BIAS + VARIANCE: Classical + Modern Elements

+ Classical theory
  - Regularization
  - Parameter selection
+ Modern theory (Bonus)

Compute Efficient: Stochastic Policy

Data: $K$-Fold
TRUE function $f_0(x) = \alpha x^2 + \beta, x + \epsilon$

We don't get to see \( \mathcal{A} \) directly, only samples from it!

WHAT IF WE FIT A LINE?

Informally, we call this **underfit.** The error is mostly by \( \text{(high bias)} \)

How about degree 5 polynomial?

**overfit**

fits the data well but changes on each sample.

Recall \( \mathcal{A} \) is quadratic, what if we use that model class?

\( \rightarrow \) low error \& low variance

\( \rightarrow \) it fits!
UNDERFIT: TRAIN loss ≥ TEST loss. But Error is high
OVERFIT: TRAIN loss < TEST loss. But Error maybe low

This is classical BIAS-VARIANCE
+ Helpful to understand ML ideas
+ Incomplete for modern models (more done)
MORE FORMAL BIAS-VARIANCE TRADEOFF

Features: \(X \rightarrow \text{Gaussian noise} \)

\[
Y_f = h_{\theta}(x) + \epsilon \quad \epsilon \sim N(0, \sigma^2)
\]

Output \(Y \in \mathbb{R} \theta, \theta \in \mathbb{R}^d \)

**Procedure**

1. **Pick an \(x \in \mathbb{R}^d\), A TEST POINT (REASON PAST TEST DATA)**
2. **Draw n points \((x^{(i)}, y^{(i)})\), \(i = 1, 2, \ldots, n\) call this \(S\)
3. \(\theta_i = h_{\theta}(x^{(i)}) + \epsilon^{(i)}\) as above.
4. **Train a model on \(S\) call it \(h_\theta : \mathbb{R}^d \rightarrow \mathbb{R}\)**

**WE EXAMINE** \[E \left[ (\hat{y}(x) - y)^2 \right] \quad \text{IN} \quad \text{TWO INDEPENDENT SOURCES OF ERRORS} \]

\(G_B = \text{Decompose the error}\)

\[
E \left[ (\hat{y}(x) - h_{\theta}(x) + \epsilon)^2 \right] = E \left[ \epsilon^2 \right] + 2E \left[ \epsilon (\hat{y}(x) - h_{\theta}(x)) \right] + E \left[ (\hat{y}(x) - h_{\theta}(x))^2 \right]
\]

\(E[\epsilon] = 0 \quad (\epsilon \text{ noise})\)

\[
\leq \sigma^2 + 0 + E \left[ \left( \hat{y}(x) - h_{\theta}(x) \right)^2 \right]
\]

**UNAVOIDABLE ERROR TERM**

Define \(h_{\text{avg}}(x) = E \left[ \hat{y}_S(x) \right] \quad \text{"long run of } S \text{"} \)

**CHECK:** \(S \text{ ENSURE AN } S, \text{ TRAIN } h_S, \text{ EVALUATE ON } x\)

This is random variable, and its expectation.
\[ E \left[ (h_S(x) - h_0(x))^2 \right] = E \left[ (h_S(x) - \hat{h}_S(x)) + (\hat{h}_S(x) - h_0(x))^2 \right] \\
= E \left[ (h_S(x) - \hat{h}_S(x))^2 \right] + (\hat{h}_S(x) - h_0(x))^2 + 0 \]

**Summary**: 
\[ E \left[ (h_S(x) - h_0(x))^2 \right] = \text{Unavoidable Error} + \text{Bias}^2 + \text{Variance} \]

**Error**: 
\[ \sigma^2 = \text{Variance} + \text{Bias}^2 + \text{Unavoidable Error} \]

**Model Complexity**: 
- Using distinct Dev/Holdout sets helps us assess variance and stability.
- If we use expressive model class, need to choose right bias.
- Regularization is at the heart of classical and modern theory.
Regularization

Reduce variance to obtain more robust model (to guard against overfitting)

\[
\rightarrow \text{Explicit (change loss we are optimizing)}
\]

\[
\rightarrow \text{Implicit (by amount of algorithm)}
\]

**Classical Settings**

\[
\argmin_{\theta} \frac{1}{2} \sum_{i=1}^{n} (x_i^T \theta - y_i)^2 + \frac{1}{2} \lambda \theta^T \theta
\]

\[
\lambda = 0 \Rightarrow \text{ordinary least squares}
\]

\[
\lambda = 10^6 \Rightarrow \theta \approx \mathbf{0} \text{ (probably looks good!)}
\]

Set \( \lambda \) to balance tradeoff! (we call this \( \lambda \))

**Solution:**

\[
\theta^* = \arg\min_{\theta} \frac{1}{2} \sum_{i=1}^{n} (x_i^T \theta - y_i)^2 + \frac{1}{2} \lambda \theta^T \theta = 0
\]

\[
\Leftrightarrow x^T(x \theta - y) = \lambda \theta = 0 \Rightarrow (x^T x + \lambda I) \theta = x^T y \quad \text{(normal equation)}
\]

**Undetermined Case** (modern use)

\[\text{Rank}(x^T x) < d \text{ if } \lambda = 0 \] there is not a unique solution!

\[\Rightarrow \exists \nu \text{ s.t. } \nu^T (x^T x) \nu = 0 \text{ hence } (x^T x)^T \nu = (x^T y)^T \nu \]

If \( \lambda = 0 \), then \( \theta \) is not unique!

Why? \( x^T x \) has eigenvalues \( \sigma_1^2 \geq \cdots \geq \sigma_d^2 > 0 \) (positive semidefinite)

\[x^T x + \lambda I \geq \sigma_1^2 + \lambda \geq \cdots \geq \sigma_d^2 + \lambda > 0 \text{ Positive definite}
\]

So, \( \theta^* = (x^T x + \lambda I)^{-1} x^T y \).

**Back to variance, did we reduce it?**

\[
\text{Var}_S(\hat{y}) = \mathbb{E}_{\varepsilon} \left[ (\hat{y}_S(x) - \hat{y}_M(x))^2 \right]
\]

We consider fixed design setup (points \( x \) fixed, only \( \varepsilon \) varied)

\[
\theta^* \text{ depends such on choice of } n' \text{-- } n
\]

\[\text{and/or}
\]

\[\text{and/or}
\]
\[
E \left[ (\Theta_{\lambda}(y) - \Theta_{\lambda}(xy)) \cdot x \right] \leq E \left[ \| \Theta_{\lambda}(y) - \Theta_{\lambda}(xy) \| \right] \cdot \|x\|^2
\]

Recall \( y = X \beta + \nu \sim N(0, \Sigma^{-1}) \) \( \nu \in \mathbb{R}^n \) \( \beta \in \mathbb{R}^p \)

\[
= E \left[ \| \left( X^T x + \lambda I \right)^{-1} X^T (y - \Theta_{\lambda}(xy)) \| \right] \cdot \| x \|^2
\]

\[
= E \left[ \| A \nu \|^2 \right] \quad A \nu \sim N(0, \Sigma^{-1}A^T A)
\]

\[
\leq \frac{c^2 \sigma_{\text{max}}^2}{(c^2 \sigma_{\text{max}}^2 + \lambda)^2} \| x \|^2
\]

As \( \lambda \) increases, variance goes down.

**Bonus Observation:** Implicitly regularize as well.

**Important in modern theory.**

Thought Experiment: Run gradient descent with \( \lambda = 0 \).

If we initialize at \( \Theta_0 \) note that:

\[
G_0 = \text{PassVar} (\Theta_0) + \text{PassVar} (\Theta_0)
\]

Claim: \( \text{PassVar} (\Theta^*) \) why?

\[
\Theta^{(n+1)} = \Theta^{(n)} - \lambda X^T (X \Theta - y)
\]

This update is always in \( \text{span} \{x\} \).

Observation: if we set \( \Theta^* = 0 \), we get minimum norm solution.

Conclusion:跟他也了

Deep learning undergrads need to control gradient update. So initialization plays a crucial role.
Belkin's 2018 “Double Descent”

First observed for deep nets, also true for kernels

→ SGD regularizes by picking minimum norm solution

in overdetermined case.

Memorization + Generalization (concrete shift to field)

Other methods amenable to this treatment

+ Augmentation
+ Dropout (adaptive regularization)
+ Mixed-ratio Augmentation

Model capacity ↑ loss! (interpolating)
Picking Hyperparameters

**Three Sets of Labeled Data**

- **Train** - fit parameters of the model
- **Dev** - "fit" hyperparameters eg \( \lambda \)
- **Test** - blinded for evaluation

**Example:** Pick degree

For degree \( c \in \{0, 1, 2, \ldots, k \} \)

- **Train** model \( c \) on **Train Set** \( \rightarrow bj \)
- **Score** each \( bj \) on **Dev Set**
- **Pick** best score, as \( h \)
- **Have** best on **Test Set**

If we have infinitely many models, eg \( \lambda \), give

\[ \lambda \in \{0, 2^{-n}, n \lambda^3, -3 \mid n \text{ same cases} \} \]
Improvements to Basic Scheme

**Data Efficiency**: Make best use of data in TRAIN/DEV/TEST

- Classical Stats: k-fold cross-validation (CV)
- Compute Efficiency: Many related hyperparameters

Modern ML: Sparse Ranking

**Data** k-fold CV

- **k=3** (but 5, 10, 20 are typical)

1. **Randomly permute Sb**

2. **Train**

<table>
<thead>
<tr>
<th>S(1), S(2)</th>
<th>S(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(1), S(2)</td>
<td>S(3)</td>
</tr>
<tr>
<td>S(1), S(2)</td>
<td>S(3)</td>
</tr>
</tbody>
</table>

3. **Combine scores** (Average, ...)

⇒ Use this score to pick the best
Computational Many techniques

Motivation Explores all parameters we don’t know how to set

Regularizers, Dropout, MCMC, etc., etc., etc.

Practical trick 1

1. Tune 1 parameter at a time (w/ grid search)
2. Sweep over base

2(5 + 6 + 7) vs. 5+6+7

Grid searches Grid searches (namely)

More Advanced Hyperparameter Search

Let $M = \{5, 6, 7\}^T$ models $T = 2$ (small $\theta$)

1. Run all models in $M$ for $T$ strays
2. Score all models in $M$
3. Set $M = \text{top} \frac{T}{2}$ of highest models

$T = 2T$

This uses constant small $\theta$ or random $M$, $T$ is constant

Runs for $T \log M^2$ rounds some guarantees

... lots more to be done...
Recall

- Bias + Variance → Modern
- Regularization → Implicit

- CV → Hyperparameter Search
- Tuning