Selecting the Machine Learning Model

Chris Cornwell

April 14, 2025

Outline

Outline

Outline

Intro

We begin with a common example – shows that getting small value of cost function on training data is not sufficient.

Intro

We begin with a common example – shows that getting small value of cost function on training data is not sufficient.

Say have data $S = \{(x_i, y_i)\}_{i=1}^{P}$ and want to fit a polynomial to this in the plane. That is, we pick a degree d and will use a polynomial function

$$w_d^\star x^d + w_{d-1}^\star x^{d-1} + \ldots + w_1^\star x + b^\star$$

(so, $\tilde{\mathbf{w}}^\star = (b^\star, w_1^\star, \ldots, w_d^\star)$ is our solved stationary point from the normal equation).

What happens when we choose d to be rather large?

Intro

We begin with a common example – shows that getting small value of cost function on training data is not sufficient.
Say have data $S = \{(x_i, y_i)\}_{i=1}^{P}$ and want to fit a polynomial to this in the plane.
That is, we pick a degree d and will use a polynomial function

$$w_d^\star x^d + w_{d-1}^\star x^{d-1} + \ldots + w_1^\star x + b^\star$$

(so, $\tilde{\mathbf{w}}^\star = (b^\star, w_1^\star, \ldots, w_d^\star)$ is our solved stationary point from the normal equation).
What happens when we choose d to be rather large? $d = 12$

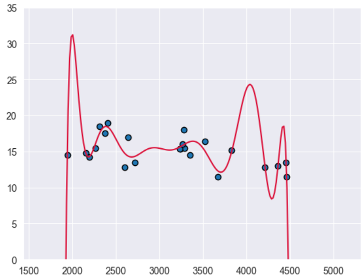

Figure: Have blue points as data; will train on them.

Intro

We begin with a common example – shows that getting small value of cost function on training data is not sufficient.

Say have data $S = \{(x_i, y_i)\}_{i=1}^{P}$ and want to fit a polynomial to this in the plane.

That is, we pick a degree d and will use a polynomial function

$$w_d^\star x^d + w_{d-1}^\star x^{d-1} + \ldots + w_1^\star x + b^\star$$

(so, $\tilde{\mathbf{w}}^\star = (b^\star, w_1^\star, \ldots, w_d^\star)$ is our solved stationary point from the normal equation).

What happens when we choose d to be rather large? $d = 12$

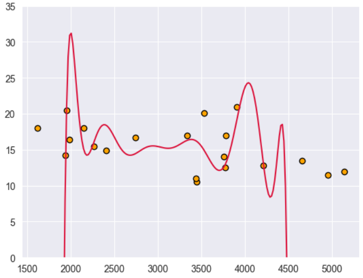

Figure: Orange points are from same population but weren't in sample.

Decision tree example

Given sample data $\mathcal{S}$, consisting of greyscale images of handwritten digits $0, 1, \ldots, 9$, say we want a decision tree model to predict the correct digit, given an image.

Let the images be p pixels by p pixels. For an image, we might take the p^2 greyscale values (0–255) and lay them end to end to get a vector in $\mathbb{R}^{p^2}$. Then, could train a decision tree with however many splits are needed so that each leaf has points in $\mathcal{S}$ with only one label.

- Gives 100% accuracy on $\mathcal{S}$, the data used to determine the parameters.
- But, the model won't perform nearly as well on data not from $\mathcal{S}$.

Let's look at doing exactly this, using images of digits available from the Python package scikit-learn.

Decision tree model for handwritten digits

The images of digits from scikit-learn are 8 pixels by 8 pixels. With the submodule `datasets` imported from `sklearn`, the data can be loaded as follows.

```python
data = datasets.load_digits()
x = data.images
y = data.target
```

To convert each 8×8 array to a vector in $\mathbb{R}^{64}$, can use the `reshape` method; after assigning `x` and `y` as above, code below would output `(1797, 64)`.

```python
# make each 8x8 array into 1d array with 64 entries
x = x.reshape(len(x), -1)
x.shape
```

Randomly select 20% of the data to test model with – it won't be used to determine the decision tree.

```python
n_test = int(0.2*len(x))
test_indices = np.random.choice(len(x), size=n_test, replace=False)
train_indices = np.delete(np.arange(len(x)), test_indices)
x_test, y_test = x[test_indices], y[test_indices]
x_train, y_train = x[train_indices], y[train_indices]
```

**Alternatively, you can use the function* `train_test_split` *from the scikit-learn submodule* `model_selection`.

Decision tree model for handwritten digits

With the setup from the last slide, use the training data to determine the decision tree.

There is a submodule `tree` within scikit-learn that we can use.

```
model = tree.DecisionTreeClassifier()
model.fit(x_train, y_train)
```

The default behavior of the `.fit()` method is that the tree has as many splits (decision stumps) as needed so that each leaf consists of points with a single label. The accuracy on x_train is, therefore, 100%.

The accuracy of the model on `x_test` will depend some on which points were put into `x_test`; however, it tends to be around only 85%.

What happened is that the **hypothesis class**, that set of functions that were possible outcomes to be the trained model, was too large (allowed too many possibilities for the resulting model).

Keeping the hypothesis class small

The cost function you compute is empirical (based on observed data). We discuss a (somewhat absurd) example to demonstrate the point that

$$\text{small counting cost} \;\not\Longrightarrow\; \text{high accuracy on "yet unseen" data}$$

Allowing for *any* function as a possible model, for any training data $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, with labels in $\{1, -1\}$, we set

$$f_S(\mathbf{x}) = \begin{cases} y_i, & \text{if } \mathbf{x} = \mathbf{x}_i \\ -1, & \text{otherwise.} \end{cases}$$

Regardless of $\mathcal{S}$, or the population distribution that it is a sample from, the empirical loss $\mathcal{L}_S(f_S) = \frac{\#\{i:\ y_i \neq f_S(\mathbf{x}_i)\}}{n}$ is zero.

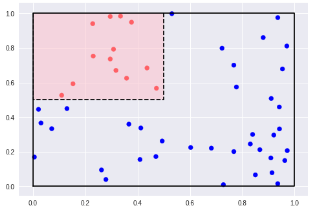

Figure: Upper-left square labeled -1

Keeping the hypothesis class small

The cost function you compute is empirical (based on observed data). We discuss a (somewhat absurd) example to demonstrate the point that

$$\text{small counting cost} \not\Longrightarrow \text{high accuracy on “yet unseen” data}$$

Suppose that the population data consists of points in $[0,1]^2$ (the square of points with both coordinates between 0 and 1), and that a point (x_1, x_2) has label -1 if and only if $0 \leq x_1 < 0.5$ and $0.5 < x_2 \leq 1$ (See Figure).

Since a sample $\mathcal{S}$ is finite, a randomly chosen point has probability 0 of being in $\mathcal{S}$. Thus, given a random $\mathbf{x} \in [0,1]^2$, with probability $3/4$ the predicted label $f_S(\mathbf{x})$ is wrong.
The issue with both
this model, and the decision tree that had no restriction on depth, is that the **variance** of the procedure $S \rightsquigarrow f_S$ is too large.

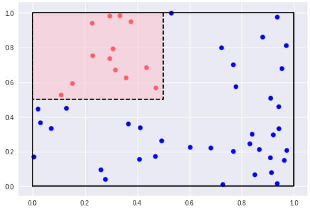

Figure: Upper-left square labeled -1

Typically, want to incorporate into your process a simulation of using the model on unseen data.
Use a random procedure to split the data. Very often, this is done with an 80-20 split. (20% “test” data)
You **are not supposed to touch the test data** until your model is finished and you are evaluating it.

Bias-Variance Trade-off

Suppose that we have chosen a hypothesis class (a class of parameterized functions, say), and a procedure $S \leadsto f_S$ that determines a prediction function from training data $\mathcal{S}$.

- *By f_S, we no longer mean only the absurd function two slides previous; instead, for a type of machine learning model and procedure to train, f_S is whichever prediction function results from training on $\mathcal{S}$.*

[1]Using subscripts for what we average over.

Bias-Variance Trade-off

Suppose that we have chosen a hypothesis class (a class of parameterized functions, say), and a procedure $S \rightsquigarrow f_S$ that determines a prediction function from training data $\mathcal{S}$.

- *By f_S, we no longer mean only the absurd function two slides previous; instead, for a type of machine learning model and procedure to train, f_S is whichever prediction function results from training on $\mathcal{S}$.*

Using Mean Squared Error as loss function, want to understand the *expected population loss*, not only over the distribution that data is drawn from, but also over training sets $\mathcal{S}$. That is, over all $(\mathbf{x}, y)$ and $\mathcal{S}$, the expected value[1] $\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2]$.

[1]Using subscripts for what we average over.

Bias-Variance Trade-off

Suppose that we have chosen a hypothesis class (a class of parameterized functions, say), and a procedure $S \rightsquigarrow f_S$ that determines a prediction function from training data $\mathcal{S}$.

- *By f_S, we no longer mean only the absurd function two slides previous; instead, for a type of machine learning model and procedure to train, f_S is whichever prediction function results from training on $\mathcal{S}$.*

Using Mean Squared Error as loss function, want to understand the *expected population loss*, not only over the distribution that data is drawn from, but also over training sets $\mathcal{S}$. That is, over all $(\mathbf{x}, y)$ and $\mathcal{S}$, the expected value[1] $\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2]$.
The expected population loss will decompose, with a so-called Bias term, and a Variance term. Their relationship is roughly as below.

[1]Using subscripts for what we average over.

Bias-Variance Trade-off

Suppose that we have chosen a hypothesis class (a class of parameterized functions, say), and a procedure $S \rightsquigarrow f_S$ that determines a prediction function from training data $\mathcal{S}$.

- *By f_S, we no longer mean only the absurd function two slides previous; instead, for a type of machine learning model and procedure to train, f_S is whichever prediction function results from training on $\mathcal{S}$.*

Using Mean Squared Error as loss function, want to understand the *expected population loss*, not only over the distribution that data is drawn from, but also over training sets $\mathcal{S}$. That is, over all $(\mathbf{x}, y)$ and $\mathcal{S}$, the expected value[2] $\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2]$.

[2] Using subscripts for what we average over.

[3] This takes some advanced mathematics to carefully describe. It means there is a "measure" on sets of functions, which allows you to integrate over a set of functions to get a new function.

Bias-Variance Trade-off

Suppose that we have chosen a hypothesis class (a class of parameterized functions, say), and a procedure $S \rightsquigarrow f_S$ that determines a prediction function from training data $\mathcal{S}$.

- *By f_S, we no longer mean only the absurd function two slides previous; instead, for a type of machine learning model and procedure to train, f_S is whichever prediction function results from training on $\mathcal{S}$.*

Using Mean Squared Error as loss function, want to understand the *expected population loss*, not only over the distribution that data is drawn from, but also over training sets $\mathcal{S}$. That is, over all $(\mathbf{x}, y)$ and $\mathcal{S}$, the expected value[2] $\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2]$.

Let $\bar{f}$ be the expected function, averaged over training sets,[3] so that $\mathbb{E}_{\mathcal{S}}[f_S] = \bar{f}$.

[2] Using subscripts for what we average over.

[3] This takes some advanced mathematics to carefully describe. It means there is a “measure” on sets of functions, which allows you to integrate over a set of functions to get a new function.

Bias-Variance Trade-off

Suppose that we have chosen a hypothesis class (a class of parameterized functions, say), and a procedure $S \rightsquigarrow f_S$ that determines a prediction function from training data $\mathcal{S}$.

- *By f_S, we no longer mean only the absurd function two slides previous; instead, for a type of machine learning model and procedure to train, f_S is whichever prediction function results from training on $\mathcal{S}$.*

Using Mean Squared Error as loss function, want to understand the *expected population loss*, not only over the distribution that data is drawn from, but also over training sets $\mathcal{S}$. That is, over all $(\mathbf{x}, y)$ and $\mathcal{S}$, the expected value[2] $\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2]$.
Let $\bar{f}$ be the expected function, averaged over training sets,[3] so that $\mathbb{E}_{\mathcal{S}}[f_S] = \bar{f}$.
In practice, could approximate $\bar{f}$ in the following way: for samples $\mathcal{S}_1, \mathcal{S}_2 \ldots, \mathcal{S}_N$, each with same number of points, drawn i.i.d. from the population, then for sufficiently large N,

$$\bar{f}(\mathbf{x}) \approx \frac{1}{N}\sum_{i=1}^{N} f_{S_i}(\mathbf{x}).$$

[2] Using subscripts for what we average over.

[3] This takes some advanced mathematics to carefully describe. It means there is a "measure" on sets of functions, which allows you to integrate over a set of functions to get a new function.

Bias-Variance Trade-off

The expected population loss decomposes as follows.

$$\begin{aligned}
&\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2] \\
&\quad = \mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}) + \bar{f}(\mathbf{x}) - y)^2] \\
&\quad = \mathbb{E}_{S,\mathbf{x}}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - y)^2] + 2\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x}) - y)]
\end{aligned}$$

Since $\bar{f}(\mathbf{x}) = \mathbb{E}_S[f_S(\mathbf{x})]$, the last term vanishes and so

$$\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2] = \mathbb{E}_{S,\mathbf{x}}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - y)^2].$$

Label y may be a distribution, given $\mathbf{x}$, not deterministic function. For $\bar{y}(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}]$, similar argument shows

$$\mathbb{E}_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - y)^2] = \mathbb{E}_{\mathbf{x}}[(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2] + \mathbb{E}_{\mathbf{x},y}[(\bar{y}(\mathbf{x}) - y)^2].$$

And so

$$\mathbb{E}_{S,\mathbf{x},y}[(f_S(\mathbf{x}) - y)^2] = \underbrace{\mathbb{E}_{S,\mathbf{x}}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))^2]}_{\text{Variance}\uparrow} + \underbrace{\mathbb{E}_{\mathbf{x}}[(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2]}_{\text{Bias}^2\uparrow} + \underbrace{\mathbb{E}_{\mathbf{x},y}[(\bar{y}(\mathbf{x}) - y)^2]}_{\text{Noise}\uparrow}.$$