A Study of Distribution Free Non-parametric Regression

Ayush Chaurasia 2016MT10617
Kumar Prithvi Mishra 2016MT10618
Prof. Sivananthan Sampath Supervisor

B.Tech. Project

Abstract

The paper first talks about "Why is non-parametric regression important?" and shows the mathematical formulation of parametric methods and then the need of non-parametric estimation. Then we dive into one specific type of non-parametric method specifically deep CNNs, how are they formulated and see how well do they approximate a function and other nuances related to it. Finally at the end we apply the theory to a medical problem and present how the method can be applied to it and analyze the results.

Contents

1 Why is Non-Parametric Regression Important?

1.1 Regression Analysis and L_2 Risk

In regression analysis one considers a random vector (X, Y), where X is $\mathbb{R}^d$-valued and Y is $\mathbb{R}$-valued, and one is interested how the value of variable Y depends on the value of vector X. This means that one wants to find a (measurable) function $f : \mathbb{R}^d \to \mathbb{R}$, such that $f(X)$ should be a close approximation to Y in some sense, i.e. making $|f(X) - Y|$ "small". We can resolve this problem by introducing L_2 *risk or mean squared error of* f,

$$\mathbb{E}|f(X) - Y|^2,$$

and requiring it to be as small as possible.

There are two reasons for considering the L_2 risk. First, this simplifies the mathematical treatment of the whole problem (the function which minimizes the L_2 risk can be derived explicitly). Second, trying to minimize the L_2 risk leads naturally to estimates which can be computed rapidly.

So we are interested in a (measurable) function $m^* : \mathbb{R}^d \to \mathbb{R}$ such that

$$\mathbb{E}|m^*(X) - Y|^2 = \min_{f:\mathbb{R}^d \to \mathbb{R}} \mathbb{E}|f(X) - Y|^2.$$

Such a function can be obtained explicitly as follows. Let

$$m(x) = \mathbb{E}\{Y|X = x\}$$

be the *regression function*. We will show that the regression function minimizes the L_2 risk. Indeed, for an arbitrary $f : \mathbb{R}^d \to \mathbb{R}$, one has

$$\mathbb{E}|f(X) - Y|^2 = \mathbb{E}|f(X) - m(X) + m(X) - Y|^2$$
$$\mathbb{E}|f(X) - Y|^2 = \mathbb{E}|f(X) - m(X)|^2 + \mathbb{E}|m(X) - Y|^2,$$

where we have used

$$\begin{aligned}
&\mathbb{E}\{(f(X) - m(X))(m(X) - Y)\} \\
&= \mathbb{E}\{\mathbb{E}\{(f(X) - m(X))(m(X) - Y)|X\}\} && \because \mathbb{E}\{\mathbb{E}\{\mathbb{Y}|\mathbb{X}\}\} = \mathbb{E}\{\mathbb{Y}\} \\
&\quad = \mathbb{E}\{(f(X){-}m(X))\mathbb{E}\{(m(X){-}Y)|X\}\} && \because \mathbb{E}\{m(X)|X\} = \mathbb{E}\{m(X)\} \\
&\quad = \mathbb{E}\{(f(X) - m(X))(m(X) - m(X))\} \\
&= 0.
\end{aligned}$$

Hence,

$$\mathbb{E}|f(X) - Y|^2 = \int_{\mathbb{R}^d} |f(x) - m(x)|^2 \mu(dx) + \mathbb{E}|m(X) - Y|^2 \tag{1}$$

The second term is fixed, so it doesn't play any role in optimizing L_2 risk. Hence, we only reduce the first term, which is the L_2 error. From above, we can see that $f(x) = \mathbb{E}\{Y|X = x\}$ is the optimal minimizer for the L_2 risk, but often we don't have the distribution of $f_{y/x}(y/x)$. Hence, we now try to estimate the distribution in the next section.

1.2 Regression Function Estimation and L_2 Error

Let $\mathbb{D}_n$ be the set of data defined by $\mathbb{D}_n = (X_1, Y_1), ..., (X_n, Y_n)$. In the regression function estimation problem one wants to use the data $\mathbb{D}_n$ in order to construct an estimate $m_n : \mathbb{R}^d \to \mathbb{R}$ of the regression function m. In general, the L_p error, $\int_{\mathbb{C}} |m_n(x) - m(x)|^p dx$, where the integration is with respect to the Lebesgue measure, $\mathbb{C}$ is a fixed subset of $\mathbb{R}^d$, and $p \geqslant 1$ is arbitrary (often $p = 2$ is used).
Recall that the main goal was to find a function f such that the L_2 risk $\mathbb{E}|f(X) - Y|^2$ is small. The minimal value of this L_2 risk is $\mathbb{E}|m(X) - Y|^2$, and it is achieved by the regression function m. Similarly to eq. (1), L_2 risk $\mathbb{E}|m_n(X) - Y|^2|D_n$ of an estimate m_n satisfies

$$\mathbb{E}\{|m_n(X) - Y|^2|D_n\} = \int_{\mathbb{R}^d} |m_n(x) - m(x)|^2 \mu(dx) + \mathbb{E}|m(X) - Y|^2. \tag{2}$$

Thus the L_2 risk of an estimate m_n is close to the optimal value if and only if the L_2 error

$$\int_{\mathbb{R}^d} |m_n(x) - m(x)|^2 \mu(dx) \tag{3}$$

is close to zero. Therefore we will use the L_2 error (3) in order to measure the quality of an estimate and we will study estimates for which this L_2 error is small.

1.3 Application to Pattern Recognition

Here we try to model pattern recognition problem also as a regression problem. In pattern recognition, Y takes only finitely many values. For simplicity assume that Y takes two values, say 0 and 1. The goal is to find a function $g^* : \mathbb{R}^d \to \{0, 1\}$ which minimizes the probability of $g^*(X) \neq Y$,i.e., to find a function g^* such that

$$\mathbb{P}\{g^*(X) \neq Y\} = \min_{g:\mathbb{R}^d \to \{0,1\}} \mathbb{P}\{g(x) \neq Y\}, \tag{4}$$

where g^* is called the Bayes decision function, and $\mathbb{P}\{g(X) \neq Y\}$ is the probability of misclassification.

1.3.1 Lemma

$$g^*(x) = \begin{cases} 1 & \text{if } \mathbb{P}\{Y = 1|X = x\} \geq 1/2, \\ 0 & \text{if } \mathbb{P}\{Y = 1|X = x\} < 1/2 \end{cases} \tag{5}$$

is the Bayes decision function, i.e., g^* satisfies (4).
PROOF. Let $g : \mathbb{R}^d \to \{0, 1\}$ be an arbitrary (measurable) function.
$\mathbb{P}\{g(x) \neq Y|X = x\} - \mathbb{P}\{g^*(X) \neq Y|X = x\}$
$= \mathbb{P}\{Y = g^*(x)|X = x\} - \mathbb{P}\{Y = g(X)|X = x\} \geq 0$
because

$$\mathbb{P}\{Y = g^*|X = x\} = \max\{\mathbb{P}\{Y = 0|X = x\}, \mathbb{P}\{Y = 1|X = x\}\}$$

by the definition of g^*. This proves

$$\mathbb{P}\{g^*(X) \neq Y|X = x\} \leq \mathbb{P}\{g(X) \neq Y|X = x\}$$

for all $x \in \mathbb{R}^d$, which implies

$$\begin{aligned}\mathbb{P}\{g^*(X) \neq Y\} &= \int \mathbb{P}\{g^*(X) \neq Y|X = x\}\mu(dx) \\ &\leq \int \mathbb{P}\{g(X) \neq Y|X = x\}\mu(dx) \\ &= \mathbb{P}\{g(X) \neq Y\}\end{aligned}$$

$\mathbb{P}\{Y = 1|X = x\}$ and $\mathbb{P}\{Y = 0|X = x\}$ are the so-called a posterior probabilities. Observe that $\mathbb{P}\{Y = 1|X = x\} = \mathbb{E}\{Y|X = x\} = m(X)$ A natural approach is to estimate the regression function m by an estimate m_n using data D_n and then to use a so-called plug-in estimate

$$g_n^*(x) = \begin{cases} 1 & \text{if } m_n \geq 1/2, \\ 0 & \text{if } m_n < 1/2 \end{cases} \tag{6}$$

The next theorem implies that if m_n is close to the real regression function m, then the error probability of decision g_n is near to the error probability of the optimal decision g^*.

1.3.2 Theorem

Let $\hat{m} : \mathbb{R}^d \to \mathbb{R}$ be a fixed function and define the plug-in decision $\hat{g}$ by

$$\hat{g}^*(x) = \begin{cases} 1 & \text{if } \hat{m} \geq 1/2, \\ 0 & \text{if } \hat{m} < 1/2 \end{cases}$$

Then

$$\begin{aligned}0 &\leq \mathbb{P}\{\hat{g}(X) \neq Y\} - \mathbb{P}\{g^*(X) \neq Y\} \\ &\leq 2\int_{\mathbb{R}^d} |\hat{m} - m(x)|\mu(dx) \\ &\leq 2\left(\int_{\mathbb{R}^d} |\hat{m} - m(x)|^2\mu(dx)\right)^{1/2}\end{aligned}$$

Note that the second inequality in particular is not tight. Therefore pattern recognition is easier than regression estimation. It follows from Theorem, that the error probability of the plug-in decision g_n defined above satisfies

$$\begin{aligned}0 &\leq \mathbb{P}\{g_n(X) \neq Y|D_n\} - \mathbb{P}\{g^*(X) \neq Y\} \\ &\leq 2\int_{\mathbb{R}^d} |m_n(x) - m(x)|\mu(dx) \\ &\leq 2\left(\int_{\mathbb{R}^d} |m_n(x) - m(x)|^2\mu(dx)\right)^{1/2}.\end{aligned}$$

Thus estimates m_n with small L_2 error automatically lead to estimates g_n with small misclassification probability. Observe however, that for (6) to be a good approximation of (5) it is only important that $m_n(x)$ should be on the same side of the decision boundary (1/2 in this case) as $m(x)$. Nevertheless, one often constructs estimates by minimizing the L_2 risk $\mathbb{E}\{|m_n(X)-Y|^2|D_n\}$ and using the plug-in rule (6), because trying to minimize the L_2 risk leads to estimates which can be computed efficiently. This can be generalized to the case where Y takes $M \geq 2$ distinct values, without loss of generality $1,..,M$. The goal is to find a function $g^* : \mathbb{R}^d \to \{1,...,M\}$ such that

$$\mathbb{P}\{g^*(X) \neq Y\} = \min_{g:\mathbb{R}^d \to \{1,...,M\}} \mathbb{P}\{g(X) \neq Y\}, \tag{7}$$

where $g^*(x)$ is called the Bayes decision function. It can be computed using the posterior probabilities $\mathbb{P}\{Y=k|X=x\}(k \in \{1,...,M\})$:

$$g^*(x) = arg \max_{1 \leq k \leq M} \mathbb{P}\{Y=k|X=x\} \tag{8}$$

The a posteriori probabilities are the regression functions

$$\mathbb{P}\{Y=k|X=x\} = \mathbb{E}\{I_{\{Y=k\}}|X=x\} = m^{(k)}(x).$$

Given data D_n, estimates $m_n^{(k)}$ of $m^{(k)}$ can be constructed from the data set

$$D_n^{(k)} = \{(X_1, I_{Y_1=k}),...,(X_n, I_{Y_n=k})\},$$

and one can use a plug-in estimate

$$g_n(x) = arg \max_{1 \geq k \geq M} m_n^{(k)}(x) \tag{9}$$

to estimate g^*. If the estimates $m_n^{(k)}$ are close to the posterior probabilities, then again the error of the plug-in estimate (9) is close to the optimal error.

1.4 Curse of Dimensionality

Let (X,Y) be the data points. If X takes values in a high-dimensional space, estimating the regression function is difficult because in general, it is not possible to densely pack the space of X with finitely many sample points, even if the sample size is very large.
Let $X_1, X_2, ..., X_n$ be the n i.i.d. $\mathbb{R}^d$-valued random variables uniformly distributed in the hypercube $[0,1]^d$.

$$d_\infty(d,n) = \mathbb{E}\left\{\min_{i=1,...,n} ||X - X_i||_\infty\right\}.$$

where, $d_\infty(d,n)$ is the expected supremum-norm distance of X to its nearest neighbor, $||x||_\infty$ is the supremum norm $||x||_\infty = \max_{l=1,...,d} |x^{(l)}|$.

Then

$$d_\infty(d,n) = \int_0^\infty \mathbb{P}\left\{\min_{i=1,\dots,n} ||X - X_i||_\infty > t\right\} dt$$

$$= \int_0^\infty \left(1 - \mathbb{P}\left\{\min_{i=1,\dots,n} ||X - X_i||_\infty \le t\right\}\right) dt$$

The bound

$$\mathbb{P}\left\{\min_{i=1,\dots,n} ||X - X_i||_\infty \le t\right\} \le n.\mathbb{P}\{||X - X_1||_\infty \le t\}$$

$$\le n.(2t)^d$$

implies

$$d_\infty(d,n) \ge \int_0^{1/(2n^{1/d})} (1 - n.(2t)^d) dt$$

$$= \frac{d}{2(d+1)} . \frac{1}{n^{1/d}}.$$

	$n = 100$	$n = 1000$	$n = 10,000$	$n = 100,000$
$d_\infty(1,n)$	≥ 0.0025	≥ 0.00025	≥ 0.000025	≥ 0.0000025
$d_\infty(10,n)$	≥ 0.28	≥ 0.22	≥ 0.18	≥ 0.14
$d_\infty(20,n)$	≥ 0.37	≥ 0.34	≥ 0.30	≥ 0.26

1.5 Bias-Variance Tradeoff

Let m_n be an arbitrary estimate. For any $x \in \mathbb{R}^d$ we can write the expected squared error of m_n at x as

$$\mathbb{E}\{|m_n(x) - m(x)|^2\} = \mathbb{E}\{|m_n(x) - \mathbb{E}\{m_n(x)\}|^2\} + |\mathbb{E}\{m_n(x)\} - m(x)|^2$$

$$= \mathbf{Var}(m_n(x)) + |bias(m_n(x))|^2.$$

Here $\mathbf{Var}(m_n(x))$ is the variance of the random variable $m_n(x)$ and $bias(m_n(x))$ is the difference between the expectation of $m_n(x)$ and $m(x)$. This also leads to a similar decomposition of the expected L_2 error:

$$\mathbb{E}\left\{\int |m_n(x) - m(x)|^2 \mu(dx)\right\} = \int \mathbb{E}|m_n(x) - m(x)|^2 \mu(dx)$$

$$= \int \mathbf{Var}(m_n(x))\mu(dx) + \int |bias(m_n(x))|^2 \mu(dx).$$

2 Universal Approximation Theorem

In this section, we first prove the universal approximation theorem, which in essence shows that a shallow neural network can approximate any continuous function.

2.1 Notations and Concepts

2.1.1 Shallow Network

A shallow network is effectively a neural network with only one hidden layer. In general, it is a finite linear combination of compositions of a fixed univariate function and a set of affine functionals. It can be represented by

$$G(x) = \Sigma_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

2.1.2 Univariate Function σ

Here σ is considered a general univariate function so called sigmoidal function.

$$\sigma(t) \to \begin{cases} 1 & \text{as } t \to +\infty \\ 0 & \text{as } t \to -\infty \end{cases}$$

Note that the sigmoidal function here need not be a monotonous function.

2.1.3 Discriminatory Function

σ is discriminatory if for a measure $\mu \in M(I_n)$

$$\int_{I_n} \sigma(y^T + \theta) d\mu(x) = 0$$

for all $y \in \mathbb{R}$ implies that $\mu = 0$

2.2 Proof of Universal Approximation Theorem

Let σ be the continuous discriminatory function. Then finite sums of the form

$$G(x) = \Sigma_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

are dense in $C(I_n)$ where $C(I_n)$ is the set of continuous functions over I_n and I_n is the unit hypercube of n dimension. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \epsilon \;\; \forall\, x \in I_n.$$

PROOF. Let $S \subset C(I_n)$ be the set of functions of the form $G(x)$. Clearly S is a linear subspace of $C(I_n)$. We claim that the closure of S is all of $C(I_n)$.
Assume that the closure of S is not all of $C(I_n)$. Then the closure of S, say R, is a closed proper subspace of $C(I_n)$. By the Hahn-Banach theorem, there is a bounded linear function on $C(I_n)$, call it L, with the property that $L \neq 0$ but $L(R) = L(S) = 0$.

By the Riesz Representation Theorem, this bounded linear functional, L, is of the form

$$L(h) = \int_{I_n} h(x) d\mu(x)$$

for some $\mu \in M(I_n), \forall\ h \in C(I_n)$. In particular, since $\sigma(y^T x + \theta)$ is in $\mathbb{R}$ for all y and θ, we must have that

$$\int_{I_n} \sigma(y^T x + \theta) d\mu(x) = 0$$

for all y and θ.
However, we assumed that σ was discriminatory so that this condition implies that $\mu = 0$ contradicting our assumption. Hence, the subspace S must be dense in $C(I_n)$.

3 No Free Lunch

3.1 PAC Learnability

In our learning problems, data points come from some domain set $\mathcal{X}$, have labels belonging to some set $\mathcal{Y}$, and can be sampled i.i.d. from a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. We are interested in finding some hypothesis $h \in H$ that explains an underlying concept $f : \mathcal{X} \to \mathcal{Y}$. We will measure the effectiveness of our hypothesis through a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. The error of our hypothesis h is $err(h) = \mathbb{E}_{(x,y)\ D}[l(h(x), y)]$. For the time being, we will make the *realizability assumption*, i.e, that $\exists$ some hypothesis $h^* \in \mathcal{H}$ such that $err(h^*) = 0$. In words, this assumption means that there is a hypothesis $h \in \mathcal{H}$ that perfectly explains the data.

3.1.1 Definition

A learning problem $(\mathcal{X}, \mathcal{Y}, l)$ is PAC learnable with respect to the hypothesis class $\mathcal{H}$ under the realizability assumption (concept $f \in \mathcal{H}$), if $\exists$ a function $m : [0,1]^2 \to \mathbb{R}$ and an algorithm with the following property: For every $\epsilon, \delta > 0$, for every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, running the algorithm on $m(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis $h \in \mathcal{H}$ such that, with probability at least $1 - \delta$, $err(h) \leq \epsilon$.

3.2 No Free Lunch Theorem

The following theorem shows that PAC-learning is impossible without restricting the hypothesis class $\mathcal{H}$.

3.2.1 Theorem

Consider any $m \in \mathbb{N}$, any domain $\mathcal{X}$ of size $|\mathcal{X}| = 2m$, and any algorithm A which outputs a hypothesis $h \in \mathcal{H}$ given a sample S. Then $\exists$ a concept $f : \mathcal{X} \to \{0, 1\}$ and a distribution $\mathcal{D}$ such that $err(f) = 0$ and $err(A(S)) \geq \frac{1}{10}$ with probability at least $\frac{1}{10}$.
PROOF. We will show that for any learner, there is some learning task that it will not learn

well. Formally, take $\mathcal{D}$ to be the uniform distribution over $\{(x, f(x))\}$. Our proof strategy will be to show the following inequality

$$Q = \mathbb{E}_{f:X\to\{0,1\}}[\mathbb{E}_{S\ \mathcal{D}^m}[err(A(S))]] \geq \frac{1}{4}$$

as an intermediate step, and then use Markov's Inequality to conclude. We proceed by invoking Fubini's theorem (to swap the order of expectations) and then conditioning on the event $x \in S$.

$$Q = \mathbb{E}_S[\mathbb{E}_f[\mathbb{E}_{x\in\mathcal{X}}[A(S)(x) \neq f(x)]]]$$

$$= \mathbb{E}_{S,x}[\mathbb{E}_f[A(S)(x) \neq f(x)|x \in S]]\mathbb{P}(x \in S) + \mathbb{E}_{S,x}[\mathbb{E}_f[A(S)(x) \neq f(x)|x \notin S]]\mathbb{P}(x \notin S)$$

The first term is, in the worst case, at least 0. Also note that $\mathbb{P}(x \notin S \geq \frac{1}{2}$. Finally, observe that $\mathbb{P}(A(S)(x) \neq f(x)) = \frac{1}{2}$ for all $x \notin S$ since we are given that the "true" concept is chosen uniformly random. Hence, we get that:

$$Q \geq 0 + \frac{1}{2}.\frac{1}{2} = \frac{1}{4},$$

which is the intermediate step we wanted to show. Then by a simple application of the reverse Markov Inequality:

$$\mathbb{P}(Q \geq \frac{1}{10}) \geq \frac{\frac{1}{4} - \frac{1}{10}}{1 - \frac{1}{10}} \geq \frac{1}{10}$$

4 Universality of Deep Convolutional Neural Networks

In particular, for deep CNNs having convolutional structures without fully connected layers, it is unknown which kinds of functions can be approximated. The paper clarifies the difference between deep CNNs and neural network in general, defines the CNN and provides a rigorous mathematical theory to see which kind of functions can be approximated and with what error rate.

4.1 Notations and Concepts

The deep CNNs considered here have a few essential ingredients:

4.1.1 Activation function

A rectified linear unit (ReLU) defined as a uni-variate nonlinear function σ given by

$$\sigma(u) = (u)_+ = max\{u, 0\}, \quad u \in \mathbb{R}$$

4.1.2 Convolutional Filter Masks

A sequence of convolutional filter masks $\mathbf{w} = \{w^{(j)}\}_j$ induce a sparse convolutional structure. A filter mask $w = (w_k)_{k=-\infty}^{\infty}$ means a sequence of filter coefficients. A fixed integer filter length $s \geqslant 2$ is used to decide non-zero weights namely, $w_k^{(j)} \neq 0$ only for $0 \leq k \leq s$ and hence controls the sparsity. The convolution of such a filter mask w with another sequence $v = (v_0, ..., v_D)$ is a sequence $w \star v$ given by $(w \star v)_i = \sigma_{k=0}^{D} w_{i-k} v_k$. This convolution leads to a $(D+s) \times D$ convolutional matrix T which has constant valued diagonals.

$$T = \begin{bmatrix} w_0 & 0 & 0 & 0 & \cdot & 0 \\ w_1 & w_0 & 0 & 0 & \cdot & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ w_s & w_{s-1} & \cdots & w_0 & 0\cdots & 0 \\ 0 & w_s & \cdots & w_1 & w_0\cdots & 0 \\ \vdots & \ddots & \ddots\ddots & \ddots & \ddots\ddots & \vdots \\ \cdots & \cdots & 0 & w_s & \cdots & w_0 \\ \cdots & \cdots & \cdots & 0 & w_s\cdots & w_1 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots\cdots & \cdots 0 & w_s & w_{s-1} \\ 0 & \cdots & \cdots & \cdots & \cdots 0 & w_s \end{bmatrix}$$

4.1.3 Convolutional Neural Networks

A deep CNN is a sequence of J vectors $h^{(j)}(x)$ of functions on $\mathbb{R}^d$ with $h^{(0)}(x) = x$ given iteratively by:

$$h^{(j)}(x) = \sigma(T^j h^{j-1}(x) - b^{(j)}), \quad j = 1, 2, ..., J$$

where $T^{(j)} = (w_{i-k}^{(j)})$ is a $d_j \times d_{j-1}$ convolutional matrix, σ acts on vectors component-wise, and $\boldsymbol{b}$ is a sequence of bias vectors $b^{(j)}$ where $b^{(j)}$ for $j = 1, 2, ..., J-1$ is of the form:

$$\begin{bmatrix} \mathrm{b}_1 & ...b_s & b_{s+1} & ...bs+1 & b_{d_{j-s+1}} & ...b_{d_j} \end{bmatrix}$$

i.e. $d_j - 2s$ repeated components in the middle and s components on each side. The last layer bias vector $b^{(J)}$ is of the form:

$$\begin{bmatrix} \mathrm{b}_1 & b_2 & ...b_{d_J - 1} & b_{d_J} \end{bmatrix}$$

4.1.4 Number of parameters in CNN

For last layer, d_J parameters for bias, d_J parameters for $c \in \mathbb{R}^{d_J}$ (linear combination of the outputs of last layer) and $(s+1)$ parameters for w^J. For layers apart from last layer, $2s+1$ parameters for $b^{(j)}$ and $(s+1)$ parameters for w^J. So the total number of free parameters in the deep CNN is $(3s+2)J + 2d_J - 2s - 1$, much smaller than that in a classical fully connected multi-layer neural network with full connection matrices $T^{(j)}$ involving $d_j d_{j-1}$ free parameters. It demonstrates the computational efficiency of deep CNNs.

4.2 Hypothesis Space

The hypothesis space of a learning algorithm is the set of all possible functions that can be represented or produced by the algorithm. For the deep CNN of depth J considered here, the hypothesis space is a set of functions defined by

$$\mathcal{H}_J^{w,b} = \left\{ \Sigma_{k=1}^{d_j} c_k h_k^J(x) : c \in \mathbb{R}^{d_J} \right\}$$

The approximation ability of the hypothesis space depend completely on the sequence of convolutional filter masks $\boldsymbol{w} = \{w^{(j)}\}_{j=1}^J$ and the sequence of bias vectors $\boldsymbol{b} = \{b^{(j)}\}_{j=1}^J$ which have been described above.

4.3 Theorem A

Let $2 \leq s \leq d$. For any compact subset Ω of $\mathbb{R}^d$ and any $f \in C(\Omega)$, $\exists$ a sequence $\boldsymbol{w}$ of filter masks, $\boldsymbol{b}$ of bias vectors and $f_J^{w,b} \in \mathcal{H}_J^{w,b}$ such that

$$\lim_{J \to \infty} \|(f - f_J^{w,b})\|_{C(\Omega)} = 0$$

where $C(\Omega)$ is the space of continuous functions on Ω with norm $\|f\|_{C(\Omega)} = sup_{x \in \Omega} |f(x)|$.

4.3.1 Essence of Theorem A

The result in theorem A verifies the result of universality of deep CNNs, asserting that any $f \in C(\Omega)$, can be approximated by $\mathcal{H}_J^{w,b}$ to an arbitrary accuracy when the depth J is large enough.

4.4 Theorem B

Let $2 \leq s \leq d$ and $\Omega \subseteq [-1,1]^d$. If $J \geqslant 2d/(s-1)$ and $f = F|_\Omega$ with $F \in H^r(\mathbb{R}^d)$ and an integer index $r > 2 + d/2$, then $\exists \boldsymbol{w}, \boldsymbol{b}$ and $f_J^{w,b} \in \mathcal{H}_J^{w,b}$ such that

$$\|(f - f_J^{w,b})\|_{C(\Omega)} \leqslant c\|F\| \sqrt{logJ} (1/J)^{1/2 + 1/d},$$

where c is an absolute constant and $\|F\|$ denotes the Sobolev norm of $F \in \mathcal{H}^r(\mathbb{R}^d)$.

4.4.1 Essence of Theorem B

Theorem B mainly presents rates of approximation by deep CNNs for functions in the Sobolev space $H^r(\Omega)$ with an integer index $r > 2 + d/2$. In this case the set $H^r(\Omega)$ is dense in $C(\Omega)$, so Theorem A follows from Theorem B by scaling.

4.4.2 Approximation ability of deep CNNs numerically

In theorem B if we take $s = \lceil 1 + d^{\tau}/2 \rceil$ and $J = \lceil 4d^{1-\tau} \rceil L$ with $0 \leqslant \tau \leqslant 1$ and $L \in \mathbb{N}$, where $\lceil u \rceil$ denotes the smallest integer not smaller than u, then we have

$$\|(f - f_J^{w,b})\|_{C(\Omega)} \leqslant c\|F\|\sqrt{\frac{(1-\tau)\log d + \log L + \log 5}{4d^(1-\tau)L}},$$

since $\lceil 4 \rceil = 5$, $\frac{1}{\lceil 4d^{1-\tau} \rceil^{1/d}} \leqslant 1$. Also, width of CNN is bounded by $12Ld$ and total number of free parameters $(5s+2)J + 2d - 2s - 1 \leqslant (73L+2)d$. If we take $L = 1$ and $\tau = 1/2$ to get a bound for the relative error

$$\frac{\|(f - f_J^{w,b})\|_{C(\Omega)}}{\|F\|} \leqslant \frac{c}{2} d^{-\frac{1}{4}} \sqrt{\log(5\sqrt{d})}$$

where depth of CNN $L = 1$ ad $\tau = 1/2$ is $J = \lceil 4d^{1-\tau} \rceil L = \left\lceil 4\sqrt{d} \right\rceil$ and at most $75d$ parameters. This explains the strong approximation of CNN and it's error bounded by depth of CNN.

5 Why and When Can Deep-but Not Shallow-networks Avoid the Curse of Dimensionality

5.1 Introduction

The main message in the paper is that deep networks have the theoretical guarantee, which shallow networks do not have, that they can avoid the curse of dimensionality for an important class of problems, corresponding to compositional functions, i.e., functions of functions. An especially interesting subset of such compositional functions are hierarchically local compositional functions where all the constituent functions are local in the sense of bounded small dimensionality.

The problems for which certain deep networks are guaranteed to avoid the curse of dimensionality correspond to input-output mappings that are compositional. The most interesting set of problems consists of compositional functions composed of a hierarchy of constituent functions that are local: An example is

$$f(x_1, ..., x_8) = h_3(h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8))).$$

The compositional function f requires only "local" computations (here with just dimension 2) in each of its constituent functions h.

5.2 Theorems

Let $I_n = [1,1]^n, \mathbb{X} = \mathcal{C}(I_n)$ be the space of all continuous functions on I_n, with $f = \max_{x \in I_n} |f(x)|$. Let $S_{N,n}$ denote the class of all shallow networks with N units of the form

$$x \rightarrow \Sigma_{k=1}^{N} a_k \sigma(\langle w_k, x \rangle + b_k)$$

where $w_k \in \mathbb{R}^n, b_k, a_k \in \mathbb{R}$. The number of trainable parameters here is $(n+2)N$. Let $m \geq 1$ be an integer, and W_m^n be the set of all functions of n variables with continuous partial derivatives of orders up to $m < \infty$ such that $||f|| + \Sigma_{1 \leq |k|_1 \leq m} ||\mathcal{D}^k f|| \leq 1$, where $\mathcal{D}^k$ denotes the partial derivative indicated by the multi-integer $k \geq 1$, and $|k|_1$ is the sum of the components of k. For the hierarchical binary tree network, the analogous spaces are defined by considering the compact set $W_m^{n,2}$ to be the class of all compositional functions f of n variables with a binary tree architecture and constituent functions h in W_m^2. We define the corresponding class of deep networks $\mathcal{D}_{N,2}$ to be the set of all deep networks with a binary tree architecture, where each of the constituent nodes is in $S_{M,2}$, where $N = |V|M$, V is the set of non–leaf vertices of the tree. We note that in the case when n is an integer power of 2, the total number of parameters involved in a deep network in $D_{N,2}$, i.e., weights and biases, is $4N$.

5.2.1 Theorem 1

Let $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ be infinitely differentiable, and not a polynomial. For $f \rightarrow W_m^n$, the complexity of shallow networks that provide accuracy at least ϵ is

$$\mathcal{N} = \mathcal{O}(\epsilon^{\frac{-n}{m}})$$

and is the best possible.

5.2.2 Theorem 2

For $f \in W_m^{n,2}$, consider a deep network with the same compositional architecture and with an activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ which is infinitely differentiable, and not a polynomial. The complexity of the network to provide approximation with accuracy at least is

$$\mathcal{N} = \mathcal{O}((n-1)\epsilon^{\frac{-2}{m}}).$$

6 Medical Problem Analysis: COVID-19

6.1 Introduction

In this paper we use a meta-learning approach to choosing the kernels and regularization parameters in regularized kernel based learning algorithms. The concept of meta-learning presupposes that the above-mentioned components of the algorithms are selected on the base of previous experience with similar learning tasks. Therefore, selection rules developed in this way are intrinsically problem-oriented. Moreover, meta-learning is very much dependent on the quality of data extracted from previous experience.

For the medical problem, we have taken the COVID-19 problem. We look to predict the number of cases in future using the past data of number of confirmed cases due to the corona virus.

Despite the naturalness of this approach, the idea of kernel based meta-learning algorithm hasn't been applied to the COVID-19 problem yet. Most of the current solutions use different models like ARIMA (time series model), logistic curve fit (regression models), RNNs (time series models) or using Prophet (model developed by Facebook research).

In this paper we demonstrate the meta-learning approach to show the power and versatility of such kernel based learning methods and also produce comparable results to state of the art models, hence is the novelty of our study.

6.2 Aim

We use a kernel-based regularization learning algorithm in which the kernel and the regularization parameter are adaptively chosen on the base of previous experience with similar learning tasks to predict the future number of cases due to the corona virus and produce comparable results to the state of the art models like ARIMA (Auto-Regressive Integrated Moving Average) model.

6.3 Motivation

Our motivation comes from the paper *"A meta-learning approach to the regularized learning-Case study: Blood glucose prediction"* [5]. In this paper the authors have analyzed and predicted blood glucose sugar levels of human beings using the similar kernel based meta-learning approach. We intend to use this similar proven approach.

6.4 Theory

6.4.1 Regression algorithm

To fit the "training" set of data $S_m = (x_i, y_i)_{i=1}^m$ with a function $f : X \to Y$ (where X is a closed subset of $\mathbb{R}^n$) and $Y \subset \mathbb{R}$ that generalizes i.e. is predictive, we use the following key algorithm [9]:

1. Start with data $(x_i, y_i)_{i=1}^m$.

2. Choose a symmetric, positive-definite function $K_x(x^{'}) = K(x, x^{'})$, continuous on $X \times X$.

3. Define $f : X \to Y$ by

$$f(x) = \sum_{i=1}^{m} c_i K_{x_i}(x)$$

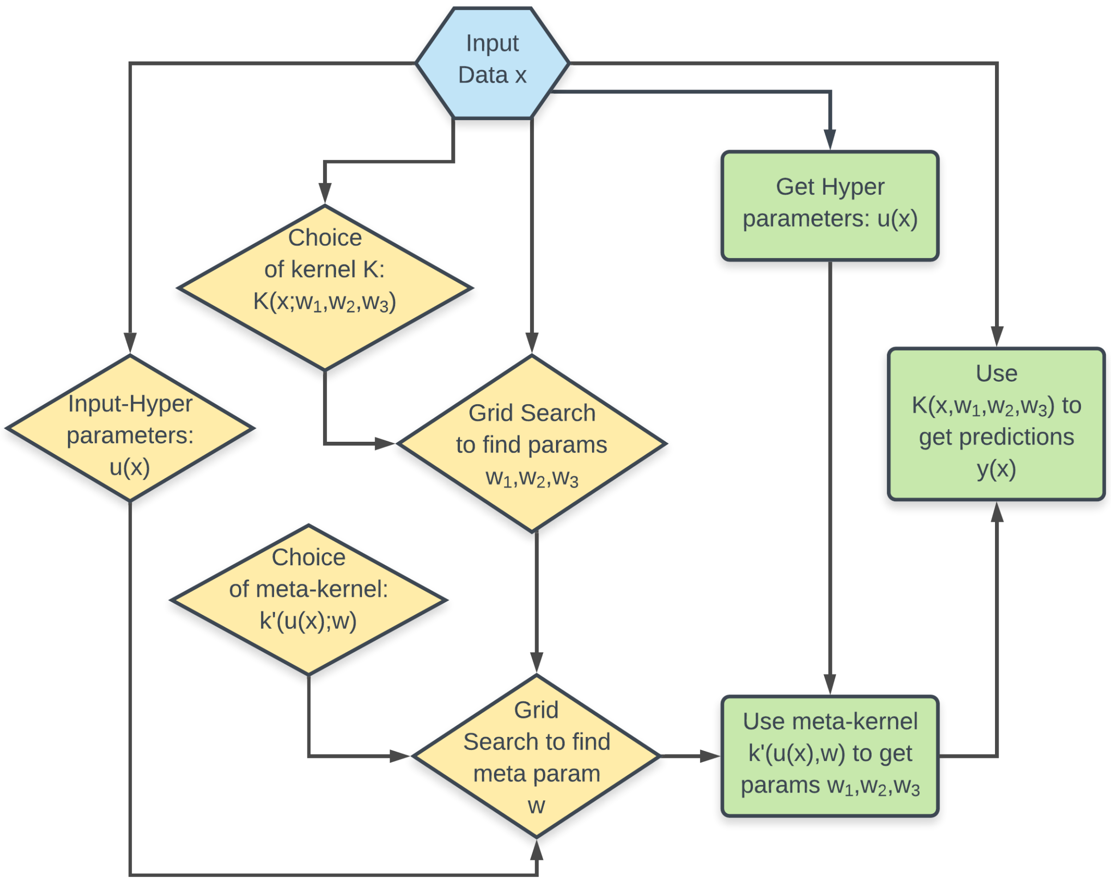

Input Data x
Choice of kernel K: K(x;w1,w2,w3)
Get Hyper parameters: u(x)
Input-Hyper parameters: u(x)
Grid Search to find params w1,w2,w3
Use K(x,w1,w2,w3) to get predictions y(x)
Choice of meta-kernel: k'(u(x);w)
Grid Search to find meta param w
Use meta-kernel k'(u(x),w) to get params w1,w2,w3

Figure 1: *The figure shows complete flow of model. The yellow boxes are the steps used for training of model and the green boxes are the steps used for prediction by model. The blue box at the top denotes the input.*

where $\boldsymbol{c} = (c_1, ..., c_m)$ and

$$(m\gamma \boldsymbol{I} + \boldsymbol{K})\boldsymbol{c} = \boldsymbol{y}$$

where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{K}$ is the square positive-definite matrix with elements $K_{i,j} = K(x_i, x_j)$, and $\boldsymbol{y}$ is the vector with coordinates y_i. The parameter γ is a positive, real number.

6.4.2 Meta-learning

Meta learning is concerned with learning about one's own learning and learning processes. The problem with the above regularization scheme is with the choice of kernel K. Several approaches to address this issue choose a kernel K globally for the whole training set $S_m = (x_i, y_i)_{i=1}^m$ but they do not account for particular input x_i. As a result, if some new input–output pair (x_μ, y_μ) is added to the training set S_m, then, in accordance with the known approaches, a kernel selection procedure should be started from scratch, which is rather costly. This is where meta-learning succeeds in adaptively choosing the kernel for every given input.

6.5 Experiment

6.5.1 Dataset

The dataset [11] contains the number of confirmed cases with COVID-19 recorded everyday for a span of 102 days (till 2^{nd} May 2020) for several countries. For training data ($\mathcal{D}$), 30 input vectors each containing consecutive 7 days of data (number of confirmed cases) have been taken. These 30 vectors have been only taken from 2 countries New York, USA and Spain.

$$TrainingData : \mathcal{D} \in \mathbb{R}^{30\times 7}$$

$$InputVector : \mathcal{D}_i = x \in \mathbb{R}^7, i = \{1, 2, ..., 30\}$$

where x (or $\mathcal{D}_i$) is number of confirmed cases for consecutive 7 days. For testing purposes similar vectors of dimension 7 have been taken randomly from different countries.

6.5.2 Model

In figure 1, we can see the flow of model. First we will go through the yellow boxes which explain the training process and then the green boxes explaining prediction process.

6.5.3 Model Training

In figure 1, we start with the 30 vectors($\mathcal{D}$), and take a input data vector x($x \in \mathcal{D}$), as described in the dataset section. Our first step is to choose a suitable base kernel. Based on the type of data we had to predict here we have taken a sum of polynomial and gaussian kernel,

$$K(x_1, x_2; w_1, w_2, w_3) = (x_1x_2)^{w_1} + w_2e^{-w_3\|x_1-x_2\|^2}.$$

where w_1, w_2, w_3 are the parameters.

Now the next step is to find the parameters w_1, w_2, w_3 using grid search for each of the vector $x \in \mathcal{D}$. After this step we are ready for meta-training part i.e. training for the meta kernel.

Again we need to choose a suitable kernel for the meta kernel, since we had to just learn parameters in this case and not any trend (which we can plot and analyse) we choose a gaussian kernel,

$$k(x_1, x_2; w) = e^{-w\|x_1-x_2\|^2}.$$

Now for meta learning 3 inputs are taken,

1. The hyper parameters (or the meta-features) $u(x)$ where $u(x) \in \mathbb{R}^2$, namely = $u(x) = (u_1(x), u_2(x))$, the coefficients of "least squares regression line" $x^{lin} = u_1(x)i + u_2(x)$, that produces the best linear fit linking the components $i \in \{1, 2, ..., 7\}$ and $x \in \{x_1, x_2, .., x_7\}$.

2. Parameters w_1, w_2, w_3 for the base kernel found by grid search in previous steps.

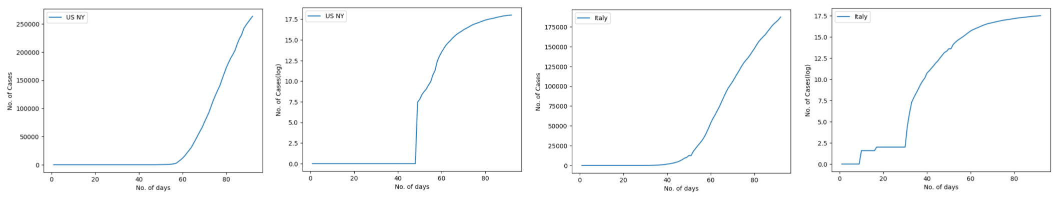


Figure 2: *Above shown are plots for 2 countries/places namely New York, US and Italy. For both the places, 2 plots are shown respectively: 1) Number of confirmed cases v/s no. of days. 2) log of number of confirmed cases v/s no.of days.*

3. Choice of meta kernel k as chosen above $e^{-w\|x_1-x_2\|^2}$

The hyper parameters $u(x)$ where $u(x) \in \mathbb{R}^2$, containing slope and intercept of the line formed by linear regression on input data vector $x(x \in \mathbb{R}^7)$, parameters w_1, w_2, w_3 for the base kernel found by grid search in previous steps, and the meta kernel k as chosen above.

Now we find parameter w (of meta-kernel k) through grid search using input as all hyper parameters samples $u(x)$ and output as parameters w_1, w_2, w_3. Through this we get the parameter value w of the meta kernel and hence the meta kernel.

Now we have found the meta kernel k, this will be used to auto-tune the parameters of the base kernel and hence provide good predictions for the number of confirmed cases.

6.5.4 Choice of base kernel

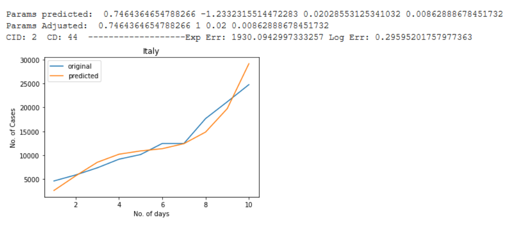
In figure 2, sample plots have been shown for 2 places. Since our model is predicting on the log curve, we want our kernel to be able to adapt to the log curve and predict correctly. If we carefully look at the log plots, they start increasing exponentially and then slow down. If extrapolated, it looks that they will eventually become constant and then come down. Hence for the exponential increasing part we would like to include a gaussian term in the kernel since it will adapt to it well too. But gaussians tend to underestimate at later stages (was also seen in our training when only with gaussian kernel), so we also use a polynomial kernel so that it pulls the curve and not underestimates. Hence we arrive at our base kernel choice as:

$$K(x_1, x_2; w_1, w_2, w_3) = (x_1 x_2)^{w_1} + w_2 e^{-w_3\|x_1-x_2\|^2}.$$

where w_1, w_2, w_3 are the parameters.

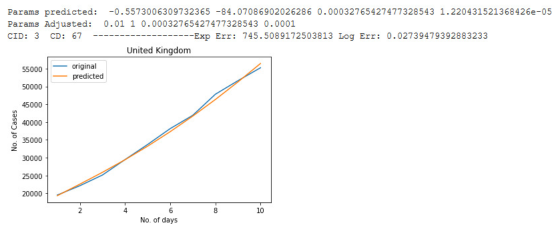
6.5.5 Choice of hyper-parameters (Meta-features)

We have taken the coefficients of least-square fit line which best fits the data. Note that in the present context one may, in principle, choose input vector x itself as a meta-feature. But, as it will be seen below, such a choice would essentially increase the dimensionality of the optimization problem in the final phase (grid search to find meta-kernel parameters) of the meta-learning.



```
Params predicted:  0.7464364654788266 -1.2332315514472283 0.02028553125341032 0.00862888678451732
Params Adjusted:  0.7464364654788266 1 0.02 0.00862888678451732
CID: 2  CD: 44  -------------------Exp Err: 1930.0942997333257 Log Err: 0.29595201757977363
```

Figure 3: *Above is the 3-day prediction plot starting from day 44 for Italy. The params field indicated the predicted parameters for the base kernel in the order w_1, w_2, w_3, γ. The adjusted params are the parameters got after using cutoff value on predicted parameters.*



```
Params predicted:  -0.5573006309732365 -84.07086902026286 0.00032765427477328543 1.220431521368426e-05
Params Adjusted:  0.01 1 0.00032765427477328543 0.0001
CID: 3  CD: 67  -------------------Exp Err: 745.5089172503813 Log Err: 0.02739479392883233
```

Figure 4: *Above is the 3-day prediction plot starting from day 67 for UK. The params field indicated the predicted parameters for the base kernel in the order w_1, w_2, w_3, γ. The adjusted params are the parameters got after using cutoff value on predicted parameters.*

Moreover, since the input vector x are formed by potentially noisy measurements (x_i), the use of low dimensional meta-features $u(x) = (u_1(x), u_2(x))$ can be seen as a regularization (denoising) by dimension reduction and as an overfitting prevention.

6.5.6 Model Prediction

Again as in figure 1, (now for the green boxes since we focus on prediction) we take the input data of past 7 days, x (number of confirmed cases) and as in the meta-training phase we again find the hyperparameters $u(x)$ (slope and intercept) through linear regression. Then we pass $u(x)$ to the meta kernel $k(u(x), w)$, where we got parameters w and hence the meta-kernel k at the end of the training phase of the model. The meta-kernel k gives us the parameters w_1, w_2, w_3 for the base kernel. Now we just give input x and parameters w_1, w_2, w_3 to the base kernel to predict the number of cases ahead. Note that the parameters w_1, w_2, w_3 given by the meta-kernel are tuned according to the input data x and hence the base kernel is adaptive according to the data, and hence called meta-learning.

6.5.7 Output Plots Analysis

In this section we show several examples and problems faced and how we tuned our model parameters to give good and accurate results.

In figure 3, the 3 day prediction plot, we can see that in tuning the parameters for the base kernel we used a cut off value too(in this case w_2 has been cut off). Hence some parameters are the different from the predicted value.The curve predicts nicely in this case. Similarly we can also see figure 4. In a similar way 5 day plots can also be seen in figures 5 and 6.

In figure 7 and 8, plots for Spain and Texas, US have been shown. As can be seen there are a quite a few oscillations or spike(though small in magnitude) in figure 7. These were large initially due to no cut off values and hence could be reduced drastically with cutoff values. In figure 8, it fits nicely though it tends to underestimate a bit but eventually estimates nicely after some time.

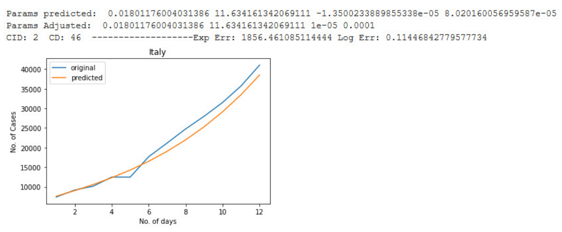

```
Params predicted:  0.01801176004031386 11.634161342069111 -1.3500233889855338e-05 8.020160056959587e-05
Params Adjusted:  0.01801176004031386 11.634161342069111 1e-05 0.0001
CID: 2  CD: 46  -------------------Exp Err: 1856.461085114444 Log Err: 0.11446842779577734
```

Figure 5: *Above is the 5-day prediction plot starting from day 46 for Italy. The params field indicated the predicted parameters for the base kernel in the order w_1, w_2, w_3, γ. The adjusted params are the parameters got after using cutoff value on predicted parameters.*

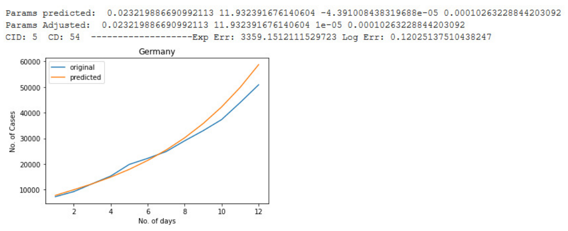

```
Params predicted:  0.023219886690992113 11.932391676140604 -4.391008438319688e-05 0.00010263228844203092
Params Adjusted:  0.023219886690992113 11.932391676140604 1e-05 0.00010263228844203092
CID: 5  CD: 54  -------------------Exp Err: 3359.1512111529723 Log Err: 0.12025137510438247
```

Figure 6: *Above is the 5-day prediction plot starting from day 54 for Germany. The params field indicated the predicted parameters for the base kernel in the order w_1, w_2, w_3, γ. The adjusted params are the parameters got after using cutoff value on predicted parameters.*

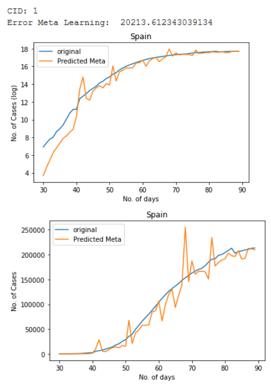

```
CID: 1
Error Meta Learning:  20213.612343039134
```

Figure 7: *Above are the 3rd day prediction plot for Spain. Upper graph is the logarithmic curve and below graph is the exponential or original curve.*

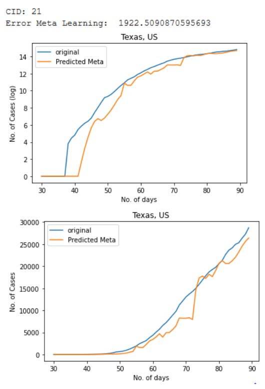

```
CID: 21
Error Meta Learning:  1922.5090870595693
```

Figure 8: *Above are the 3rd day prediction plot for Texas, US. Upper graph is the logarithmic curve and below graph is the exponential or original curve.*

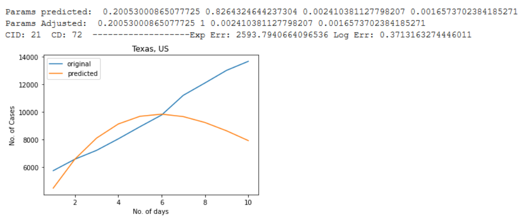

```
Params predicted:  0.20053000865077725 0.8264324644237304 0.002410381127798207 0.001657370238418527l
Params Adjusted:  0.20053000865077725 1 0.002410381127798207 0.0016573702384185271
CID: 21  CD: 72  -------------------Exp Err: 2593.7940664096536 Log Err: 0.3713163274446011
```

Figure 9: *Above graph is the 3 day prediction for Texas, US starting from day 72. This graph is plotted without tuning in grid search step of the model.*

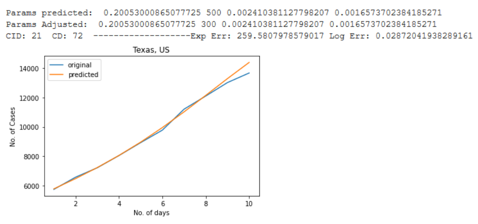

```
Params predicted:  0.20053000865077725 500 0.002410381127798207 0.0016573702384185271
Params Adjusted:  0.20053000865077725 300 0.002410381127798207 0.0016573702384185271
CID: 21  CD: 72  -------------------Exp Err: 259.5807978579017 Log Err: 0.0287204193828916l
```

Figure 10: *Above graph is the corrected graph of figure 9 (on the left). This correction is made after tuning of grid search step in our model.*

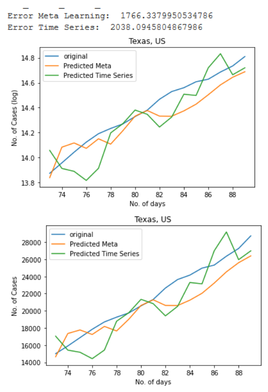

```
Error Meta Learning:  1766.3379950534786
Error Time Series:  2038.0945804867986
```

Figure 11: *Above are the 3rd day prediction plot for both time series and our kernel model v/s original curve for Texas, US. Upper graph is the logarithmic curve and below graph is the exponential or original curve.*

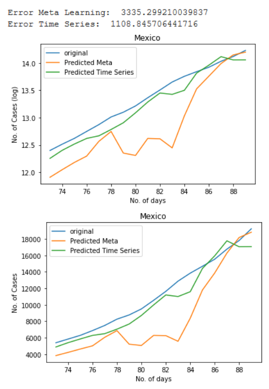

```
Error Meta Learning:  3335.299210039837
Error Time Series:  1108.845706441716
```

Figure 12: *Above are the 3rd day prediction plot for both time series and our kernel model v/s original curve for Mexico. Upper graph is the logarithmic curve and below graph is the exponential or original curve.*

Another tuning of model can be seen in figures 9 and 10. Initially while doing grid search we took a range of values for parameters but it wasn't exhaustive(it can never be) and the plot tend to predict poorly in some cases like in figure 9. In the correction step, during grid search we changed the range of parameter search and included a mix of some higher and some lower ranges. This also solved the problem of some poor predictions since for some predictions the base kernel needed very high value of some parameter and in some predictions it needed very low value of some parameters(this can be seen by the value of parameter w_2 in both the cases which initially predicted was 1 and later on correction it became 300).

6.5.8 Comparison with time-series model

In figure 11, we can see that the kernel model performs better than the time-series models. The time series models, gives a lot of peaks and variations in this case. On the contrary, in figure 12, we can see that the time-series model performs better than the kernel model. Since there is a large dip in the kernel method. Though if you carefully compare figures 11 and 12, you will find that the magnitude of deviation of kernel model is same in both the cases (around 2000s). It just looks magnified due to the scale of graph. On the other hand in general, time series model generally oscillates a lot and mostly in all cases it tends to converge at the end which is not the case with the kernel model.

6.6 Conclusion

In the above sections we made several key improvements by tuning the parameters in model. But still there is always scope to improve it, choosing a better family of kernels, both for base and meta kernel, improving grid search and making it more exhaustive, tweaking with the cut off values, taking different hyper parameters to reduce dimensionality of data (here we took linear regression coefficients), some different feature extraction or simply learning some other function transformation of the original data (here we took logarithm transformation). Apart from the improvements, we tried to show that these kind of kernalized learning methods are also very powerful and can give good results which can compete with state of the art models like the time series model taken here.

References

[1] László Györfi. *A Distribution-Free Theory of Non-parametric Regression.* Springer, 2003.

[2] G. Cybenko (1989) *Approximation by Superpositions of a Sigmoidal Function.* Math. Control Signals Systems (1989) 2: 303-314

[3] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, Qianli Liao (2017) *Why and When Can Deep-but Not Shallow-networks Avoid the Curse of Dimensionality: A Review.* International Journal of Automation and Computing (2017): 503-519

[4] Ding-Xuan Zhou (2018) *Universality of Deep Convolutional Neural Networks.* Applied and Computational Harmonic Analysis: 787-794

[5] V. Naumova, S.V. Pereverzyev, S. Sivananthan (2012) *A meta-learning approach to the regularized learning-Case study: Blood glucose prediction.* Neural Networks: Volume 33, September 2012, Pages 181-193

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[7] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels.* The MIT Press, 2002.

[8] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[9] Tomaso Poggio and Steve Smale. *http://www.ams.org/notices/200305/fea-smale.pdf* 2003.

[10] Ariel Schvartzman. *https://www.cs.princeton.edu/courses/archive/spring16/cos511/lec2.pdf* 2016.

[11] Link to dataset used in experiment. *https://www.kaggle.com/c/covid19-global-forecasting-week-4* 2020.

[12] Link to code repository. *GitHub*