli. Likewise with Multinoulli. Such models, which include all possibilities, are known as Universal Models. Such models are guaranteed to incur 0 modelling error on any learning task in the respective input/output space. While estimation error is expected to decay nicely with increasing samples in case of the Bernoulli and the Multinoulli models (law of large numbers); for most universal models, unfortunately, estimation error may be large and may not decay with increasing

samples. Recall the discussion from section 4.1 that unrestricted models may not learn (for example, capacity $\mathcal{C}$ is ∞. Refer theorem 4.1.1).

A middle ground is achieved by so-called Universal Approximators. These are models which are guaranteed to have modelling error always less than a given (arbitrarily low) tolerance and are guaranteed to have estimation errors that decay with increasing samples (i.e., capacity $\mathcal{C} < \infty$). Examples are linear models, exponential (log-linear) models, non-parametric models like nearest neighbour and kernel density estimator etc. More specifically, consider the linear models. Here, by choosing $\phi(x)$ as vector with entries as basis functions, one can arbitrarily well approximate functions in that space by including large enough number of basis functions. For example, $\phi(x)$ is the vector of all possible monomials upto degree r. In other words, for large enough r, the modelling error is small enough. Also, the estimation error with linear models (along with bounded loss functions) can be bounded by $O(\frac{n^r \log(m)}{m})$. Thus the estimation error indeed decays with m. Thus universal approximators seem to be a better choice than universal models.

However, most universal approximators are plagued with the curse of dimensionality. In other words, though their modelling error is tolerable, the rate at which the estimation error decays to 0 may be unacceptably slow, especially for high dimensions. This is clear from the estimation error bounds of $O\left(\frac{n^r \log(m)}{m}\right)$ and $O\left(\frac{\sqrt{n}}{m^{\frac{1}{n+1}}}\right)$ for the linear and nearest neighbour model respectively. Moreover, when modelling error has to be low, i.e., r is high, the estimation error may be very high. An exception to this is the Gaussian-kernelized SVM model, which is a universal approximator with acceptable bounds on estimation error, even in high dimensions. But even with this exceptional model, a general principle seems to be that as modelling error decreases, estimation error increases and vice-versa. This sometimes is referred to as the Bias-Complexity trade off. Interested students may see chapter 5 in Shalev-Shwartz and Ben-David [2014].

In summary, biased/restrictive models are best in terms of estimation error. Further, if the bias (i.e., the model) is well designed, then it will be *aligned* with the learning task, making the modelling error also low. For example, restricting candidates to linear functions is approximation-free in Ohm's Law experiment. For some kinds of domain knowledge some kinds of modelling techniques are well-suited. For example, when the form of the Bayes optimal is known, then the linear model with appropriate feature map is the best. When biases like sparisity/low-rank etc. need to be enforced, then regularized linear models seem the best. When modeling functions related human perception, neural network models seem the best. etc. When no particular domain knowledge is available or it is not clear how to encode the available domain knoweldge in the model as a bias, then, as a second option, universal approximators (that are not cursed) seem well-suited.

Sometimes, even after careful modelling one may be left with a choice for models. For example, whether to use 1-NN, 3-NN, 5-NN etc. Or which smoothing kernel bandwidth needs to be used? or which regularization hyperparameter needs to be employed? This is answered by the task of model selection.

6.1 Model Selection

Even after careful design, the values of hyperparameters are typically unknown (domain knowledge may not be specific enough). Also, one may not have enough domain knowledge to make a choice between the models in these various frameworks. In such a case, one has to deal with the problem of hyper-parameter estimation or that of selecting the "best" of these seemingly equivalent[1] models. This problem is known as the problem as Model Selection. Let the models (hyperparameters) be $\mathcal{M}_1(\vartheta_1), \ldots, \mathcal{M}_n(\vartheta_n)$.

Since training data is reserved for parameter estimation, and domain expertise is used-up in model design, one needs access to extra independent information, called the Validation/Development (Dev) data for performing hyperparameter estimation or Model selection. We assume that the samples in the Validation and Training sets are all iid from the same underlying, unknown, $p^*(x, y)$. Section 11.2 in Shalev-Shwartz and Ben-David [2014] is an excellent reference for model selection.

With this set-up, hyper-parameter estimation or model selection can simply be done using ERM but now using Validation data. This is called as Validation procedure:

1. Using Training data, we estimate (using ERM/MM/MLE/MCLE) the parameters in (with) each model (hyperparameter): $\hat{\theta}_1, \ldots, \hat{\theta}_n$.
2. Using Validation/Dev data, we estimate (using ERM) the hyperparameter i.e., pick the model/hyperparameter whose $\hat{\theta}$ gives least average Validation set error. Let the index of this model (hyperparameter) be $\hat{i}$.
3. The final prediction function is that obtained by performing parameter estimation in the $\hat{i}^{th}$ model using ERM/MM/MLE/MCLE over the entire training and validation data together. Let this parameter be $\hat{\theta}$.

One shortcoming with the above procedure is that the hyperparameter estimation (model selection) is done using only m_v no. datapoints, though the final

[1] Equivalent in terms of satisfying the domain knowledge based constraints.

parameter estimation is done using $m_t + m_v$ no. datapoints. Here, m_t, m_v denote the sizes of the training, validation sets respectively. Using clever re-sampling one can handle this asymmetry and make it appear as if $m_t + m_v$ datapoints are used for both parameter and hyperparameter estimation. An example of such a clever technique is k-fold Cross-Validation (CV). Please refer section 7.10 in Hastie et al. [2001] for details. When number of folds is same as $m_t + m_v$, this procedure is called as Leave One Out (LOO) CV. In general, given enough samples, LOO error is a better estimate for the true risk than k-fold CV, which is in turn a better estimate than the validation set error.

We finally commented that questions like which set of models are to be considered (hyperhyperparameter estimation :) can be again answered if additional data (hypervalidation data :) is given and so on... However, usually one stops at the level of hyperparameter estimation. Hence it is extremely important to decide on the set of models to be considered purely based on domain expertise and other learning considerations, and never after number crunching over the training/validation datasets!

Bibliography

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352. URL https://doi.org/10.1023/A:1013689704352.

D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT*, 2010.

V. Feldman, V. Guruswami, P. Raghavendra, and Yi Wu. Agnostic learning of monomials by halfspaces is hard. *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 385–394, 2009.

Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4): 2341–2368, 2013.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 518–529. Morgan Kaufmann, 1999. URL http://www.vldb.org/conf/1999/P49.pdf.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009. ISBN 9780262013192. URL https://books.google.co.in/books?id=7dzpHCHzNQ4C.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258.

Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL probml.ai.

Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. URL probml.ai.

Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014. ISBN 1461346916.

Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

D.W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. A Wiley-interscience publication. Wiley, 1992. ISBN 9780471547709. URL https://books.google.co.in/books?id=7crCUS_F2ocC.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014. ISBN 1107057132, 9781107057135.

Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Stochastic convex optimization. In *COLT 2009 - The 22nd Conference on Learning Theory, Montreal, Quebec, Canada, June 18-21, 2009*, 2009. URL http://www.cs.mcgill.ca/%7Ecolt2009/papers/018.pdf#page=1.

Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/310ce61c90f3a46e340ee8257bc70e93-Paper.pdf.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.