Vapnik-Chervonenkis Dimension, Probabilistic Bounds and Neyman-Pearson Classification

Jiajun Zhang and Emile Petit
MATH 598 Presentation

November 23, 2025

Classifiers and Misclassification

Let $\mathcal{D}_n := \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)\}$ be the training set, $(\boldsymbol{x}_i, y_i)$ are jointly distributed random variables from the probability space $\mathcal{X} \times \mathcal{Y}$. We consider a binary classification problem where $\mathcal{Y} = \{0, 1\}$. A **classifier** is a Borel-measurable function $f : \mathcal{X} \to \mathcal{Y}$. Our goal is to find the "best" classifier based on the data we have and make predictions on other data. Misclassification occurs when $f(\boldsymbol{x}_i) \neq y_i$, and we may ask the following:

- How likely will a classifier fits the training sample well, but performs poorly outside the traing sample?
- How large will the training sample be in order to have a low misclassification rate overall?
- In general how can we control the misclassification rate under a certain level?

To answer those questions, we will study the theory of Vapnik-Chervonenkis (VC) dimension and the probabilistic bounds derived from it.

Constructing Binary Classifiers

Let $f : \mathcal{X} \to \{0, 1\}$ be a binary classifier, let $S \subset \mathcal{X}$, we define

$$f\Big|_S : S \to \{0, 1\}, \quad f\Big|_S(\boldsymbol{x}) = f(\boldsymbol{x}), \forall \boldsymbol{x} \in S$$

as the restriction of f to S. Let $\mathcal{C}$ be the class of all classifiers, we define

$$\Pi_{\mathcal{C}}(S) := \left\{ f\Big|_S : f \in \mathcal{C} \right\}$$

as the number of distinct restrictions of classifiers in $\mathcal{C}$ defined by S, i.e, the set of all possible dichotomies on S:

$$\Pi_{\mathcal{C}}(S) := \{(f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_m)), f \in \mathcal{C})\}$$

where we assume $S := \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m\}$ and $f(\boldsymbol{x}_i) \in \{0, 1\}$. Clearly $|\Pi_{\mathcal{C}}(S)| \leq 2^{|S|}$.

Vapnik-Chervonenkis Dimension

Definition: A finite set $S \subset \mathcal{X}$ is *shattered* by $\mathcal{C}$, if $|\Pi_{\mathcal{C}}(S)| = 2^{|S|}$. The *Vapnik-Chervonenkis Dimension* (VCD) of $\mathcal{C}$, denote as $\mathbf{VCD}(\mathcal{C})$ (or simply V) is the cardinality d of the largest set S shattered by $\mathcal{C}$. We say $\mathbf{VCD}(\mathcal{C}) = \infty$ if $\mathcal{C}$ shatters arbitrarily sets.

- In other words, $S = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m\} \subset \mathcal{X}$ is shattered by $\mathcal{C}$ if for every assignment of labels to those points, there exists $f \in \mathcal{C}$ such that it makes no error when evaluating that set of data points.

We also define the **Growth Function** as

$$\Pi_{\mathcal{C}}(m) = \max\{|\Pi_{\mathcal{C}}(S)|, |S| = m\}$$

where $m \in \mathbb{N}$. If $m \leq \mathbf{VCD}(\mathcal{C})$, we simply have $\Pi_{\mathcal{C}}(m) = 2^m$.

Examples of Shattering and VC-Dimension

Some Examples:

- **Interval Classifiers:** Consider the class of classifiers $f \in \mathcal{C}$ as intervals in $\mathbb{R}$. We then claim that $\mathbf{VCD}(\mathcal{C}) = 2$, i.e. an interval classifier can shatter any sets of two points.

Figure: Any interval can shatter 2 points

Figure: Any interval cannot shatter 3 points of this pattern

Examples of Shattering and VC-Dimension

Some Examples:

- **Rectangles in $\mathbb{R}^2$:** Consider $\boldsymbol{x} \in \mathbb{R}^2$ and define $\mathcal{C}$ as the class of all rectangles:

  $$[\alpha_1, \beta_1] \times [\alpha_2, \beta_2]$$

  We may assume any three points are not co-linear because if so then it reduces to the case of an interval classifier.

- We claim that $\mathbf{VCD}(\mathcal{C}) = 4$.

Examples of Shattering and VC-Dimension

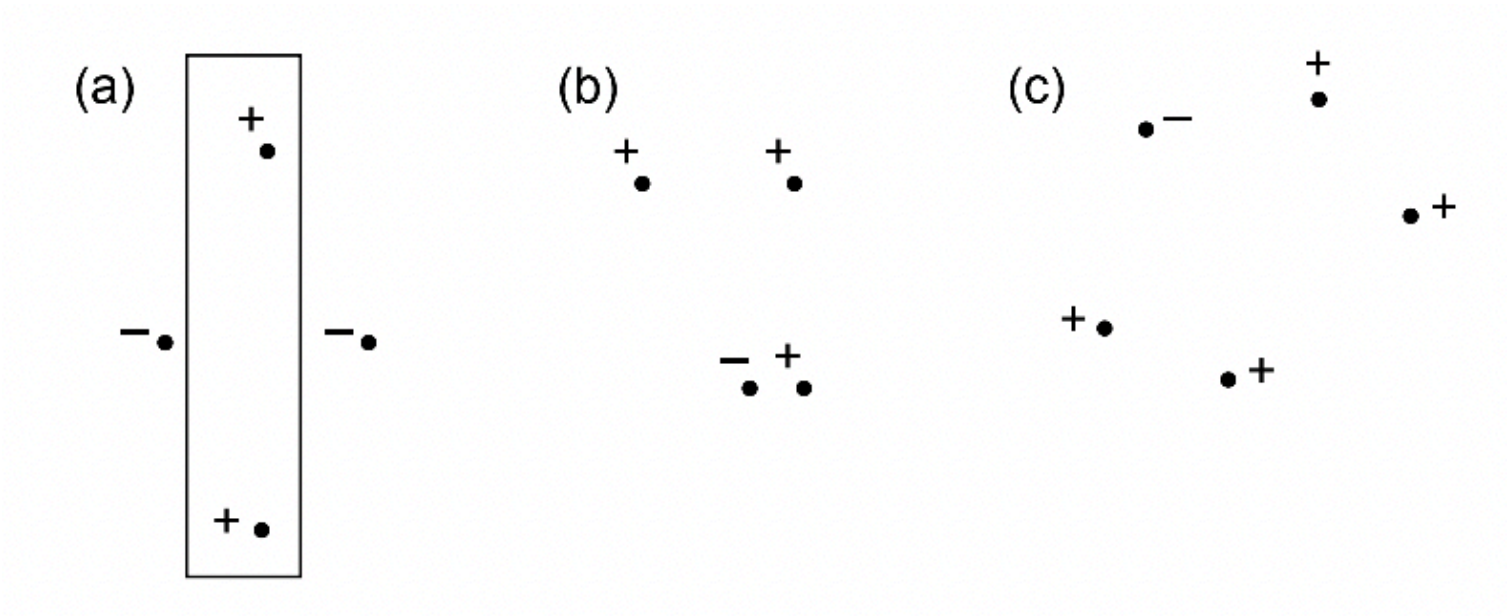

Figure: There exists a pattern of four points that can be shattered by rectangale, but not all patterns are possible. However no rectangle can shatter 5 points.

Examples of Shattering and VC-Dimension

Some Examples:

- **Linear Threshold Functions:** Consider $\boldsymbol{x} \in \mathbb{R}^n$ and define $\mathcal{C}$ as the class of all linear half spaces: $\boldsymbol{x}^\top \boldsymbol{\beta} > c$
- We claim that $\mathbf{VCD}(\mathcal{C}) = n + 1$.

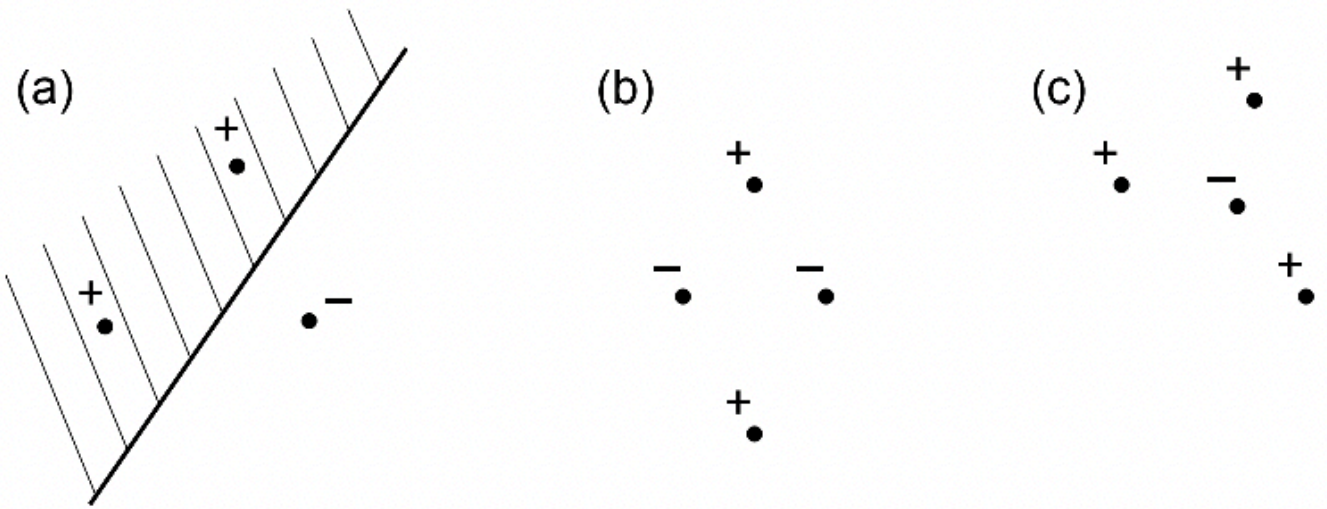

Figure: An example with $n = 2$. A linear half plane can shatter any 3 points but can not shatter any 4 points.

Examples of Shattering and VC-Dimension

Some Examples:

- **Convex d-gons in $\mathbb{R}^2$:** Consider $\mathcal{C}$ whose boundary is a convex d-gon in the plane.
- We claim that $\mathbf{VCD}(\mathcal{C}) = 2d + 1$.
- The idea is, we pick $2d + 1$ points on a circle, and for any labeling of those points, we can insert a convex polygon that either:
  - Has vertices as some of those points
  - Has edges tangent to the circle at some of those points

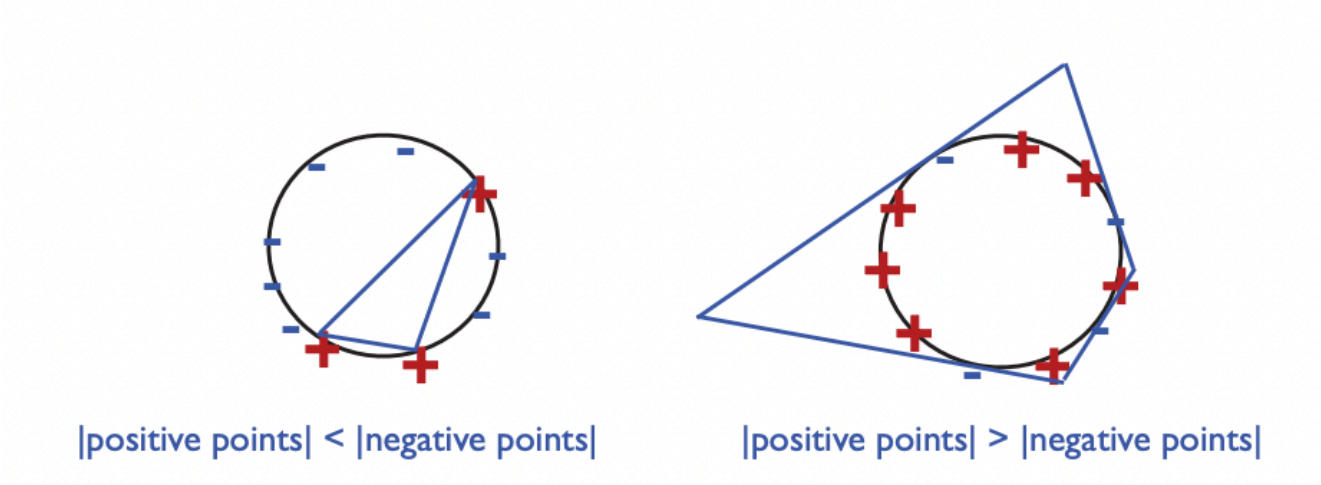

Figure: A 4-gon can shatter any 9 points on a circle

Properties of Growth Function

We have seen that the growth function satisfies $\Pi_{\mathcal{C}}(m) = 2^m$ if $m \leq \textbf{VCD}(\mathcal{C})$ What if $m > \textbf{VCD}(\mathcal{C})$? We define a function $\Phi_d(m)$ such that $\Phi_d(0) = 1, \Phi_0(m) = 1$ for all $d, m \in \mathbb{N}$ and

$$\Phi_d(m) = \Phi_d(m-1) + \Phi_{d-1}(m-1)$$

for all $d, m > 0$. Then we have

$$\Pi_{\mathcal{C}}(m) \leq \Phi_d(m)$$

if $\textbf{VCD}(\mathcal{C}) = d$. The proof is done using induction on d, m. Also for all m, d,

$$\Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}$$

Properties of Growth Function

Finally, we can bound $\Phi_d(m)$ by the following: When $m \geq d$, we have

Lemma (Sauer's Lemma)

Let $\mathbf{VCD}(\mathcal{C}) = d$, *for* $m \geq d$, *we have*

$$\Pi_{\mathcal{C}}(m) \leq \left(\frac{em}{d}\right)^d = \mathcal{O}(m^d).$$

Misclassification Rate

Suppose f is the optimal classifier, $f' \in \mathcal{C}$ be any classifier, let

$$(f' \oplus f)(\boldsymbol{x}) = \begin{cases} 1 & \text{if } f'(\boldsymbol{x}) \neq f(\boldsymbol{x}) \\ 0 & \text{otherwise} \end{cases}$$

- Define $err(f') = \mathbb{P}[(f \oplus f')(\boldsymbol{x}) = 1]$ as the *misclassification rate*, and $\Delta_f(\mathcal{C}) = \{f' \oplus f, f' \in \mathcal{C}\}$. We claim that $\mathbf{VCD}(\Delta_f(\mathcal{C})) = \mathbf{VCD}(\mathcal{C})$, and let

  $$\Delta_{f,\epsilon}(\mathcal{C}) = \{\tilde{f} \in \Delta_f(\mathcal{C}), \mathbb{P}[\tilde{f}(\boldsymbol{x}) = 1] \geq \epsilon\}$$

  i.e, the set of all classifiers in $\Delta_f(\mathcal{C})$ with misclassification rate at least ϵ.

ϵ-net

Definition: We say S is an ϵ-*net* for $\Delta_f(\mathcal{C})$ if $\forall \tilde{f} \in \Delta_{f,\epsilon}(\mathcal{C})$, $\exists \boldsymbol{x} \in S$ such that $\tilde{f}(\boldsymbol{x}) = 1$.

- It means that, if a classifier f' is "bad enough" that has misclassification rate of at least ϵ, then there must exist one point $\boldsymbol{x}$ such that f' misclassifies the label of $\boldsymbol{x}$;
- Conversely, if the training set S is an ϵ-net, then any classifier f' that fits the sample well (with zero training error) must have misclassification error $< \epsilon$.

Probabilistic Bounds

Now let S be the training set, we will draw two samples from S as follows:

- First draw a sample S_1 of size m, let A be the event that S_1 fails to be an ϵ-net for $\Delta_f(\mathcal{C})$.
- This means $\exists \tilde{f} \in \Delta_{f,\epsilon}(\mathcal{C})$ such that $\tilde{f}(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in S_1$, but has misclassification rate at least ϵ.
- Fix such a $\tilde{f}$ and draw a second sample S_2 of size m ($S_1 \cap S_2 = \varnothing$), then we know $\mathbb{P}(\tilde{f}(\boldsymbol{x}_i) = 1) \geq \epsilon$ for all $\boldsymbol{x}_i \in S_2$, and we have

$$X = \sum_{i=1}^{m} \tilde{f}(\boldsymbol{x}_i) \sim \mathit{Binomial}(m, p), p \geq \epsilon$$

we are interested in the probability

$$\mathbb{P}\left(X \leq \frac{\epsilon m}{2}\right)$$

Probabilistic Bounds

Lemma (Lower Tail Cherboff Bound)

Let $X \sim$ *Binomial*(m, p), $\mathbb{E}X = \mu$, *for all* $0 < \delta < 1$,

$$\mathbb{P}(X \leq (1-\delta)\mu)) \leq \exp\left(-\frac{\delta^2\mu}{2}\right)$$

Thus by this lemma, we can show that

$$\mathbb{P}\left(X \leq \frac{\epsilon m}{2}\right) \leq \frac{1}{2},$$

whenever $m \geq (8 \ln 2)/\epsilon$.

Probabilistic Bounds

- Now consider the event B as follows: A sample $S = S_1 \cup S_2$ of size $2m$ with $|S_1| = |S_2| = m$, with the classifier $\tilde{f}$ we fixed such that $|\{\boldsymbol{x} \in S, \tilde{f}(\boldsymbol{x}) = 1\}| \geq \epsilon m/2$ and $\tilde{f}(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in S_1$.
- We have already showed that,

$$\mathbb{P}(B|A) = \mathbb{P}(X \geq \epsilon m/2) \geq \frac{1}{2}$$

hence

$$\mathbb{P}(B) = \mathbb{P}(B|A)\mathbb{P}(A) \geq \frac{1}{2}\mathbb{P}(A) \quad \textbf{(Vapnik and Chervonenkis, 1971)}$$

Probabilistic Bounds

Now it is purely a combinatorial problem to bound $\mathbb{P}(B)$. B is equivalent as saying: If we are given $2m$ balls with $r \geq \epsilon m/2$ are red and the rest are black, then we divide them into 2 sets of m balls each, what is the probability that the first set has no red balls? The answer is simply $\binom{m}{r}/\binom{2m}{r}$, which can be bounded by

$$\frac{\binom{m}{r}}{\binom{2m}{r}} := \prod_{i=0}^{r-1} \frac{m-i}{2m-i} \leq \frac{1}{2^r}$$

Thus by Sauer's lemma we finally have

$$\mathbb{P}(A) \leq 2\mathbb{P}(B) \leq 2 \cdot \left|\Pi_{\Delta_{f,\epsilon}(\mathcal{C})}(S)\right| 2^{-\frac{\epsilon m}{2}} \leq 2 \cdot \left(\frac{2em}{d}\right)^d 2^{-\epsilon m/2}$$

Probabilistic Bounds

From the bound we obtained, for some δ, we solve

$$2\left(\frac{2em}{d}\right)^d 2^{-\epsilon m/2} \leq \delta$$

and we get

$$\frac{\epsilon m}{2} \geq \frac{d}{\ln 2} \ln\left(\frac{2em}{d}\right) + 1 - \log_2 \delta$$

By choosing m large enough, for instance

$$m \geq \kappa_0 \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d}{\epsilon} \log \frac{1}{\epsilon}\right)$$

where κ_0 is some universal constant, we would have $\mathbb{P}(A) \leq \delta$. It means that, if we draw at least m training samples, then with probability at least $1 - \delta$, every consistent classifier f on S will have misclassification error $< \epsilon$.

Probabilistic Bounds for Testing Error

Recall that a classifier is a Borel measurable function $f : \mathcal{X} \to \mathcal{Y} := \{0, 1\}$, given training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ we define

$$\hat{f} := \arg\min_{f \in \mathcal{C}} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{f(\boldsymbol{x}_i) \neq y_i\}$$

i.e., it is the classifier in $\mathcal{C}$ that minimizes the empirical misclassification rate over the training set. With the assumption that $\mathcal{Y} = f(\mathcal{X})$, we actually have $R(\hat{f}) = 0$, that is, the empirical risk minimization would be zero on the training set. We would always achieves zero training error, but then we would like to ask the misclassification error overall. We denote $R(\hat{f})$ as the testing error of $\hat{f}$.

Probabilistic Bounds for Testing Error

Theorem (Vapnik and Chervonenkis, 1971)

Let $\mathcal{C} := \{f : \mathcal{X} \to \{0, 1\}\}$ be a class such that $\mathbf{VCD}(\mathcal{C}) = d$. Given a training sample $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, let $\hat{f}$ be an ERM classifier (minimizes the empirical risk), then for all probability distribution P over $\mathcal{X} \times \mathcal{Y}$ that satisfies $\mathcal{Y} = f(\mathcal{X})$ for some $f \in \mathcal{C}$, for any $n \geq d, \epsilon > 0$,

$$\mathbb{P}(R(\hat{f}) \geq \epsilon) \leq 2 \left(\frac{2en}{d} \right)^d e^{-\epsilon n/2}$$

The theorem is equivalent as saying, for any $n \geq d$, with probability at least $1 - \delta$,

$$R(\hat{f}) \leq \frac{2\left(d \log \frac{2en}{d} + \log \frac{2}{\delta} \right)}{n}$$

Uniform Convergence of Empirical Risk

If we define $R(f)$ as the true misclassification rate, and $\hat{R}(f)$ as the empirical misclassification rate It can be shown that

Theorem (Vapnik and Chervonenkis, 1971)

$$\mathbb{P}\left(\sup_{f\in\mathcal{C}}|R(f)-\hat{R}(f)|>\epsilon\right)\leq 8\left(\frac{en}{d}\right)^{d}e^{-n\epsilon^2/32}$$

for any distribution P on $\mathcal{X}\times\mathcal{Y}$, any $n\geq d,\epsilon>0$.

Hence we have

$$\sum_{n=1}^{\infty}\mathbb{P}\left(\sup_{f\in\mathcal{C}}|R(f)-\hat{R}_n(f)|>\epsilon\right)\leq\sum_{n=1}^{\infty}Cn^{d}e^{-n\epsilon^2/32}<\infty$$

which guarantees the uniform convergence by Borel-Cantelli lemma, hence

$$\sup_{f\in\mathcal{C}}|R(f)-\hat{R}_n(f)|\to 0\quad a.s$$

References

- **1.** Computational Learning Theory: VC Dimension, Lecture 3, by Varun Kanade;
- **2.** On the Uniform Convergence of Relative Frequences of Events to Their Probabilities, by V.N. Vapnik and A. Ya. Chervonenkis;
- **3.** Machine Learning Theory (CSC 482A/581B), Lectures 7 & 8, by Nishant Mahta;

Type I & II Error

- Misclassification occurs when $f(\boldsymbol{x}_i) \neq y_i$, it can be summarized into two types of misclassification:
  - **Type I Error:** $\mathbb{P}(f(\boldsymbol{x}_i) = 1 | y_i = 0)$, namely, the conditional probability that the predicted label is 1 given that the true label is 0.
  - **Type II Error:** $\mathbb{P}(f(\boldsymbol{x}_i) = 0 | y_i = 1)$, namely, the conditional probability that the predicted label is 0 given that the true label is 1.
- We use $R_0(f), R_1(f)$ to denote the type I,II error made by classifier f, respectively.

Minimizing the Risk

In order to find the "best" classifier, the classicial binary classification aims to minimize the weighted sum of type I,II error:

$$f^* := \arg\min_f \mathbb{P}(y=0)R_0(f) + \mathbb{P}(y=1)R_1(f)$$

where we denote $\pi_0 := \mathbb{P}(y=0), \pi_1 := \mathbb{P}(y=1)$ be some priori probabilities of the class $0, 1$.

However, this minimization is not always optimal.

Minimizing the Risk

- Consider an application in cancer diagnosis. If we let $y = 0$ be the class of healthy people, and $y = 1$ be the class of diseased people. Then the concequences of type I & II error may be different, i.e misclassify a diseased person as healthy may be more serious than misclassify a healthy person as diseased!
- Hence in some cases we are more interested in controlling one of $R_0(f), R_1(f)$ below a certain level.
- In Neyman-Pearson classification, we will manually set up a threshold α, such that we aim to minimize $R_1(f)$ under the constrain that $R_0(f) \leq \alpha$ or vise versa. This is the **NP Oracle Classifier**:

$$f^* = \text{argmin}_{f, R_0(f) \leq \alpha} R_1(f)$$

or

$$f^* = \text{argmin}_{f, R_1(f) \leq \alpha} R_0(f)$$

Neyman-Pearson Lemma

- The NP Oracle Classifier f^* can be obtained by Neyman-Pearson lemma:

$$f^*(\boldsymbol{x}) := \begin{cases} 1 & \text{if } \frac{f_1(\boldsymbol{x})}{f_0(\boldsymbol{x})} > \eta \\ 0 & \text{otherwise} \end{cases}$$

where $f_0(\boldsymbol{x}), f_1(\boldsymbol{x})$ are the density from two classes and η satisfies

$$\mathbb{P}\left(\frac{f_1(\boldsymbol{x})}{f_0(\boldsymbol{x})} > \eta \middle| y = 0 \right) = \alpha$$

- So if the density of $f_0(\boldsymbol{x}), f_1(\boldsymbol{x})$ are given, then NP-lemma will give us the optimal classifier for a given significance level α.
- We are more interested in the case when their density is not known, i.e we only know $f_j, j \in \{0, 1\}$ by a training set $\mathcal{D}_n$.
- Two classical techniques are **Empirical Risk Minimization Approach** and **Plug-in Approach**.

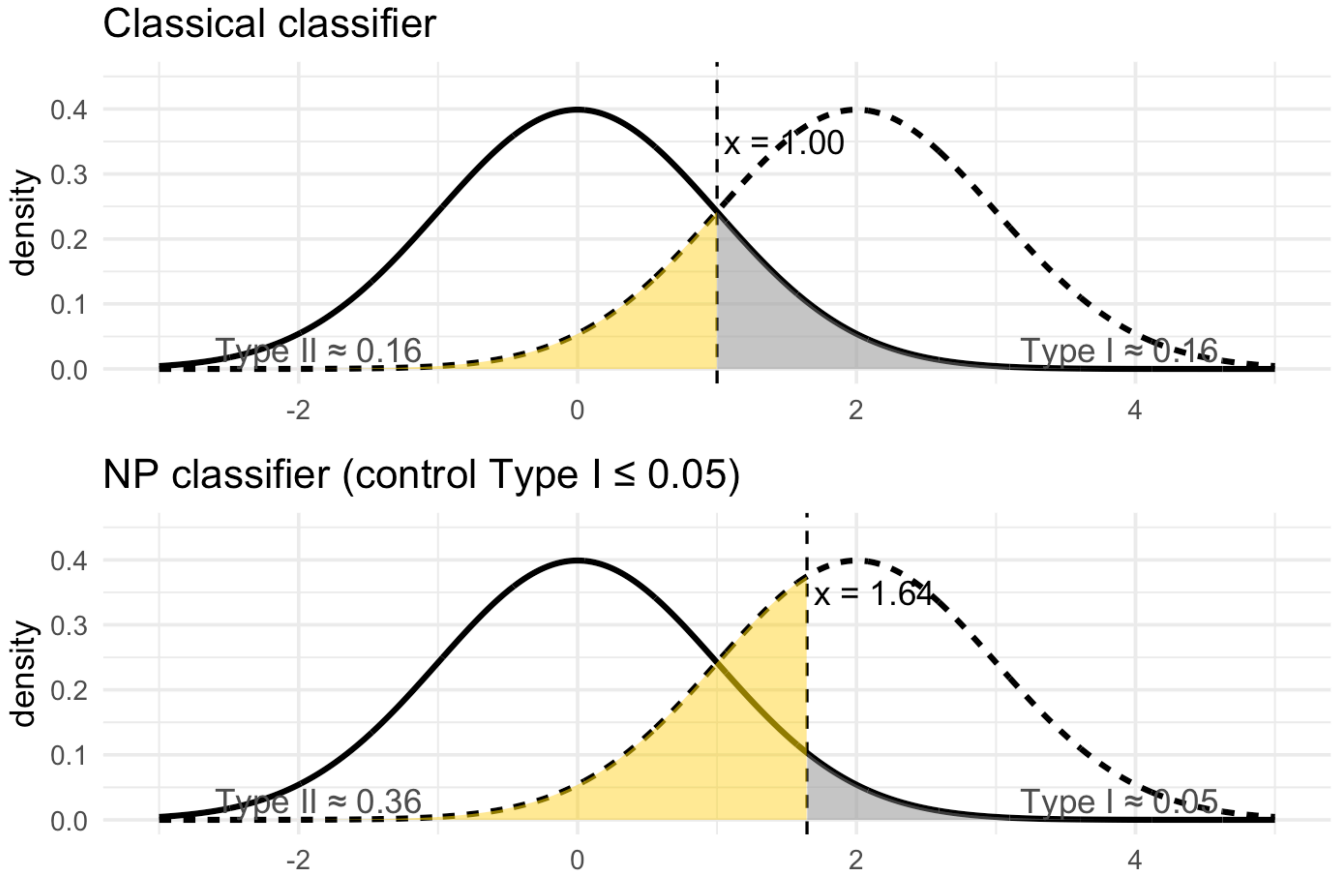

Figure: A simulation: The true distributions of $\boldsymbol{x}$ under two classes are $\mathcal{N}(0,1)$ (with $y=0$), $\mathcal{N}(2,1)$ (with $y=1$). If user prefers a type I error $R_0(f) \leq 0.05$, a classical classifier $\mathbf{1}\{x \geq 1\}$ would have $R_0(f) = 0.16$, but the NP classifier $\mathbf{1}\{x \geq 1.65\}$ will satisfy the type I error constraint while minimize type II error.

Plug-in Approach

- Using the Neyman-Pearson Lemma, we can find η from the oracle using the following formula

$$\eta_\alpha = \inf \left\{ \eta : \frac{\sum_{i=1}^{N} I(f(x_i) > \eta, y_i = 0)}{\sum_{i=1}^{N} I(y_i = 0)} \leq \alpha \right\}$$

  where $f(x)$ is a classifier that output the probability of x being in class 1.
- The problem with this approach is around 40% of the oracle will make more than α type 1 error.

Plug-in Approach

Suppose that we divide the training data into two parts, one with data from bath classes 0 and 1 for training a base algorithm to obtain f and the other a a left-out class 0 sample for choosing η. Applying the resulting f to the left-out class 0 sample of size n, we denote the resulting classification scores as $T_1, \ldots, T_n$ which ar real-valued random variables. Then we denote by $T_{(k)}$ the kth order statistic. For a new observation, if we denote its classification score based f as T, then we can construct a classifier $\hat{\phi}_k = (T > T_{(k)})$. Then, the population type I error of $\hat{\phi}_k$, denoted by $R_0(\hat{\phi}_k)$, is a function of $T_{(k)}$ and hence a random variable. Assuming the the data used to train the base algorithm and the left-out class 0 data are independent, we have

$$\mathbb{P}\left[R_0(\hat{\phi}_k) > \alpha\right] \leq \sum_{j=k}^{n} \binom{n}{j} (1-\alpha)^j \alpha^{n-j}$$

That is the probability that the type I error of $\hat{\phi}_k$ exceeds α is under a constant that only depends on k and α. We call this probability the "violation rate" of $\hat{\phi}_k$ and denote its upper bound by

Umbrella algorithm

Input:

Training data: A mixed i.i.d sample $S = S^0 \cup S^1$, where S^0 and S^1 are class 0 and 1 samples, respectively

α: Type I error upper bound $0 \leq \alpha \leq 1$

δ: A small tolerance level, $0 < \delta < 1$

M: Number of random splits on S^0

Output:

An ensemble NP classifier

$$\hat{\phi}_\alpha(X) = I\left(\frac{1}{M}\sum_{i=1}^{M}\phi_i(X) \geq \frac{1}{2}\right)$$

Umbrella algorithm

Function: RankThreshold(n, α, δ)
For k in $\{1, \ldots, n\}$ do

$$v(k) \leftarrow \sum_{j=k}^{n} \binom{n}{j} (1-\alpha)^j \alpha^{n-j}$$

$k^* \leftarrow \min\{k \in \{1, \ldots, n\} : v(k) \leq \delta\}$
Return k^*

Umbrella algorithm

Procedure NPClassifier(S, α, δ, M)

1: $n = |S^0|/2$
2: $k^* \leftarrow \text{RankThreshold}(n, \alpha, \delta)$
3: **for** $i \in \{1, \ldots, M\}$ **do**
4: $\quad S^0_{i,1}, S^0_{i,2} \leftarrow$ random split on S^0
5: $\quad S_i \leftarrow S^0_{i,1} \cup S^1$
6: $\quad f_i \leftarrow \text{ClassificationAlgorithm}(S_i)$
7: $\quad T_i = \{t_{i,1}, \ldots, t_{i,n}\} \leftarrow \{f_i(x_1), \ldots, f_i(x_n)\}$
8: $\quad \{t_{i,1}, \ldots, t_{i,n}\} \leftarrow \text{sort}(T_i)$
9: $\quad t^*_i \leftarrow t_{i,(k^*)}$
10: $\quad \phi_i(X) = I(f_i(X) > t^*_i)$
11: **end for**

Umbrella algorithm

- This approach gives good results but suffer 2 main drawbacks
- The precision of values in computers: $(1-\alpha)^j \alpha^{n-j}$
- $\binom{n}{j}$ is an overflow for double precision float when $n \geq 1030$

Tree OLNP

The goals of this algorithm are the following:

- False positive controllability: Property of any NP-classifier
- Dynamic data modeling power: The classifier must be able to adapt to higher dimension complexity as the number of data points increases
- Computational scalability: The algorithm must be able to run efficiently with respect to the number of data points

Tree OLNP

The minimization problem to find the NP Oracle classifier $f^* = \text{argmin}_{f, R_0(f) \leq \alpha} R_1(f)$ yields

$$2 \times \text{sgn}(f^*(x)) - 1 = \hat{y} = 1 \iff \frac{p_{X|Y}(x|1)}{p_{X|Y}(x|-1)} \geq \hat{\mu} \frac{\pi^-}{\pi^+}$$

$$\text{subject to } R_0(f^*) = \alpha \text{ (likelihood ratio test)} \tag{3}$$

$$\iff f^* = \arg\min_f \hat{\mu}_\alpha \pi^- R_0(f) + \pi^+ R_1(f)$$

$$\iff f^* = \arg\min_f \hat{\mu}_\alpha \frac{n_T^-}{T} \frac{1}{n_T^-} \sum_{t=1}^{T} \text{sgn}(f(x_t)) \, \text{sgn}(-y_t)$$

$$+ \frac{n_T^+}{T} \frac{1}{n_T^+} \sum_{t=1}^{T} \text{sgn}(-f(x_t)) \, \text{sgn}(y_t) \text{ (empirical risk)} \tag{5}$$

where $\hat{\mu}$ is the unknown cost of deciding 1 when in fact $y = -1$ and it is estimated such that $R_0(f^*) = \alpha$. Thus, $\hat{\mu}$ is replaced with $\hat{\mu}_\alpha$.

$$\iff f^* = \arg\min_f \frac{1}{T} \sum_{t=1}^{T} \mu_\alpha \operatorname{sgn}(-y_t f(x_t))$$

$$(\text{with } \mu_\alpha = \hat{\mu}_\alpha \text{ if } y_t = -1 \text{ and } \mu_\alpha = 1 \text{ if } y_t = 1), \tag{6}$$

To obtain a differential loss function for this optimization, we can replace $\operatorname{sgn}(-y_t f(x_t)) \in \{0, 1\}$ with the continuous valued sigmoid

$$l(-y_t f(x_t)) = \frac{1}{1 + e^{y_t f(x_t)}} \in [0, 1]$$

OLNP

We get the optimization

$$
\begin{aligned}
f^* &= \arg\min_f \frac{1}{T}\sum_{t=1}^{T} \mu_\alpha l(-y_t f(x_t)) \\
&= \arg\min_f \frac{1}{T}\sum_{t=1}^{T} s_{t+1} \\
&= \arg\min_f L(f)
\end{aligned}
$$

This optimization is solved by the SGD:

$$
f^* \leftarrow f_t = f_{t-1} - \beta_f \nabla f_{t-1} \mu_{\alpha,t-1} l(-y_t f_{t-1}(x_t)), \tag{8}
$$

where $\beta_f > 0$ is the learning rate.

OLNP

Since the type I error rate cost $\hat{\mu}_\alpha$ should be increased if $I(f(x_t))$ exceeds the user-specified desired α (and decreased otherwise), we can estimate $\hat{\mu}_\alpha$ through stochastic updates by

$$\hat{\mu}_\alpha \leftarrow \hat{\mu}_{\alpha,t} = \hat{\mu}_{\alpha,t-1} + \beta_\mu(I(f_{t-1}(x_t)) - \alpha) \text{ only when } y_t = -1, \quad (9)$$

with $\beta_\mu > 0$ and $\beta_f \gg \beta_\mu$ to ensure convergence. Note that μ_α must be initialized at 1.

OLNP

To obtain the *online linear NP classifier* (OLNP), we simply use $f(x) = w^T x$ with $w = [\tilde{w}, b]$. Then based on a data stream $\{(x_t, y_t)\}$ and the updates in (8) and (9) for estimating the parameters w, OLNP can be sequentially trained as $f_t(\cdot)$ in an online manner.
Note that OLNP has the property of false positive controllability, and employs a typical regularization by also minimizing $\lambda \frac{||f||^2}{2}$ i.e., penalizing the squared magnitude of the classifier parameters where λ is the regularization constant.

Tree OLNP

In order to have a time varying degree of nonlinearity, they use a finite ensemble of piecewise linear classifiers (ensemble of experts).
An n-piecewise linear classifier $f_{\mathcal{P}}$ utilize a partition $\mathcal{P} = \{R_1, R_2, \ldots, R_n\}$ of the observation space $\mathbb{R}^d$ and keeps a separate OLNP f_R for each partition R. The update of the type I error cost $\hat{\mu}$ in (9) is conducted based on all observations in order for all OLNPs to have access tot the same cost structure, whereas each partition region OLNP conducts the parameter update in (8) with only those instance falling in the respective region. The resulting $f_{\mathcal{P}}$ is an online n-piecewise (hence non-linear) OLNP, whose degree of nonlinearity is n.

Tree OLNP

Given an ensemble of partitions $\{\mathcal{P}_i\}_{i=1}^{N_q}$ with various corresponding pieces $\{n_i\}_{i=1}^{N_q}$, an ensemble of piecewise OLNPs $\{f_{\mathcal{P}_i}\}_{i=1}^{N_q}$ with various degrees oof nonlinearities follows. We call $f_{\mathcal{P}_i}$ and expert and denote it by f_i for simplicity. Then the randomized NP classifier is a mixture of experts in the online setting here with the subscript t: for $1 \leq i \leq N_q$

$$g_t(x) = 2 \cdot \text{sgn}(f_{i,t}(x)) - 1 \text{ with probability } q_{i,t} \tag{11}$$

Tree OLNP

At any time $t \geq 1$, the accumulated NP loss $S_{i,t}$ as well as the NP performance of the expert f_i can be measured by

$$\begin{aligned} S_{i,t} &= \sum_{t'=1}^{t} s_{i,t'} \\ &= S_{i,t-1} + s_{i,t} \end{aligned}$$

with initial $S_{i,0} = 0$ and

$$\begin{aligned} U_{i,t} &= q_{i,0} \exp(-h \cdot S_{i,t}) \\ &= U_{i,t-1} \exp(-h \cdot s_{i,t}) \end{aligned}$$

where $q_{i,0}$'s are initial probability assignments which should be chosen inversely proportional to the expert powers, and $h > 0$ is a learning rate parameter.

Tree OLNP

Using the accumulated performance, we define the dynamic probability assignments as

$$q_{i,t} = \frac{U_{i,t-1}}{\sum_{j=1}^{N_q} U_{j,t-1}}, \tag{12}$$

where $\sum_{i=1}^{N_q} q_{i,t} = 0$ and $q_{i,t} \geq 0, \forall t \geq 1$. Observe that simpler experts outweigh in the beginning of the stream and more powerful experts gradually dominate as time progresses.

Tree OLNP

Instead of having all the experts in a list, we construct a tree as follows. For A depth-D binary tree is constructed on which region from the observation space is kept and split into two, and the resulting two sub-regions are assigned to the corresponding child nodes. This process is applied to all the nodes recursively starting from the root node until the leaves are reached, with the root node keeping the complete observation space. We create a partition for each possible union of regions from the tree. We train an OLNP for each region and define an expert for each partition.

Tree OLNP

Obtaining the node decisions only is sufficient to obtain the final decision of the algorithm $g_{t-1}(x_t)$ for any observation x_t in general for a depth-D tree visiting the nodes $\{\nu = O, \nu_1, \ldots, \nu_D\}$ as

$$g_{t-1}(x_t) = 2 \cdot \mathrm{sgn}(f_{\nu_i, t-1}(x_t)) - 1 \tag{14}$$

with probability $\tilde{q}_{i,t-1}$ for $0 \leq i \leq D$, where the expert probabilities q are accumulated in $\tilde{q}$ by

$$\tilde{q}_{i,t-1} = \sum_{j=1}^{N_q} q_{j,t-1} \cdot 1_{f_{j,t-1}(x_t) = f_{\nu_i, t-1}(x_t)}, \quad 0 \leq i \leq D$$

Therefore, the binary partitioning tree-based implementation of the algorithm has computational complexity $O(D) = O(\log \log N_q)$

Tree OLNP

Let $\mathcal{U}_{\nu,t}$ be the combined performance variable that is defined for any node over the tree but updated in time for only those nodes $\nu \in \{\nu_0 = O, \nu_2, \ldots, \nu_D = leaf\}$ the observation x_t visits as

$$\mathcal{U}_{\nu,t} = \exp(-hS_{\nu,t}) = \exp(-hS_{\nu,t-1}) \cdot \exp(-hs_{\nu,t}) \tag{15}$$

if ν is a leaf and

$$\mathcal{U}_{\nu,t} = q_s \mathcal{U}_{\nu_r,t} \mathcal{U}_{\nu_l,t} + (1 - q_s) \exp(-hS_{\nu,t}), \tag{16}$$

if ν is not a leaf node. Where $s_{\nu,t}$ and $S_{\nu,t}$ are the stochastic and the accumulated NP losses of the OLNP running at the node ν and $0 \leq q_s \leq 1$ is the split probability of the tree that is related to the expert probabilities $q_{i,t}$ with $1 - q_s$ being the probability of choosing the expert of the complete observation space at time $t = 1$.

Pseudo-code Tree OLNP

1: Set the tree depth D, target false alarm rate τ, regularization λ, and the parameters β_f, β_μ and h.
2: Construct the tree and initialize.
3: **for** $t = 1, 2, \ldots$ **do**
4: Receive the observation x_t and find the visited nodes ν_i's.
5: Calculate the lumped probabilities (top-down) $\hat{q}_{i,t-1}$'s. (12)
6: Calculate the decision using $g_{t-1}(x_t)$ and observe the feedback y_t. (11)
7: Update μ_τ and obtain $\mu_{\tau,t}$. (9)
8: Calculate the instantaneous (top-down) NP loss $s_{\nu,t}$ for each visited node. (10)
9: Obtain the update node classifiers (top-down) $f_{\nu,t}$'s. (8)
10: Obtain the updated combined performance $\mathcal{U}_{\nu,t}$'s for each visited node (bottom-up).(15, 16)
11: **end for**