STA 314: Statistical Methods for Machine Learning I
Lecture 3 - Bias-Variance Decomposition

Chris J. Maddison
University of Toronto
Today

- Expand a bit on Q3 and Q4 of the HW1.
- Today we will talk about the bias-variance decomposition, which is beginning to make more precise our discussion of overfitting and underfitting last class.
Given any finite set \( \{x_i\}_{i=1}^{N} \) of \( x_i \in \mathbb{R} \), we can define the uniform random variable over \( \{x_i\}_{i=1}^{N} \), which is any \( D \) such that
\[
P(D = x_i) = \frac{1}{N}
\]

Sampling from this random variable is easy: sample an integer \( J \in \{1, \ldots, N\} \) uniformly at random and return \( x_J \).

For this distribution, we have
\[
\mathbb{E}[D] = \sum_{i=1}^{N} P(D = x_i)x_i = \sum_{i=1}^{N} \frac{1}{N}x_i
\]
In supervised learning, our learning algorithms \((k\text{-NN, decision trees})\) produce predictions \(\hat{y}^*(x) \approx t\) for a query point \(x\).
We can think of this as picking a predictor function $\hat{y}^* \in \mathcal{H}$ from a hypothesis class by minimizing the average loss on the training set

$$\hat{y}^* = \arg \min_{y \in \mathcal{H}} \hat{R}[y, D^{train}]$$

Then, we measure the average loss on an unseen test set to approximate how well $\hat{y}^*$ does on the true data generating distribution,

$$\hat{R}[\hat{y}^*, D_{test}] \approx R[\hat{y}^*]$$
Recall: supervised learning

- This view of supervise learning is a very idealized view:
  - $k$-NN algorithm for $k > 1$ doesn’t really select the predictor by minimizing a global loss.
  - Decision tree fitting does select $\hat{y}^*$ based on training loss, but it is often greedy and sometimes does not find the global optimal $\hat{y}^*$.

- Still, it’s a very useful general model for supervised learning.
Let’s consider Q4 in HW1 as a way to review this supervised framework.
Recall that overly simple hypothesis classes underfit the data, and overly complex ones overfit.

Last lecture we talked about this intuitively.

We can quantify this effect in terms of the bias-variance decomposition.

- So far we’ve been talking about the training set as if it is fixed, but it makes more sense to think of it as random.
- So, we’d like to understand how our learning algorithm is impacted by selecting a predictor on a finite, random, training set.
Bias-Variance Decomposition: Basic Setup

- Recall: the training set \( D^{\text{train}} = \{(x^{(i)}, t^{(i)})\}_{i=1}^{N} \) contains \( N \) i.i.d. draws from a single data generating distribution \( p_{\text{data}} \).
- Consider a fixed query point \( x \) (green \( x \) below).
- Consider sampling many training sets \( D^{\text{train}}_n \) independently from \( p_{\text{data}} \).
Bias-Variance Decomposition: Basic Setup

- For each training set $D_n^{\text{train}}$, run learning alg. to get a predictor $\hat{y}_n^* \in \mathcal{H}$.
- Compute the prediction $\hat{y}_n^*(x)$ and compare it to a label $t$ drawn from $p_{\text{data}}(t|x)$.
- We can view $\hat{y}_n^*$ as a random variable, where the randomness comes from the choice of training set.
Biased-Variance Decomposition: Basic Setup

Here is the analogous setup for regression:

- **fit to dataset 1**
- **fit to dataset 2**
- **fit to dataset 3**

(query location)

**lots of fits**

**histogram of y**
Bias-Variance Decomposition: Basic Setup

- Recap of basic setup:
  - Fix a query point $x$.
  - Sample the (true) target $t$ from the conditional distribution $p_{data}(t|x)$.
  - Repeat:
    - Sample a random training dataset $D_{train}^n$ i.i.d. from the data generating distribution $p_{data}$.
    - Run the learning algorithm on $D_{train}^n$ to get a prediction $\hat{y}_n^*(x)$ from $H$ at $x$.
    - Compute the loss $L(\hat{y}_n^*(x), t)$.
  - Average the losses.

- Notice: $y$ is independent of $t$ given $x$.

This gives a distribution over the loss at $x$, with expectation $E[L(\hat{y}^*(x), t) | x]$ taken over $t$ and the random training set $D_{train}$ where $\hat{y}^* = \arg\min_{y \in H} \hat{R}[y, D_{train}]$.

- For each query point $x$, the expected loss is different. We are interested in minimizing the expectation of this with respect to $x \sim p_{data}(x)$. 
Bayes Optimality

- For now, focus on squared error loss, \( L(y, t) = \frac{1}{2}(y - t)^2 \) with \( y, t \in \mathbb{R} \).
- A first step: suppose we knew the conditional distribution \( p_{\text{data}}(t \mid x) \). What is the best deterministic value \( y(x) \in \mathbb{R} \) should we predict?
  - Here, we are treating \( t \) as a random variable and choosing \( y(x) \).
- **Claim:** \( y^*(x) = \mathbb{E}[t \mid x] \) is the best possible prediction.
- **Proof:** Consider a fixed \( y \in \mathbb{R} \)
  
  \[
  \mathbb{E}[(y - t)^2 \mid x] = \mathbb{E}[y^2 - 2yt + t^2 \mid x]
  = y^2 - 2y\mathbb{E}[t \mid x] + \mathbb{E}[t^2 \mid x]
  = y^2 - 2yy^*(x) + y^*(x)^2 + \text{Var}[t \mid x]
  = (y - y^*(x))^2 + \text{Var}[t \mid x]
  \]
Bayes Optimality

\[ \mathbb{E}[(y - t)^2 \mid x] = (y - y^*(x))^2 + \text{Var}[t \mid x] \]

- The first term is nonnegative, and can be made 0 by setting \( y = y^*(x) \).
- The second term corresponds to the inherent unpredictability, or noise, of the targets, and is called the Bayes error.
  - This is the best we can ever hope to do with any learning algorithm. An algorithm that achieves it is Bayes optimal.
  - Notice that this term doesn’t depend on \( y \).
- This process of choosing a single value \( y^*(x) \) based on \( p_{\text{data}}(t \mid x) \) is an example of decision theory.
Bayes Optimality

- But, in practice, our prediction \( \hat{y}^*(x) \) is not \( y^*(x) \). Instead, it is a random variable (where the randomness comes from randomness of the training set) taking values in \( \mathcal{H} \).

- We can decompose out the expected loss.

- Suppressing the dependence on \( x \) for clarity:

\[
\mathbb{E}[(\hat{y}^* - t)^2] = \mathbb{E}[(\hat{y}^* - y^*)^2] + \text{Var}(t) \\
= \mathbb{E}[y^*^2 - 2y^*\hat{y}^* + \hat{y}^*^2] + \text{Var}(t) \\
= y^*^2 - 2y^*\mathbb{E}[\hat{y}^*] + \mathbb{E}[\hat{y}^*^2] + \text{Var}(t) \\
= y^*^2 - 2y^*\mathbb{E}[\hat{y}^*] + \mathbb{E}[\hat{y}^*]^2 + \text{Var}(\hat{y}^*) + \text{Var}(t) \\
= (y^* - \mathbb{E}[\hat{y}^*])^2 + \text{Var}(\hat{y}^*) + \text{Var}(t)
\]

\( \text{bias} \) \quad \text{variance} \quad \text{Bayes error}
Let’s step back and consider what we just did. First, recall:

- Picking a predictor by minimizing the average loss on the training set
  
  \[ \hat{y}^* = \arg \min_{y \in H} \hat{R}[y, D^{train}] \]

  returns a random predictor \( \hat{y}^* \).

- But, we’re interested in our performance in terms of expected loss:

  \[ \mathcal{R}[\hat{y}^*] \]

In our case:

\[ \mathcal{R}[\hat{y}^*] = \mathbb{E} \left[ \mathbb{E} \left[ (\hat{y}^*(x) - t)^2 \mid x \right] \right]. \]
Bayes Optimality

\[ \mathbb{E} \left[ \mathbb{E} \left[ (\hat{y}^*(x) - t)^2 \mid x \right] \right] = \mathbb{E} \left[ (y^*(x) - \mathbb{E}[\hat{y}^*(x) \mid x])^2 \right] + \text{Var}[\hat{y}^*(x) \mid x] + \text{Var}[t \mid x] \]

- So, we just split the expected loss $\mathcal{R}[\hat{y}^*]$ into three terms:
  - **bias**: how wrong the expected prediction is
  - **variance**: the amount of variability in the predictions
  - **Bayes error**: the inherent unpredictability of the targets

- How does our choice of $\mathcal{H}$ interact with this analysis?
Bayes Optimality

- If $H$ is large, then $\hat{y}^*$ can get close $y^*$, therefore reducing bias. It’s also sensitive to the finite training set, therefore increasing variance.

- If $H$ is small, then $\hat{y}^*$ is typically from $y^*$, therefore increasing bias. It’s less sensitive to the finite training set, therefore reducing variance.

- Even though this analysis only applies to squared error, we often loosely use “bias” and “variance” as synonyms for “underfitting” and “overfitting.”
Bias and Variance

- Throwing darts = predictions for each draw of a dataset

- Be careful, the expected loss averages over points \( x \) from the data distribution, so this produces its own type of variance.

Source: ESL.
In practice, measure the average loss $\hat{R}[\hat{y}^*, D_{\text{test}}]$ on the test set instead of $R[\hat{y}^*]$.

Let’s visualize the bias-variance decomposition by plotting the space of predictions of the model, where each axis correspond to predictions on a two test examples $(x^{(1)}, x^{(2)})$. 

![Diagram showing contours of expected loss with predictions $(\hat{y}^*(x^{(1)}), \hat{y}^*(x^{(2)}))$ and $(y^*(x^{(1)}), y^*(x^{(2)}))$.]
The Bayes error is an irreducible error that comes from the randomness in \( p_{\text{data}}(t \mid x) \).
Selecting a predictor $\hat{y}^* \in \mathcal{H}$ from a training set comes with bias and variance.
An overly simple model (e.g. $k$-NN with large $k$) might have

- high bias (too simplistic to capture the structure in the data)
- low variance (there's enough data to get a stable estimate of the decision boundary)

\[ \mathbb{E}[^\hat{\mathbf{y}}^* (\mathbf{x}) | \mathbf{x}] \]
Bias/Variance Decomposition: Another Visualization

- An overly complex model (e.g. KNN with $k = 1$) may have
  - low bias (since it learns all the relevant structure)
  - high variance (it fits the quirks of the data you happened to sample)

Bayes optimal prediction

$E[\hat{y}^*(x) | x]$
Before we move on to bagging, it’s a good time to mention validation.

We may want to assess how likely a learning algorithm is to generalize before picking one and reporting the final test error.

In other words, until now we’ve been picking predictors that optimize the training loss, but we want a technique for picking predictors that are likely to generalize as well.
For example, we may want to assess the following types of choices:

1. **Hyper-parameters of the learning algorithm that lead to better generalization.** Often there are parameters that cannot be fit on the training set, e.g., $k$ in $k$-NN, because the training set would give meaningless answers about the best setting, i.e., $k = 1$ is always gives optimal training set loss for $k$-NN.

2. **Picking predictors that generalize better.** E.g., should we use a decision tree or $k$-NN if we want to generalize?

- **We make these choices using validation** to avoid measuring test loss (then the test set would no longer be unseen data!).

- **Suppose we are trying to estimate the generalization of two learning algorithms,** e.g., a decision tree and a $k$-NN model.
The most common method of validation is to hold-out a subset of the training set and use it to assess how likely we are to generalize to unseen data.

In our example of deciding between a decision tree and $k$-NN in terms of generalization, we would fit $\hat{y}_{kNN}^*$ and $\hat{y}_{d\text{-tree}}^*$ on the training set and measure the average loss on the validation set

$$\hat{R}[\hat{y}_{kNN}^*, D_{valid}] \text{ vs. } \hat{R}[\hat{y}_{d\text{-tree}}^*, D_{valid}]$$

We pick the predictor $\hat{y}_{kNN}^*$ vs. $\hat{y}_{d\text{-tree}}^*$ with lowest validation loss.

Problem: this is usually a waste of data.
- Second most common way: partition training data randomly into $K$ equally sized subsets. For each “turn”, use the first $K - 1$ subsets (or “folds”) as training data and the last subset as validation.
K-fold cross validation

- In our running example: fit a new predictor using each learning algorithm on $K - 1$ folds for each of the $K$ turns, and measure the validation loss on the held-out fold, averaged over the turns:

$$\frac{1}{K} \sum_{i=1}^{K} \hat{R}[\hat{y}_{kNN,i}^*, D_{i}^{valid}] \text{ vs. } \frac{1}{K} \sum_{i=1}^{K} \hat{R}[\hat{y}_{d-tree,i}^*, D_{i}^{valid}]$$

where $\hat{y}_{A,i}^*$ is the predictor fit on the training subset of the $i$th turn using algorithm $A$ and $D_{i}^{valid}$ is the validation subset of the $i$th turn.

- We pick the learning algorithm, e.g., $k$-NN v. decision tree, with lowest validation loss averaged across the $K$ turns.