

Machine Learning Theory

Last modified: 2026-10-27

i Source

Adapted from Brian Hutchinson's Fall 2025 lecture notes. The section on the VC dimension paraphrases Chapter 2 of *Learning From Data* [1]; its examples and code are new.

1 Bayes Decision Theory and Bayes Error

- We set up the problem as follows
 - Our model inputs are x
 - We know the true joint distribution $P(x, y)$
Technically we only need $P(y|x)$ (or equivalently, true priors and likelihoods)
 - What is the best class y to pick for each datapoint?
- Scenario: an email inbox that receives a large batch of messages, some of them spam. We must decide whether to filter each message into the spam folder.

- Assume we know true $P(S|x)$, where x are the observable features/attributes.

For simplicity, we'll let $x \in \mathbb{R}$ be a measurement of suspiciousness (a spam score)

- Draw $P(S = y, x)$ and $P(S = n, x)$

Area under $P(S = y, x)$ is $P(S = y)$, and likewise area under $P(S = n, x)$ is $P(S = n)$

Also

$$P(S = y|x) = \frac{P(S = y, x)}{P(S = y, x) + P(S = n, x)}$$

- We filter a message if $P(S = y|x) > \theta$ for some threshold θ :

$$h(x) = \delta(P(S = y|x) > \theta)$$

where δ is the kronecker delta function:

$$\delta(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{if } z \text{ is false} \end{cases}$$

Exercise 1.1 (Two messages, one score). Two email messages have exactly the same spam score x , but one is spam and the other is not. Can any function $h(x)$ classify both of them correctly?

Solution

Solution 1.1. No. A function of x returns one prediction per value of x , so it gives both messages the same answer, and one of them must be wrong.

Exercise 1.2 (Two messages, one score (part 2)). Use the setup of Exercise 1.1. What does this say about whether some h can always be right when $\Pr(S = y \mid x)$ lies strictly between 0 and 1?

Solution

Solution 1.2. More generally, whenever $0 < \Pr(S = y \mid x) < 1$, messages with that same score include both spam and non-spam, so every h errs on some of them. The best h can do is pick the more likely label, and its error rate at that x is the probability of the other one. Only if the score determined the label, so that $\Pr(S = y \mid x)$ was always 0 or 1, could some h be right every time.

Exercise 1.3 (Slide the threshold). For the rule $h(x) = \delta(\Pr(S = y \mid x) > \theta)$, describe which messages you block when $\theta = 0$.

Solution

Solution 1.3. With $\theta = 0$, you filter every message whose probability of being spam is above 0: anything with even a slight chance of being spam.

Exercise 1.4 (Slide the threshold (part 2)). Use the setup of Exercise 1.3. Describe which messages you block when $\theta = 1$.

Solution

Solution 1.4. With $\theta = 1$, a probability can never exceed 1, so you block nothing.

Exercise 1.5 (Slide the threshold (part 3)). Use the setup of Exercise 1.3. As θ rises from 0 to 1, what happens to the set of messages you block?

Solution

Solution 1.5. Raising θ makes the condition $\Pr(S = y \mid x) > \theta$ harder to meet, so the set of filtered messages only shrinks. Each value of θ trades real messages wrongly filtered against spam wrongly delivered: a low threshold errs toward filtering too much, and a high one toward too little.

Exercise 1.6 (When a mistake costs more). Suppose wrongly filtering a real message is far more costly than letting a spam message through. Should the threshold θ sit above or below $1/2$?

Solution

Solution 1.6. Above $1/2$.

Exercise 1.7 (When a mistake costs more (part 2)). Use the setup of Exercise 1.6. Why?

Solution

Solution 1.7. At a threshold of $1/2$ you block whenever a message is more likely spam than not, which minimizes the error rate when both mistakes cost the same. If wrongly filtering a real message costs more, you should hold back unless you are more sure than that, so the threshold must rise. The more a wrongly filtered message costs relative to a missed spam message, the closer θ moves to 1.

- In general, we want to pick h given $P(y|x)$ to minimize risk.

$$\begin{aligned} R(h) &= E[L(Y, h(X))] \\ &= \int_x \underbrace{\sum_{m=1}^C L(m, h(x)) p(Y = m|x) p(x)}_{R(h|x)} dx \end{aligned}$$

We minimize this by picking the class that minimizes the *conditional risk*:

$$\begin{aligned} h(x) &:= \arg \min_c E_{Y|x}[L(Y, c)] \\ &= \arg \min_c \sum_{m=1}^C L(m, c) p(Y = m|x) \end{aligned}$$

This is known as *Bayes decision rule*.

- Even with this rule, the risk is usually not 0. The minimum risk is called *Bayes risk*:

$$R^* = \min_h R(h)$$

No rule can have lower risk than this.

- Example loss:

$$L(S = y, h(x) = y) = 0, \quad L(S = y, h(x) = n) = 1, \quad L(S = n, h(x) = y) = 5, \quad L(S = n, h(x) = n) = 0$$

We should filter a message if

$$\overbrace{L(S = y, h(x) = y)}^0 P(S = y|x) + \overbrace{L(S = n, h(x) = y)}^5 P(S = n|x) < \underbrace{L(S = y, h(x) = n)}_1 P(S = y|x) + \underbrace{L(S = n, h(x) = n)}_0 P(S = n|x)$$

aka if

$$5P(S = n|x) < P(S = y|x) \quad \text{aka} \quad \frac{P(S = y|x)}{P(S = n|x)} > 5$$

We filter a message only if it is more than 5 times as likely to be spam as not.

- A common goal is to minimize the error rate. We can accomplish this with the zero-one loss.

$$L(y, h(x)) = 1 - \delta(y = h(x))$$

The expected conditional loss is

$$\begin{aligned} E_{Y|x}[L(Y, h(x))] &= \sum_{c=1}^M (1 - \delta(c = h(x))) P(Y = c|x) \\ &= \sum_{c \neq h(x)} P(Y = c|x) \\ &= 1 - P(Y = h(x)|x) \end{aligned}$$

so we want to pick

$$\begin{aligned} h(x) &:= \arg \min_c E_{Y|x}[L(Y, c)] \\ &= \arg \max_c P(Y = c|x). \end{aligned}$$

(We have used this rule before, and it is intuitive.) For a binary task, this simplifies to

$$h(x) = \delta(P(y = 1|x) > \underbrace{0.5}_{\theta})$$

- The Bayes error (the classification counterpart of the residual variance) is the Bayes risk with zero-one loss:

$$E^* = \min_h R(h)$$

- How does the rule change if we know only estimates of the true distributions?
 - Even if we apply our rule to the estimates, our decision will be suboptimal
 - A suboptimal decision boundary will also have *reducible error*, the excess over the Bayes error
- We can never do better than Bayes error... or can we? What allows us to reduce the Bayes error?

Add more features. The Bayes error, like the residual variance, is irreducible only for a given set of inputs. Example: x_2 is presence of some hormone

- When can we use this?
 - With discriminative models that give us $P(y|x)$
 - With generative models that give us $P(y, x)$

$$\begin{aligned} P(y|x) &= \frac{\overbrace{P(x, y)}^{joint}}{\sum_y \overbrace{P(x, y)}^{joint}} \\ &= \frac{\overbrace{P(x, y)}^{joint}}{\underbrace{P(x)}_{evidence}} \end{aligned}$$

- With generative models that give us $P(x|y)$ (e.g., a Gaussian mixture model, GMM) and class priors $P(y)$

$$\begin{aligned} P(y|x) &= \frac{\overbrace{P(x|y)}^{likelihood} \overbrace{P(y)}^{prior}}{\underbrace{P(x)}_{evidence}} \\ &= \frac{P(x|y)P(y)}{\sum_{y'} P(x|y')P(y')} \end{aligned}$$

i Watch

von Luxburg's lectures cover:

- Bayesian decision theory^a [2, Pt. 4]
- the Bayes classifier^b [2, Pt. 5]

^a<https://www.youtube.com/watch?v=XN9A6lU7Xqg>

^bhttps://www.youtube.com/watch?v=u4LBC_RI3Ho

2 Bias and Variance, Revisited

- Recall that bias and variance are two ways that your model can mismatch the true target function
- For regression with squared error, the expected¹ error over all training sets splits into squared bias, variance and residual variance, as the bias–variance decomposition shows
 - Low bias if $h_{\mathcal{X}}(x)$ approximates $F(x)$ well on average
 - Low variance if $h_{\mathcal{X}}(x)$ does not change much depending on the training set
 - As $n \rightarrow \infty$, $h_{\mathcal{X}}(x)$ is less variable
 - Best way to reduce both is to know something about $F(x)$
 - Zero bias and variance means you always output the correct answerNothing to learn
- Classification is more difficult to analyze
 - Can think of it as estimating the decision boundary/boundaries
 - Lower “bias” in this case means closer to the Bayes’ optimal boundary
 - Lower variance means less variance in our decision boundary estimates
 - For classification, variance dominates biasSimple classifiers are more important (hence the emphasis on linear classifiers)

i Watch

- Abu-Mostafa decomposes the out-of-sample error into bias and variance, from 8:52^a [3], Lecture 8.
- Starmer explains bias and variance^b [4].

^a<https://www.youtube.com/watch?v=zrEyxfl2-a8&t=532s>

^b<https://www.youtube.com/watch?v=EuBBz3bI-aA>

i Read

GeeksforGeeks explains the bias–variance tradeoff^a [5].

^a<https://www.geeksforgeeks.org/machine-learning/ml-bias-variance-trade-off/>

3 Curse of Dimensionality

- Many forms
- Very difficult to learn local properties when p becomes large
- Intuition based on low dimensions fails in high dimensions

Example 3.1 (How long an edge captures a fraction of the data). Suppose datapoints are uniformly distributed in a p -dimensional unit hypercube. A smaller cube that captures a fraction r of the volume, and so of the datapoints on average, has edge length $r^{1/p}$, because its volume is its edge length raised to the power p . Table 1 computes this edge length for two fractions, $r = 0.1$ and $r = 0.01$, and four values of p .

¹<https://morrison-lab.github.io/pds/expectation.html#def-expectation>

```

# One row per pair of dimension D and volume fraction r.
edge_tbl <- expand_grid(r = c(0.1, 0.01), D = c(3, 10, 100, 1e5))
# The edge length that gives volume r in D dimensions solves edge^D = r.
edge_tbl$edge <- edge_tbl$r^(1 / edge_tbl$D)
# Print D with thousands separators, since scientific notation is hard to read
# in a table.
edge_tbl$D <- format(
  edge_tbl$D, big.mark = ",", scientific = FALSE, trim = TRUE
)
edge_tbl <- edge_tbl[, c("D", "r", "edge")]
# Show the edge to five decimals, since the values crowd against 1 as D grows.
knitr::kable(edge_tbl, digits = 5, align = "rrr",
  col.names = c("$\\ninputvars$", "$r$", "$r^{1/\\ninputvars}$"))

```

Table 1: Edge length of the sub-cube that captures a fraction r of the unit hypercube’s volume.

p	r	$r^{1/p}$
3	0.10	0.46416
3	0.01	0.21544
10	0.10	0.79433
10	0.01	0.63096
100	0.10	0.97724
100	0.01	0.95499
100,000	0.10	0.99998
100,000	0.01	0.99995

To capture 10% of the volume in $p = 3$ dimensions, the sub-cube needs an edge of 0.464, about 46% of the way along each axis. In $p = 100$ dimensions it needs an edge of 0.977, which stretches across 98% of every axis. A “local” neighborhood that holds even a tenth of the data therefore spans nearly the whole range of every feature.

Example 3.2 (How far the nearest datapoint is from the origin). Suppose N datapoints are sampled uniformly from the unit hyperball in p dimensions. A ball of radius r about the origin has a fraction r^p of the hyperball’s volume, so a given point misses it with probability $1 - r^p$, and all N points miss it with probability $(1 - r^p)^N$. The median distance from the origin to the closest point is the radius at which this probability is $\frac{1}{2}$:

$$(1 - r^p)^N = \frac{1}{2} \iff r = (1 - (\frac{1}{2})^{1/N})^{1/p}.$$

Table 2 computes this median distance for two sample sizes and four values of p .

```
# One row per pair of dimension D and sample size N.
dist_tbl <- expand.grid(N = c(100, 10000), D = c(3, 10, 100, 1e5))
# Solve (1 - r^D)^N = 1/2 for the radius r, as in the formula above.
dist_tbl$dist <- (1 - 0.5^(1 / dist_tbl$N))^(1 / dist_tbl$D)
# Print D and N with thousands separators, since scientific notation is hard to
# read in a table.
dist_tbl$D <- format(
  dist_tbl$D, big.mark = ",", scientific = FALSE, trim = TRUE
)
dist_tbl$N <- format(
  dist_tbl$N, big.mark = ",", scientific = FALSE, trim = TRUE
)
dist_tbl <- dist_tbl[, c("D", "N", "dist")]
# Show the distance to five decimals, since the values crowd against 1 as D
# grows.
knitr::kable(dist_tbl, digits = 5, align = "rrr",
  col.names = c("$\\mathrm{ninputvars}$", "$N$", "median distance"))
```

Table 2: Median distance from the origin to the closest of N points sampled uniformly from the unit hyperball.

p	N	median distance
3	100	0.19045
3	10,000	0.04108
10	100	0.60804
10	10,000	0.38378
100	100	0.95147
100	10,000	0.90867
100,000	100	0.99995
100,000	10,000	0.99990

In $p = 3$ dimensions with $N = 10,000$ points, the closest point is typically only 0.041 from the origin. In $p = 100$ dimensions with the same sample the closest point is typically 0.909 away, nearly at the boundary of the unit ball. In high dimensions almost all datapoints are near the boundary, and making predictions at the boundary is harder.

- Complexity of problems can often grow exponentially with p
- To maintain the same sampling density, you need N^p points

All high dimensional problems are sparse

- Any method that must produce locally varying functions in high dimensions will perform poorly
- To handle this, we need a simple model, so we assume

$$y = w^T x + b + \varepsilon, \quad \text{where } \varepsilon \sim N(0, \sigma^2)$$

then our expected² Risk (over datasets) for a linear model is

$$\sigma^2(p/N) + \sigma^2$$

The risk grows with p and σ^2 and shrinks with N .

- One other common strategy to handle high dimensional data is to map it to a lower dimensional space, also called “dimensionality reduction”. Often we do this by finding a linear subspace that keeps the

²<https://morrison-lab.github.io/pds/expectation.html#def-expectation>

most important information in the data.

$$x \in \mathbb{R}^p \xrightarrow[f(x)]{} \tilde{x} \in \mathbb{R}^{p'} \xrightarrow[h(\tilde{x})]{} y, \quad \text{where } p \gg p'$$

where $f(x)$ is linear and $h(\tilde{x})$ is nonlinear.

- **Random Projection.** Generate a random $p \times p'$ matrix W , drawing each element from a Gaussian distribution³. Then $f(x) = W^T x$.

Draw oblong point cloud and show the points being projected a random line.

- **Principal Components Analysis (PCA):** $f(x)$ maps x to the lower dimensional space that maximizes variance (preserves as much of the original variance as possible)

Draw oblong point cloud and show the line that maximizes variance

- **Linear Discriminant Analysis (LDA):** $f(x)$ maps x to the lower dimensional space that maximizes the separation between classes (for classification tasks)

Draw two oblong parallel point clouds that are close together and show the line that maximizes separation, compared to the PCA line.

- **Others.** We can interpret the input-to-hidden layer of a neural network as doing this.

Watch

Weinberger's lecture on the curse of dimensionality^a [6], Lecture 4.

^a<https://www.youtube.com/watch?v=BbYV8UfMJSA>

Read

GeeksforGeeks explains the curse of dimensionality^a [7].

^a<https://www.geeksforgeeks.org/machine-learning/curse-of-dimensionality-in-machine-learning/>

4 No Free Lunch Theorem

- Is there a learning algorithm that is better than all others? The answer is not simple.
- Inductive Bias
 - The *inductive bias* of a learning algorithm is the set of assumptions it makes about the target function that restrict or prioritize models h during training.
 - Let A_i be a learning algorithm. For example, A_1 might be a support vector machine (SVM) with a particular set of hyperparameters, A_2 might be K-nearest neighbors (KNN) with $k = 5$, and A_3 might be a single hidden layer neural network with particular hyperparameters. In general, A_i is non-deterministic, so its output is random:

$$h_i \sim A_i(\mathcal{X})$$

where h_i is a random variable representing the algorithm's output. If A_i is deterministic, it always returns the same h_i given the same $\mathcal{X}$.

- In each case, h_i uses the finite training data to predict for new inputs. It can answer questions about x that it has not seen before. *Models with different inductive biases will predict differently for new inputs.*

³<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

- So which A_i have the best inductive biases? It depends on the true target function F , so we turn to the space of problems.
- The space of problems.
 - We can either think of the training data pairs (x, y) being generated by some $y = F(x) + \varepsilon$ where ε is some noise, or we can think of the training data pairs (x, y) being generated by some $P(x, y)$.
 - We use $P(x, y)$.
 - Draw a box labeled $\mathcal{P}$ that contains all possible $P(x, y)$.
 - * This is the space of all problems we might want to solve.
 - * Draw a point and label it as handwritten digits
 - * Draw a point and label it as cat/dog classification
 - * Draw a point and label it as relating spam scores to spam
 - * Draw a region and label it as the space of protein prediction problems
 - Imagine the following theoretical experiment for a given algorithm A_i :
 - * Sample a problem from $\mathcal{P}$ by drawing $P \sim \mathcal{P}$
 - * Sample a training set $\mathcal{X}$ from P by drawing $\mathcal{X} \sim P$
 - * Sample a model h_i from A_i by drawing $h_i \sim A_i(\mathcal{X})$
 - * Compute the risk of h_i on P .
 - If you repeat that experiment indefinitely for a given loss, is there an A_i that will have the lowest risk on average?

- **The No Free Lunch Theorem**

- The No Free Lunch (NFL) theorem has many forms and implications [8]
- We will look at one form, suited to the notation we have been using in these notes
- First, we update our risk notation to include the algorithm, training set and problem P :

$$R(A_i|\mathcal{X}, P) = E_{h_i \sim A_i(\mathcal{X})}[E_{(X,Y) \sim P}[L(Y, h_i(X))]]$$

In words: the expected loss of algorithm A_i on problem P given training set $\mathcal{X}$.

- If we randomly sample a problem $P \sim \mathcal{P}$ and a training set $\mathcal{X} \sim P$ (i.e., the experiment we just described), then the NFL theorem says

$$E_{P \sim \mathcal{P}}[E_{\mathcal{X} \sim P}[R(A_i|\mathcal{X}, P)]] = E_{P \sim \mathcal{P}}[E_{\mathcal{X} \sim P}[R(A_j|\mathcal{X}, P)]] \text{ for all } i, j$$

In words: no, there is no algorithm that is better than all others on average across all problems and the training sets they would produce.

- There are other, arguably stronger forms. Their assumptions seem unrealistic to us.
- This does *not* mean it doesn't matter which algorithm you use for machine learning!
- This certainly does not mean all h_i are equally good (we recently studied Bayes' decision rule).
- So what does this mean? What do we do with this information?
 - We start by rearranging the NFL finding to:

$$E_{P \sim \mathcal{P}}[E_{\mathcal{X} \sim P}[R(A_i|\mathcal{X}, P) - R(A_j|\mathcal{X}, P)]] = 0 \text{ for all } i, j$$

- This allows for $R(A_i|\mathcal{X}, P) - R(A_j|\mathcal{X}, P)$ to be positive or negative for *given* P and $\mathcal{X}$
- They cancel only *on average* across problems

- Draw the space $\mathcal{P}$ with large and small plusses and minuses, indicating areas where a given algorithm does better or worse. All the plusses and minuses average to zero, but can appear in very different ways.
 - * A given algorithm might be extraordinarily good on a certain class of problems, but very poor on others
 - * A given algorithm may be consistently average across all problems
 - Different A_i have different inductive biases that make them better or worse at different problems
 - How do we know which algorithm to use? Either
 - * Try them “all” (as many as possible) and see which one works best
 - * Use domain knowledge to pick an algorithm that is likely to work well for your problem (because you know its inductive bias works well for related problems)
- You presumably used a combination of these for the project. The more you use machine learning, the more helpful the second approach will be.

Watch

Abu-Mostafa asks whether learning is feasible^a [3], Lecture 2.

^a<https://www.youtube.com/watch?v=MEG35RDD7RA>

Watch

Abu-Mostafa defines the Vapnik–Chervonenkis (VC) dimension, from 5:15^a [3], Lecture 7.

^a<https://www.youtube.com/watch?v=Dc0sr0kdBVI&t=315s>

Watch

The CS229 discussion section on learning theory^a [9].

^a<https://www.youtube.com/watch?v=iVOxMcumR4A>

Read

GeeksforGeeks explains the no free lunch theorem^a [10].

^a<https://www.geeksforgeeks.org/machine-learning/what-is-no-free-lunch-theorem/>

5 Ugly Duckling Theorem

- So far we have mostly assumed that data points will be represented by feature vectors.
 - At some point we chose how many and which features to use
 - This raises the question: what features *should* we use?

Exercise 5.1 (List features in one minute). Set a one-minute timer and write down every feature of a house that might help predict its sale price. Then count your list.

Solution

Solution 5.1. There is no correct list, and the count depends on how finely you slice. Total floor area, the area of each room, the number of rooms and the ratio of kitchen to living area are four features built from the same facts about the house. Any list you write is a finite sample from a much larger set of possible features, and which ones you chose reflects what you already believed would matter.

- You may consider some of these features to be “better” than others, but the **Ugly Duckling Theorem** argues that no feature representation is *inherently* better than another (i.e., in the absence of assumptions about the true problem P) [11].
- The basic idea is that
 - Your model will need to use what points in the input space have in common to make predictions, either mapping “like” (similar) points to the same class or to similar outputs in general.
 - “Similarity” must depend in some way on the values of the features you use
 - Via our choice of features, we can make any two points arbitrarily similar or dissimilar
- **Example.** Define “similarity” as the number of features in common, and use only binary features.
 - Let’s pick two items: a desk and the Democratic Republic of the Congo.
- We can make these things arbitrarily similar by introducing an infinite number of things they have in common:
 - Neither existed 7.2 billion years ago
 - Neither is named Tanya
 - Neither is a duck
 - Neither serves on the board of ByteDance, creator of TikTok
 - Neither appears in the Disney movie Aladdin
 - Etc.
- We can make these things arbitrarily dissimilar by introducing an infinite number of things they do not have in common:
 - Only one can fit into a classroom
 - Only one has a population greater than 100 million
 - We have seen only one of them firsthand
 - Only one is home to the world’s deepest river
 - Only one is a member state of the United Nations
 - Etc.
- Infinite number of features in common
- Infinite number of features that differ
- So without assumptions about the problem P , is it better for these points to be similar or dissimilar? We can’t say.
- Why “Ugly Duckling”? By choosing features, we can make a duckling arbitrarily similar to a swan.
- We conclude the following.
 - **There is no such thing as a universally good feature representation.**
 - A given feature representation will help some tasks and hurt in others (both in theory, i.e., Bayes’ risk, and in practice, i.e., how well they match the inductive biases of a given learning algorithm)

- The feature that would minimize Bayes’ risk is the output y itself.

6 Generalization bounds and the VC dimension

Hoeffding’s inequality bounds the gap between the held-out error and the true error of **one** classifier fixed before the data were drawn. Training breaks that condition, because the learned classifier $\hat{f}$ is chosen from a prediction function family⁴ $\mathcal{Y}$ after looking at the training set. If $\mathcal{Y}$ has M classifiers, the union bound says some classifier in $\mathcal{Y}$ has a training error far from its true error with probability at most M times the one-classifier bound, $2M\exp\{-2\varepsilon^2 n\}$, and so does the one we pick. For a family with infinitely many classifiers, such as every line in the plane, that bound says nothing.

The union bound is wasteful, though. Two lines that differ slightly misclassify almost the same points, so their training errors move together, and the events “this line’s training error is far off” overlap heavily. What matters for the training error is not how many classifiers $\mathcal{Y}$ has, but how many *different ways it can label the n training points*. That count is finite, at most 2^n , even when $\mathcal{Y}$ is infinite.

Definition 6.1 (Shattering). A family $\mathcal{Y}$ of classifiers with two possible labels **shatters** a set of n points when classifiers in $\mathcal{Y}$ can produce all 2^n labelings of those points [12].

Example 6.1 (Shattering one point and two points). Take inputs on a line. A *positive ray* labels a point $+1$ if it lies to the right of a threshold and -1 otherwise, and a *positive interval* labels a point $+1$ if it lies between two thresholds. A single point has two labelings, $+1$ and -1 , and a ray can produce both by putting its threshold on either side of the point. So the rays shatter any one point. Two points have four labelings, and a ray cannot label the left point $+1$ and the right point -1 , so the rays do not shatter them. An interval can produce all four labelings of two points, so the intervals shatter them.

Definition 6.2 (Growth function). For a family $\mathcal{Y}$ of classifiers with two possible labels, the **growth function** $m_{\mathcal{Y}}(n)$ is the largest number of different labelings of n points that classifiers in $\mathcal{Y}$ can produce, where the largest is taken over all choices of the n points. It never exceeds 2^n [13].

Definition 6.3 (VC dimension). The **Vapnik–Chervonenkis dimension** $d_{\text{VC}}(\mathcal{Y})$ of a family $\mathcal{Y}$ is the largest n for which $m_{\mathcal{Y}}(n) = 2^n$, that is, the size of the largest set of points that $\mathcal{Y}$ shatters [14]. If $\mathcal{Y}$ shatters sets of every size, $d_{\text{VC}}(\mathcal{Y}) = \infty$.

Example 6.2 (Counting the labelings of rays and intervals). Take inputs on a line. A *positive ray* labels a point $+1$ if it lies to the right of a threshold a and -1 otherwise. A *positive interval* labels a point $+1$ if it lies between two thresholds $a < b$. For n points in increasing order, a threshold only matters through which gap between points it falls in, so trying one threshold per gap, and all pairs of gaps for intervals, finds every labeling:

6.1 Python

⁴<https://morrison-lab.github.io/sds/model-validation.html#def-prediction-function-family>

```
import numpy as np

def labelings_gf(n, kind):
    # Points 0, 1, ..., n - 1 on a line. A threshold at k - 0.5
    # splits the points into those below k and those from k up, for k = 0..n.
    points = np.arange(n)
    found = set()
    for a in np.arange(n + 1) - 0.5:
        if kind == "ray":
            found.add(tuple(points > a))
        else:
            for b in np.arange(n + 1) - 0.5:
                found.add(tuple((points > a) & (points < b)))
    return len(found)

counts_gf = [(n, labelings_gf(n, "ray"), labelings_gf(n, "interval"), 2 ** n) for n in range(1, 7)]
print("columns: n, rays, intervals, all labelings")

columns: n, rays, intervals, all labelings
print(np.array(counts_gf))

[[ 1  2  2  2]
 [ 2  3  4  4]
 [ 3  4  7  8]
 [ 4  5 11 16]
 [ 5  6 16 32]
 [ 6  7 22 64]]
```

6.2 R

```
labelings_gf <- function(n, kind) {
    # Points 1, ..., n on a line. A threshold at k + 0.5 splits the points
    # into those up to k and those after it, for k = 0..n.
    points <- seq_len(n)
    cuts <- 0:n + 0.5
    grid <- if (kind == "ray") data.frame(a = cuts, b = Inf) else expand.grid(a = cuts, b = cuts)
    labels <- apply(grid, 1, function(ab) paste(points > ab[["a"]] & points < ab[["b"]], collapse = ""))
    length(unique(labels))
}

counts_gf <- sapply(1:6, function(n) c(ray = labelings_gf(n, "ray"),
                                       interval = labelings_gf(n, "interval"), all = 2^n))
data.frame(n = 1:6, t(counts_gf))

# A tibble: 6 x 4
      n    ray interval    all
<int> <dbl>   <dbl> <dbl>
1     1     2     2     2
2     2     3     4     4
3     3     4     7     8
4     4     5    11    16
5     5     6    16    32
6     6     7    22    64
```

Rays give $n + 1$ labelings and intervals give $\binom{n+1}{2} + 1$, which grow like a power of n , not like 2^n . Rays shatter one point, with 2 of 2 labelings, but not two, with 3 of 4: no ray labels the left point +1 and the right one -1. Intervals shatter two points, with 4 of 4 labelings, but not three, with 7 of 8: no

interval labels the two outer points +1 and the middle one −1. So the VC dimension of rays is 1 and of intervals is 2 (Definition 6.3 above).

Theorem 6.1 (The VC generalization bound). *Let $\mathcal{Y}$ be a family of classifiers with two labels and finite VC dimension d_{VC} .*

1. *For every n ,*

$$m_{\mathcal{Y}}(n) \leq \sum_{i=0}^{d_{\text{VC}}} \binom{n}{i} \leq n^{d_{\text{VC}}} + 1. \quad (1)$$

2. *Let $\hat{f}_{\mathcal{Y}}$ be the classifier in $\mathcal{Y}$ chosen using a training set of n independent draws from $P(x, y)$. For every $\delta > 0$, with probability at least $1 - \delta$ over the draw of the training set,*

$$R(\hat{f}_{\mathcal{Y}}) \leq \hat{R}(\hat{f}_{\mathcal{Y}}) + \sqrt{\frac{8}{n} \ln \frac{4m_{\mathcal{Y}}(2n)}{\delta}}. \quad (2)$$

Here $\hat{R}(\hat{f}_{\mathcal{Y}})$ is the training error of $\hat{f}_{\mathcal{Y}}$ under the misclassification loss.

i Proof

Proof. We state this without proof. Part 1 is the Sauer–Shelah lemma [15], [16]. Part 2 is the Vapnik–Chervonenkis inequality [17], [18], in the form given by Abu-Mostafa et al. [1, Ch. 2] and in the Caltech CS 156 slides for lectures 6 and 7 [3]. $\square$

i Remark (Reading the VC bound)

Remark 6.1 (Reading the VC bound). By Equation 1, a family with finite VC dimension has a growth function that grows only polynomially, so the square root in Equation 2 shrinks to zero as n grows, and more data always tightens the guarantee. A family with $d_{\text{VC}} = \infty$ gives no such guarantee. A larger d_{VC} raises the penalty term, which is the formal version of the trade-off between flexibility and variance. The bound is loose, since it holds for every distribution and every learning algorithm. In practice the number of training points needed grows roughly in proportion to d_{VC} , and a common rule of thumb asks for around ten times d_{VC} [1, Ch. 2]. For a linear classifier on p inputs^a with an intercept, $d_{\text{VC}} = p + 1$, which equals its number of parameters. That match does not hold in general: a family can have one parameter and infinite VC dimension.

^a<https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

Exercise 6.1 (Can a line shatter four points?). A linear classifier on two inputs labels a point +1 on one side of a line and −1 on the other. Three points in general position can be given all $2^3 = 8$ labelings by some line.

Four points are the corners of a square. Name a labeling of the four that no line produces.

Solution

Solution 6.1. Label two opposite corners +1 and the other two −1. The segment joining the +1 corners crosses the segment joining the −1 corners at the square’s center. A half-plane is convex^a, so a line that puts both +1 corners on one side puts the whole segment between them on that side, and the same holds for the −1 corners on the other side. The center would then lie on both sides, which is impossible.

^a<https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

Exercise 6.2 (Can a line shatter four points? (part 2)). Use the setup of Exercise 6.1. What does this say about the VC dimension of lines in the plane, given that three points can be shattered?

Solution

Solution 6.2. Three points can be shattered, so $d_{VC} \geq 3$. The square shows that this arrangement of four points cannot be. For any other four points, either one lies inside the triangle of the other three, and labeling the three +1 and the one inside -1 fails by the same convexity argument, or they form a convex^a quadrilateral whose diagonals^b cross, and the square's labeling fails. (Four points on one line need a separate check, which also fails.) So no four points are shattered and $d_{VC} = 3 = p + 1$ for $p = 2$ inputs^c, as Remark 6.1 states.

^a<https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

^b<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

^c<https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

Exercise 6.3 (How surprising is a perfect fit?). Ten points on a line receive labels ± 1 by independent coin flips, so the labels carry no pattern at all. Using Example 6.2, find the largest possible chance that a positive ray fits all ten labels perfectly.

Solution

Solution 6.3. A family can fit a labeling of the ten points only if that labeling is one of the labelings it produces, and there are 2^{10} equally likely labelings. So the chance is at most $m_y(10)/2^{10}$:

```
n_fl <- 10
# The growth function of rays is n + 1.
chance_ray <- (n_fl + 1) / 2^n_fl
round(chance_ray, 3)
```

```
[1] 0.011
```

A ray fits random labels about 1.1% of the time.

Exercise 6.4 (How surprising is a perfect fit? (part 2)). Use the setup of Exercise 6.3. Find the largest possible chance that a positive interval fits all ten labels perfectly, using Example 6.2.

Solution

Solution 6.4.

```
# The growth function of intervals is choose(n + 1, 2) + 1.
chance_interval <- (choose(n_fl + 1, 2) + 1) / 2^n_fl
round(chance_interval, 3)
```

```
[1] 0.055
```

An interval fits random labels about 5.5% of the time.

Exercise 6.5 (How surprising is a perfect fit? (part 3)). Use the setup of Exercise 6.3. What does a perfect fit then tell you, for each family?

Solution

Solution 6.5. A family that shatters the ten points would fit them every time, so its fit says nothing. Hence a perfect fit by a ray is unlikely to be luck, so it is evidence of a real pattern, and the same fit by

an interval is about 5 times less convincing. This is the argument behind *Occam's razor* in machine learning [1, Ch. 5]: a simpler family, one with fewer labelings, is less likely to fit by chance, so its fit is more significant. The claim is about reliability on new data, not elegance, and it holds only if the family was fixed before looking at the labels. Choosing the family after looking at the labels is a form of data snooping.

7 Further reading

- Duda et al. [19], Chapter 2, on Bayesian decision theory, and Chapter 9, on the no free lunch and ugly duckling theorems
 - Abu-Mostafa et al. [1], Chapter 2, on the growth function, the VC dimension and the generalization bound
 - Beyer et al. [20] — nearest neighbours in high dimensions
- [1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data: A short course*. AMLBook, 2012. Available: <https://amlbook.com/>
 - [2] U. von Luxburg, “Statistical machine learning.” University of Tübingen, lecture videos on YouTube, 2020. Available: <https://www.youtube.com/playlist?list=PL05umP7R6ij2XCvrRzLokX6EoHWaGA2cC>
 - [3] Y. S. Abu-Mostafa, “Learning from data: CS 156 lectures.” Caltech, lecture videos on YouTube, 2012. Available: <https://www.youtube.com/playlist?list=PLD63A284B7615313A>
 - [4] J. Starmer, “Machine learning fundamentals: Bias and variance.” StatQuest, video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=EuBBz3bI-aA>
 - [5] GeeksforGeeks, “Bias-variance trade off - machine learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/ml-bias-variance-trade-off/>
 - [6] K. Weinberger, “CS4780: Machine learning for intelligent systems, spring 2017.” Cornell University, lecture videos on YouTube, 2017. Available: <https://www.youtube.com/playlist?list=PLI8OIHZGYOQ7bkVbuRthEsaLr7bONzbXS>
 - [7] GeeksforGeeks, “Curse of dimensionality in machine learning.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/curse-of-dimensionality-in-machine-learning/>
 - [8] Wikipedia contributors, “No free lunch theorem — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/No_free_lunch_theorem
 - [9] A. Ng *et al.*, “CS229: Machine learning, autumn 2018.” Stanford Online, lecture videos on YouTube, 2018. Available: <https://www.youtube.com/playlist?list=PLoROMvovdv4rMiGQp3WXShtMGgzqpfVfbU>
 - [10] GeeksforGeeks, “What is no free lunch theorem.” GeeksforGeeks, tutorial article, 2021. Available: <https://www.geeksforgeeks.org/machine-learning/what-is-no-free-lunch-theorem/>
 - [11] Wikipedia contributors, “Ugly duckling theorem — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Ugly_duckling_theorem
 - [12] Wikipedia contributors, “Shattered set — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Shattered_set
 - [13] Wikipedia contributors, “Growth function — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Growth_function
 - [14] Wikipedia contributors, “Vapnik–chervonenkis dimension — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Vapnik%E2%80%93Chervonenkis_dimension
 - [15] N. Sauer, “On the density of families of sets,” *Journal of Combinatorial Theory, Series A*, vol. 13, no. 1, pp. 145–147, 1972, doi: 10.1016/0097-3165(72)90019-2⁵.
 - [16] Wikipedia contributors, “Sauer–shelah lemma — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Sauer%E2%80%93Shelah_lemma

⁵[https://doi.org/10.1016/0097-3165\(72\)90019-2](https://doi.org/10.1016/0097-3165(72)90019-2)

- [17] V. N. Vapnik and A. Ya. Chervonenkis, “On the uniform convergence of relative frequencies of events to their probabilities,” *Theory of Probability and Its Applications*, vol. 16, no. 2, pp. 264–280, 1971, doi: 10.1137/1116025⁶.
- [18] Wikipedia contributors, “Vapnik–chervonenkis theory — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Vapnik%E2%80%93Chervonenkis_theory
- [19] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. New York: Wiley, 2001.
- [20] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ‘nearest neighbor’ meaningful?” in *Database theory — ICDT’99*, in Lecture notes in computer science, vol. 1540. Springer, 1999, pp. 217–235. doi: 10.1007/3-540-49257-7_15⁷.

⁶<https://doi.org/10.1137/1116025>

⁷https://doi.org/10.1007/3-540-49257-7_15