Machine Learning Application
Winter 2020

(Sendhil Mullainathan)

Notes by Hyunwoo Roh

University of Chicago

Demystifying the Prediction: Short Journey from theory to practice

1 Secret sauce of Prediction

We know that using observation data in training set can mislead the learner (model), and result in overfitting. Overfitting happens through diverse channels and the most well known case is when the prediction model f becomes overly complicated or expressive (i.e., 'n'-th order polynomial). To overcome this problem, we can simply restrict the search space of function (i.e., set limit to the order of polynomial). However, such a process to deal with overfitting can be complicated due to the the following problem or limitation.

- Algorithmic limitation
  - We don't know that the true function class (if exists) is included in the function class that we choose
  - We hope that the chosen or well known function classes incorporate the true function class. In analysis, therefore, we separate the function class into two.
- Data knowledge limitation
  - We don't know what the true underlying joint distribution Pop is over the (X, Y)
  - We hope that distribution of observed data $(x_{i,}y_i)$ represent the population distribution well under the i.i.d assumption.
- Computational limitation
  - There is a computation limit to find a best parameter or weights when using the selected best function class.
  - We hope that a set of parameter within the computational limit is as close as to the one beyond the computational limit.

Assumptions in "Learning theory" (by Vapnik in Rodrigo' Book)

1. No assumption is made about the joint probability function $(x, y) \sim Pop$
2. Examples must be sampled from Pop in an independent manner.
   (a) Sequence (x, y) is realized by Pop
3. P_{XY} is static, so it does not change over time
   (a) Not suitable for problems involving time dependent data, as typical for real-world time series data
   (b) We will generalize this assumption for non-stationary or non-mixing data.
4. P_{XY} is unknown at the training stage

2 End Goal of Standard Machine Learning Problem

The end goal in practice is to find a best set of parameters or weights associated with the best function or algorithm with given loss l such that

$$f_\Phi^* = comp \min_\Phi \underbrace{\mathbb{E}_{(x,y)^{\overset{i.i.d}{\sim} P_{XY}}} [l(f(x\,;\Phi), y)]}_{\text{in-sample (validation) loss}} \quad \text{over} \quad \underbrace{f \in F}_{\text{function class}} \quad \text{subject to} \quad \underbrace{R(f) \leq c}_{\text{complexity restriction}}$$

where the selected best function along with best parameter set achieves the computational minimum of expected in-sample (validation) loss within the chosen function class given the constraint on model complexity.
Then, the prediction is done as follows:

$$\hat{y} = f^*(x; \Phi^*)$$

Here, the parameter plays a different role in Data model and Algorithm model that are introduced in Brieman paper (2005). Data modelling approach views a set of parameters in a way that it controls the data generating process whereas the algorithmic approach views it in a way that it controls the how algorithm should learn from the data.

3 Notation

- $\mathcal{X}$: input space, instance, feature, independent variable domain (a set)
- $\mathcal{Y}$: output space, target variable, dependent variable, label domain (a set)
  - $y \in \mathcal{Y}$: true value
    - Binary classification: $y = \{0, 1\}$
    - Regression problem: $y \in \mathbb{R}$
  - $\hat{y} \in \mathcal{Y}$: predicted value
- Φ : parameter / weight space

3.1 Samples

- $Train = S_n = ((x_1, y_1), ...(x_n, y_n))$: in-sample observations we have in training data set
  - We assume the samples $\{X_i, Y_i\}_{i=1}^n$ are i.i.d from population (true and unknown) joint distribution over $\mathcal{X} \times \mathcal{Y}$
    - Independence captures the idea that these come from different sources, and that these do not carry information about each other
    - Identically distributed assumption captures the idea that these are somehow a representative sample of the type of instances we might encounter in the future.
  - Also called Training set
- $Test = S_{out} = ((x_{n+1}, y_{n+1}), ...)$: out-of-sample observations we don't have in training data set
  - We test on S_{out} data
  - infinite sequence of pairs that will be sampled (not yet observed) from true (unknown) population distribution Pop over $\mathcal{X} \times \mathcal{Y}$
  - Also called Test set
- $Population = S = S_n \cup S_{out} = \mathcal{X} \times \mathcal{Y}$
  - Population where its population distribution Pop over $\mathcal{X} \times \mathcal{Y}$

3.2 Function

We as a learner usually have a function class in our mind. This function class represents a prior knowledge or expert knowledge. We often know what the function class is but we barely know what exactly function f is.

- F : function class (often called hypothesis class")
  - $f \in F$: function, predictor, hypothesis, classifier
  - We only take care of measurable functions where we can compute the expectation.
- F_A : function class within the algorithmic limit
  - We differentiate F_A from F. In other words, we do not require F to include the best true model.
  - $f_A \in F_A \subset F$
- l : loss function
  - Given any set F (model) and some domain $\mathcal{X} \times \mathcal{Y}$, let l be any function from $l : F \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$
  - loss function measures how "different" are between actual value and predicted value

3.3 Different Types of Losses (Risk)

3.3.1 4 Losses without specific function (predictor, hypothesis)

Here, we consider 4 different type of losses. Note that we want to choose a map $\hat{f}_n \in F$. It is important to note that the selected model $\hat{f}_n$ is a function of the training data

$$\hat{f}_n(x) = f(x; Train)$$

The subscript symbol n denotes the dependence on the training set size, and the "hat" indicates this is a function of the training dataset. Since we cannot make the loss as small as possible regardless of the test samples, we target the loss to be small on average. We take an average over population and training sample.

- $L(\hat{f}_n) = E_{(x,y)\sim Pop}\left[l(\hat{f}_n(x), y) \mid \hat{f}_n\right]$: **Population Loss (Statistical Risk)**
  - Expectation is over population distribution where the function is dependent on random training sample.
  - Conditional statement is there to ensure the definition is sensible even if f is a random quantity especially when the function is dependent on random training sample. This makes sense because we usually select the function based on the realized training sample
  - We will also consider the function where it is independent of the training data
  - Expectation of instantaneous loss with respect to (unknown) true/population Pop over $\mathcal{X} \times \mathcal{Y}$
  - In STL, it is called Statistical risk
  - $\int_{\mathcal{X}\times\mathcal{Y}} l(f(x), y) dP_{XY}(x, y) = \int_{\mathcal{X}\times\mathcal{Y}} l(f(x), y) p_{XY} dxdy$
- $\hat{L}(\hat{f}_n) = \mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}_n(x), y) \mid \hat{f}_n\right]$: **Empirical Loss (Empirical Risk)**
  - $\frac{1}{N_{Train}} \sum_{(x,y)\in Train}\left[l(\hat{f}_n(x), y) \mid \hat{f}_n\right]$
  - In STL, it is called Empirical Risk
  - This statement does not make any assumption about the true population distribution P_{XY}. So the data can be produced by ANY distribution whatsoever.

Consider that we take expectation over the training data and any randomized choices of the learning algorithms.

- $E\left[L(\hat{f}_n)\right]$: Averaged Population Loss

$$E\left[E_{(x,y)\sim Pop}\left[l(\hat{f}_n(x),y) \mid \hat{f}_n\right]\right] = \mathbf{E}_{(\mathbf{x},\mathbf{y})\sim \mathbf{Pop}}\left[\mathbf{l}(\hat{\mathbf{f}}_\mathbf{n}(\mathbf{x}),\mathbf{y})\right]$$

  - Out-of-sample Evaluation of Algorithm
  - Compared to this, $L(\hat{f}_n)$ is a finer measure since it takes into consideration the specific realization of the stochastic process and does not average over all possible training dataset and associated functions.

- $E\left[\hat{L}(\hat{f}_n)\right]$: Averaged Empirical Loss

$$E\left[\mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}_n(x),y) \mid \hat{f}_n\right]\right] = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \mathbf{Train}}\left[\mathbf{l}(\hat{\mathbf{f}}_\mathbf{n}(\mathbf{x}),\mathbf{y})\right]$$

  - In-sample $\mathbb{E}$valuation of Algorithm

3.3.2 Best Functions respect to "Expected Population Loss" and "Expected Empirical Loss"

Above, we considered losses without specific function. Here, we will choose four types of functions that is the best in terms of Averaged Population Loss and Averaged Empirical Loss for each function class F and $F_\mathcal{A}$

- $f^* = \arg\underbrace{\min}_{f\in F}\mathbf{E}\left[\mathbf{L}(\hat{\mathbf{f}}_\mathbf{n})\right] = \arg\underbrace{\min}_{f\in F} E_{(x,y)\sim Pop}\left[l(\hat{f}_n(x),y)\right]$
  - Best function w.r.t Expected **Population** Loss "beyond" **Algorithmic Limit**
  - Such chosen function is independent of training sample
- $f^*_\mathcal{A} = \arg\underbrace{\min}_{f\in F_\mathcal{A}}\mathbf{E}\left[\mathbf{L}(\hat{\mathbf{f}}_\mathbf{n})\right] = \arg\underbrace{\min}_{f\in F_\mathcal{A}} E_{(x,y)\sim Pop}\left[l(\hat{f}_n(x),y)\right]$
  - Best function w.r.t Expected **Population** Loss "within" **Algorithmic Limit**
  - Such chosen function is independent of training sample
- $\hat{f}^*_n = \arg\underbrace{\min}_{f\in F}\mathbf{E}\left[\hat{\mathbf{L}}(\hat{\mathbf{f}}_\mathbf{n})\right] = \arg\underbrace{\min}_{f\in F}\mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}_n(x),y)\right]$
  - Best function w.r.t Expected **Empirical** Loss "beyond" **Algorithmic Limit**
  - Such chosen function is dependent of training sample
- $\hat{f}^*_{\mathcal{A},n} = \arg\underbrace{\min}_{f\in F_\mathcal{A}}\mathbf{E}\left[\hat{\mathbf{L}}(\hat{\mathbf{f}}_\mathbf{n})\right] = \arg\underbrace{\min}_{f\in F_\mathcal{A}}\mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}_n(x),y)\right]$
  - Best function w.r.t Expected **Empirical** Loss "within" **Algorithmic Limit**
  - Such chosen function is dependent of training sample

3.3.3 Evaluation associated with 4 types of best functions: Analysis Toolkit

Here, we evaluate the chosen function in two different ways

1. Take expectation over population (not calculable)
   (a) This is for the sake of theoretical analysis
2. Take expectation over samples (calculable)

At the end, we ended up with 8 different types of losses.

- $L(f^*) = E_{(x,y)\sim Pop}\left[l(f^*(x), y)\right]$: Irreducible population loss of f^*
  - Best out of sample function beyond algorithmic limit evaluated using out of sample
  - Not computable and this is independent of randomly drawn samples (training set)
- $L(f^*_{\mathcal{A}}) = E_{(x,y)\sim Pop}\left[l(f^*_{\mathcal{A}}(x), y)\right]$: Irreducible population loss of $f^*_{\mathcal{A}}$
  - Best out of sample function within algorithmic limit evaluated using out of sample
  - Not computable and this is also independent of randomly drawn samples
- $L(\hat{f}^*_n) = E_{(x,y)\sim Pop}\left[l(\hat{f}^*_n(x), y) \mid Train\right]$
  - Best in sample function beyond algorithmic limit evaluated using out of sample
  - We use conditional statement here because $\hat{f}^*_n$ depends on randomly drawn training data set.
  - Still not computable and best in sample function dependent on randomly drawn training dataset
- $L(\hat{f}^*_{\mathcal{A},n}) = E_{(x,y)\sim Pop}\left[l(\hat{f}^*_{\mathcal{A},n}(x), y) \mid Train\right]$
  - Best in sample function within algorithmic limit evaluated using out of sample
  - We use conditional statement here because $\hat{f}^*_{\mathcal{A},n}$ depends on random sample
  - Still not computable and best in sample function dependent on randomly drawn training dataset

To sum up, by construction,

$$L(f^*) < L(f^*_{\mathcal{A}}) < \min\left(L(\hat{f}^*_n), L(\hat{f}^*_{\mathcal{A},n})\right)$$

- $\hat{L}(f^*) = \mathbb{E}_{(x,y)\sim Train}\left[l(f^*(x), y)\right]$
  - Best out of sample function beyond algorithmic limit evaluated using in sample
  - Not computable
- $\hat{L}(f^*_{\mathcal{A}}) = \mathbb{E}_{(x,y)\sim Train}\left[l(f^*_{\mathcal{A}}(x), y)\right]$
  - Best out of sample function within algorithmic limit evaluated using in sample
  - Not computable
- $\hat{L}(\hat{f}^*_n) = \mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}^*_n(x), y) \mid Train\right]$
  - Best in sample function beyond algorithmic limit evaluated using in sample
  - Not computable

- $\hat{L}(\hat{f}^*_{\mathcal{A},n}) = \mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}^*_{\mathcal{A},n}(x), y) \mid Train\right]$
  - Best in sample function within algorithmic limit evaluated using in sample
  - Computable!

To sum up, by construction,

$$\hat{L}(\hat{f}^*_n) < \hat{L}(\hat{f}^*_{\mathcal{A},n}) < \min\left(\hat{L}(f^*_{\mathcal{A}}), \hat{L}(f^*)\right)$$

4 Bias - Complexity tradeoff

Back to the main question, what do we want to make it as small as possible? We want

$$L(\hat{f}^*_{\mathcal{A},n}) = E_{(x,y)\sim Pop}\left[l(\hat{f}^*_{\mathcal{A},n}(x), y) \mid Train\right]$$

to be as small as possible. However, this quantity depends on the different possible realization of training set. Therefore, to get rid of this randomness, we take expectation over training data. Then,

$$\begin{aligned}
\mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right)\right] =& \mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right) + \mathbf{L}\left(\mathbf{f}^*_{\mathcal{A}}\right) - \mathbf{L}\left(\mathbf{f}^*_{\mathcal{A}}\right) + L(f^*) - L(f^*)\right] \\
=& \mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right) - \mathbf{L}\left(\mathbf{f}^*_{\mathcal{A}}\right)\right] + \mathbf{L}\left(\mathbf{f}^*_{\mathcal{A}}\right) + L(f^*) - L(f^*) \\
=& \underbrace{\mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right) - \mathbf{L}\left(\mathbf{f}^*_{\mathcal{A}}\right)\right]}_{\text{Estimation Error, } \geq 0} + \underbrace{L(f^*)}_{\text{Irreducible Error}} \\
&+ \underbrace{\mathbf{L}\left(\mathbf{f}^*_{\mathcal{A}}\right) - L(f^*)}_{\text{Approximation Error}}
\end{aligned}$$

- Irreducible error
  - We cannot reduce this error.
  - Also called as Bayes risk
- Approximation error
  - the minimum approximation error can be achieved by choosing the best $f^*(x)$ predictor.
  - This term measures how much error we have because we restrict ourselves to a specific class $f^*_{\mathcal{A}}(x)$. It also called as model mis-specification error.
  - Enlarging the functional class can decrease the approximation error.
  - The approximation error does not depend on the sample size and is determined by the functional class chosen.
- Estimation error
  - Estimation error occurs because $\hat{f}^*_{\mathcal{A},n}$ is an in sample minimizer while $f^*_{\mathcal{A}}$ is out of sample minimizer.

$$\begin{aligned}
&\mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right)\right] \\
=&\mathbb{E}_{(x,y)\sim Train}\left[E_{(x,y)\sim Pop}\left[l\left(\hat{f}^*_{\mathcal{A},n}(x), y\right) \mid Train\right]\right] \\
=&E_{(x,y)\sim Pop}\left[l\left(\hat{f}^*_{\mathcal{A},n}(x), y\right)\right] \\
>&E_{(x,y)\sim Pop}\left[l\left(f^*_{\mathcal{A}}(x), y\right)\right] = L\left(f^*_{\mathcal{A}}\right)
\end{aligned}$$

We hope that as $n \to \infty$

$$
\begin{aligned}
& E_{(x,y)\sim Pop}\left[l\left(\hat{f}^*_{\mathcal{A},n}(x), y\right) - l\left(f^*_{\mathcal{A}}(x), y\right)\right] \to 0 \\
\Longleftrightarrow & \hat{f}^*_{\mathcal{A},n}(x) \text{ gets closes to } f^*_{\mathcal{A}}(x) \\
\Longleftrightarrow & l\left(\arg \underbrace{\min}_{f\in F_{\mathcal{A}}} \left\{\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \mathbf{Train}}\left[l(\mathbf{f}(\mathbf{x}), \mathbf{y})\right] - \mathbf{E}_{(\mathbf{x},\mathbf{y})\sim \mathbf{Pop}}\left[l(\mathbf{f}(\mathbf{x}), \mathbf{y})\right]\right\}(x), y\right) \to 0 \\
\Longleftrightarrow & l\left(\arg \underbrace{\min}_{f\in F_{\mathcal{A}}} \left\{\frac{1}{\mid N_{Train} \mid} \sum_{(x,y)\in Train} [l(f(x), y)] - E_{Pop}\left[l\left(f(x), y\right)\right]\right\}(x), y\right) \to 0
\end{aligned}
$$

- The quality of this estimation depends on the training set size N_{Train} , and the complexity of the functional class or equivalently the size of functional class
  - If we increase the training set size, by LLN the difference will decrease. To get the upper bound of such difference, we use Chernoff inequality, later extended by Hoeffding inequality
  - If we increase the complexity of algorithm, or increase the size of functional class, both value decreases simultaneously, let alone the difference.

4.1 More on Estimation Error

We further decompose the estimation error.

$$
\begin{aligned}
& \mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right) - L\left(f^*_{\mathcal{A}}\right)\right] \\
= & \mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right) - \hat{\mathbf{L}}(\hat{\mathbf{f}}^*_{\mathcal{A},\mathbf{n}}) + \hat{\mathbf{L}}(\hat{\mathbf{f}}^*_{\mathcal{A},\mathbf{n}}) - \hat{L}(f^*_{\mathcal{A}}) + \hat{L}(f^*_{\mathcal{A}}) - L\left(f^*_{\mathcal{A}}\right)\right] \\
= & \underbrace{\mathbb{E}_{(x,y)\sim Train}\left[\underbrace{\hat{\mathbf{L}}(\hat{\mathbf{f}}_{\mathcal{A},\mathbf{n}}) - \hat{L}(f^*_{\mathcal{A}})}_{\text{by construction, } \leq 0}\right]}_{\text{wrong function looks good in-sample}} + \underbrace{\mathbb{E}_{(x,y)\sim Train}\left[\underbrace{L\left(\hat{f}^*_{\mathcal{A},n}\right) - \hat{\mathbf{L}}(\hat{\mathbf{f}}^*_{\mathcal{A},\mathbf{n}})}_{\text{by construction, } \geq 0}\right]}_{\text{unobserved overfit} \xrightarrow{p} 0} + \underbrace{\mathbb{E}_{(x,y)\sim Train}\left[\underbrace{\hat{L}(f^*_{\mathcal{A}}) - L(f^*_{\mathcal{A}})}_{\text{by construction, } \geq 0}\right]}_{\text{uncontrollable overfit} \xrightarrow{p} 0}
\end{aligned}
$$

Eventually, in an estimation error, what we want to manage is $L\left(\hat{f}^*_{\mathcal{A},n}\right)$ which is unobserved out-of-sample loss.

$$
\underbrace{L\left(\hat{f}^*_{\mathcal{A},n}\right)}_{\text{unobserved out-of-sample loss}} = \overbrace{\hat{L}(\hat{f}_{\mathcal{A},n})}^{\text{observed in-sample loss}} + \underbrace{\left(L\left(\hat{f}^*_{\mathcal{A},n}\right) - \hat{\mathbf{L}}(\hat{\mathbf{f}}_{\mathcal{A},\mathbf{n}})\right)}_{\text{unobserved overfit}}
$$

Although we do well on in-sample, some of that "fit" is overfit from the out-of-sample point of view. If we do well on out-of-sample, again some of that "fit" is overfit from the in-sample point of view.

"Here comes the regularization on model complexity"

5 Evaluation of Best In-sample Algorithm

From above, we have seen how to decompose $\mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right)\right]$ into estimation error and approximation error. Then we further decompose the estimation error into three parts. We plug everything back to get

$$\mathbb{E}_{(x,y)\sim Train}\left[L\left(\hat{f}^*_{\mathcal{A},n}\right)\right] = \underbrace{\mathbf{L}\left(\mathbf{f}^*_{\mathcal{A}}\right) - L(f^*)}_{\text{Approximation Error}} + \underbrace{L(f^*)}_{\text{Irreducible Error}} + \underbrace{\mathbb{E}_{(x,y)\sim Train}\left[\underbrace{\hat{L}(f^*_{\mathcal{A}}) - \mathbf{L}(\mathbf{f}^*_{\mathcal{A}})}_{\text{by construction, } \geq 0}\right]}_{\text{uncontrollable overfit} \xrightarrow{p} 0}$$

$$+ \underbrace{\mathbb{E}_{(x,y)\sim Train}\left[\underbrace{\hat{\mathbf{L}}(\hat{\mathbf{f}}^*_{\mathcal{A},\mathbf{n}}) - \hat{L}(f^*_{\mathcal{A}})}_{\text{by construction, } \leq 0}\right]}_{\text{wrong function looks good in-sample}} + \underbrace{\mathbb{E}_{(x,y)\sim Train}\left[\underbrace{L(\hat{f}^*_{\mathcal{A},n}) - \hat{\mathbf{L}}(\hat{\mathbf{f}}^*_{\mathcal{A},\mathbf{n}})}_{\text{by construction, } \geq 0}\right]}_{\text{unobserved overfit} \xrightarrow{p} 0}$$

To sum up,

- Complexity constraint threshold of algorithm **(c)** $\uparrow \equiv$ enlarge the set of function class (complicated)
  $\Longrightarrow$ Approximation error $\downarrow$, In-sample error $\downarrow$, Unseen error from overfit $\uparrow$
- Complexity constraint threshold of algorithm **(c)**$\downarrow \equiv$ reduce the set of function class (simpler)
  $\Longrightarrow$ Approximation error $\uparrow$, In-sample error $\uparrow$, Unseen error from overfit $\downarrow$

6 What happens if we deal with time series data?

When we deal with the time series data, there is an extra concern. One biggest problem is that time series data is known to have distribution changing over time. Although we believe that there is a little difference between the distribution of the near past and near future, we don't know until when that similarity would remain (hold) if it is subject to change at some point. Previously in the standard learning scenario, we assume that we receive a sample $(X_1, Y_1), \cdots, (X_n, Y_n)$ are $i.i.d$ drawn from the unknown $Population$ distribution. However, that is not often the case. The simplest example would be

$$Y_t = \alpha_t Y_{t-1} + \epsilon_t, \qquad \alpha_t = -0.9 \text{ if } t \in [1000, 2000] \text{ and } 0.9 \text{ otherwise}$$

In this case with synthetic example, we obviously know that there are two different ways of data generating process. For the sake of simplicity of the analysis, we assume that we know in priori there are two different population distribution and randomly sampled training data set respectively.

$$Pop = Pop_1 + Pop_2$$
$$Train = Train_1 + Train_2$$

We can easily generalize to m different subset of population and train data. Let's reconsider the all the definitions above.

- $L(f^*) = E_{(x,y)\sim Pop}\left[l(f^*(x), y)\right]$ and $L(f^*_{\mathcal{A}}) = E_{(x,y)\sim Pop}\left[l(f^*_{\mathcal{A}}(x), y)\right]$
  - The above two does not change much with time series data. This is because we expect f^* or $f^*_{\mathcal{A}}$ to capture everything because this is the most ideal function although the population distribution changes over time.
- $L(\hat{f}^*_n) = E_{(x,y)\sim Pop}\left[l(\hat{f}^*_n(x), y) \mid Train\right]$

  - There is a room to analyze this quantity more. Even if we choose a function class that considers the state dependent data generating process, there is a limit on perfectly estimating the different hidden states.

- $L'(\hat{f}_n^*) = E_{(x,y)\sim Pop_1}\left[l(\hat{f}_n^*(x), y) \mid Train_1\right] + E_{(x,y)\sim Pop_2}\left[l(\hat{f}_n^*(x), y) \mid Train_2\right]$
  - In general with m,
$$\sum_{i=1}^{m} E_{(x,y)\sim Pop_i}\left[l(\hat{f}_n^*(x), y) \mid Train_i\right]$$
  - By construction,
$$L'(\hat{f}_n^*) \lesssim L(\hat{f}_n^*)$$
    equality holds when the data is stationary.
- $L(\hat{f}_{\mathcal{A},n}^*) = E_{(x,y)\sim Pop}\left[l(\hat{f}_{\mathcal{A},n}^*(x), y) \mid Train\right]$
- $L'(\hat{f}_{\mathcal{A},n}^*) = E_{(x,y)\sim Pop_1}\left[l(\hat{f}_{\mathcal{A},n}^*(x), y) \mid Train_1\right] + E_{(x,y)\sim Pop_2}\left[l(\hat{f}_{\mathcal{A},n}^*(x), y) \mid Train_2\right]$

To sum up, by construction,

$$L(f^*) < L(f_{\mathcal{A}}^*) < \min\left(L'(\hat{f}_n^*), L'(\hat{f}_{\mathcal{A},n}^*)\right) \lesssim \min\left(L(\hat{f}_n^*), L(\hat{f}_{\mathcal{A},n}^*)\right)$$

- $\hat{L}(f^*) = \mathbb{E}_{(x,y)\sim Train}\left[l(f^*(x), y)\right]$ and $\hat{L}(f_{\mathcal{A}}^*) = \mathbb{E}_{(x,y)\sim Train}\left[l(f_{\mathcal{A}}^*(x), y)\right]$
  - Best out of sample functions believed to capture stationarity so that it is not meaningful to divide into two training set in evaluation.
- $\hat{L}(\hat{f}_n^*) = \mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}_n^*(x), y) \mid Train\right]$
  - There is a room to analyze this quantity more. Even if we choose a function class that considers the state dependent data generating process, there is a limit on perfectly estimating the different hidden states. Instead, we know that two training samples are randomly sampled from two different population distribution.
- $\hat{L}'(\hat{f}_n^*) = \mathbb{E}_{(x,y)\sim Train_1}\left[l(\hat{f}_n^*(x), y) \mid Train_1\right] + \mathbb{E}_{(x,y)\sim Train_2}\left[l(\hat{f}_n^*(x), y) \mid Train_2\right]$
  - In general with m,
$$\hat{L}'(\hat{f}_n^*) = \sum_{i=1}^{m} \mathbb{E}_{(x,y)\sim Train_i}\left[l(\hat{f}_n^*(x), y) \mid Train_i\right]$$
- $\hat{L}(\hat{f}_{\mathcal{A},n}^*) = \mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}_{\mathcal{A},n}^*(x), y) \mid Train\right]$
  - Computable!
  - In time series this is path dependent loss
  - $\mathbb{E}_{(x,y)\sim Train}\left[l(\hat{f}_{\mathcal{A},n}^*(x), y) \mid (x,y)_1 \cdots (x,y)_T\right]$
- $\hat{L}'(\hat{f}_{\mathcal{A},n}^*) = \mathbb{E}_{(x,y)\sim Train_1}\left[l(\hat{f}_{\mathcal{A},n}^*(x), y) \mid Train_1\right] + \mathbb{E}_{(x,y)\sim Train_2}\left[l(\hat{f}_{\mathcal{A},n}^*(x), y) \mid Train_2\right]$
  - this is not computable because we don't know in prior where the train1 and train2 are

- Since the above calculation is too idea, we instead think of path dependent loss.

$$\begin{aligned}\underleftarrow{\hat{L}}'(\hat{f}^*_{\mathcal{A},n}) &= \mathbb{E}_{(x,y)\sim Train}\left[l\left(\hat{f}^*_{\mathcal{A},n}(x_{t-1}), y_t\right) \mid (x,y)_1 \cdots (x,y)_T\right] \\ &= \mathbb{E}_{(x,y)\sim Train}\end{aligned}$$

- $\underleftarrow{\hat{L}}'(\hat{f}^*_{\mathcal{A},n}) = \min\left\{\underleftarrow{\hat{L}}'_b(\hat{f}^*_{\mathcal{A},n})\right\}$

Using the newly introduced notations, we can update the following:

Degree of Non-stationarity

We can see the degree of non-stationarity as the difference between $\hat{L}(\hat{f}^*_{\mathcal{A},n})$ and $\hat{L}'(\hat{f}^*_{\mathcal{A},n})$

$$\text{Non-stationarity} = \hat{L}'(\hat{f}^*_{\mathcal{A},n}) - \hat{L}(\hat{f}^*_{\mathcal{A},n})$$

Problem is that we cannot calculate this now that we don't know which data point belongs to which training set among many.

7 Example of function class and its parameters

To summarize some existing models in the case where there are n input variables & 1 target variable.

Function class F	**Parameters** Φ	**Regularizer** $R(f)$
Parametric predictor		
OLS Regression	$\beta \in \mathbb{R}^n$	None
OLS with $L0$	$\beta \in \mathbb{R}^n$	$\parallel \beta \parallel_0 = \sum_{j=1}^{n} \mathbf{I}_{\beta_i \neq 0}$
OLS with $L1$	$\beta \in \mathbb{R}^n$	$\parallel \beta \parallel_1 = \sum_{j=1}^{n} \mid \beta_j \mid$
OLS with $L2$	$\beta \in \mathbb{R}^n$	$\parallel \beta \parallel_1 = \sum_{j=1}^{n} (\beta_j)^2$
PCA Regression	$\beta \in \mathbb{R}^i \quad i \leq n$	None
PLS Regression	$\beta \in \mathbb{R}^i \quad i \leq n$	None
ARIMA		Orders of AR, MA, and Differencing
Non-parametric predictor		
Decision tree	Threshold values at each node	Depth, minimal leaf size, number of nodes/leaves, information gain at split
Random forest	Parameters of multiple trees	Number of trees, complexity of trees number of features for each tree
Kernel Regression	None	Kernel bandwidth (Smooth factor)
Gaussian Process	$\mathbb{R}^\infty$: function	Kernel and its hyper-parameters
Mixed predictors		
Single Neuron	$[W, b] \in \mathbb{R}^n$	None
FFNN (Feed Forward)	$[W, b] \in \mathbb{R}^{g_1(n, N_1, N_2)}$	Number of hidden layer (N_1), Number of units for each hidden layer ($N_2 \in R^{N_1}$), Structure on neurons and layers (g_1)
RNN (Recurrent)	$[W, b] \in \mathbb{R}^{g_2(n, N_3, N_4)}$	Number of FFNN (N_3) , Number of hidden units (N_4), Structure on neurons, layers, and RNN cells (g_2)
CNN (Convolutional)	$[W, b] \in \mathbb{R}^{g_3(n, N_5, N_6)}$	Filter size ($N_5 \times N_6$), Structure on neurons, layers, and CNN cells (g_3)
Splines (Regression)	$\beta \in \mathbb{R}^{c+k+1}$	Number of knots(c), Order of spline(k) Location of knots, Choice of basis function
Combined predictors		
Bagging	Parameters of weak (multiple) predictors & Weights on Boosting and Stacking	Number of draws, Size of bootstrap samples, Individual regulariszation hyper-parameter
Boosting		Learning rate, Number of iteration Individual regulariszation hyper-parameter
Stacking		Different weak predictors, Meta-model to decide how to combine weak predictors Hyper-parameter of chosen Meta-model

Table 1: Summary of Statistical Model, Machine learning Model, and Deep learning Model