

# Selecting the Machine Learning Model

Chris Cornwell

April 14, 2025

# Outline

## Overfitting

Having only concern be to minimize empirical loss

# Outline

## Overfitting

Having only concern be to minimize empirical loss

# Intro

Given sample data  $\mathcal{S}$ , consisting of greyscale images of handwritten digits 0, 1,  $\dots$ , 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.

```
1 | data = datasets.load_digits()
2 | x = data.images
3 | y = data.target
```

To convert each  $8 \times 8$  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).

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

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

```
1 | n_test = int(0.2*len(x))
2 | test_indices = np.random.choice(len(x), size=n_test, replace=False)
3 | train_indices = np.delete(np.arange(len(x)), test_indices)
4 | x_test, y_test = x[test_indices], y[test_indices]
5 | 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.

```
1 | model = tree.DecisionTreeClassifier()  
2 | 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

We discuss a (somewhat absurd) example to demonstrate the point that  
small empirical loss  $\nRightarrow$  small population loss

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

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

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

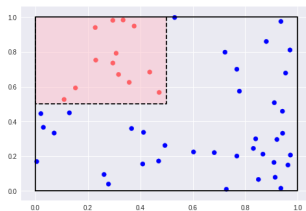


Figure: Upper-left square labeled -1

# Keeping the hypothesis class small

We discuss a (somewhat absurd) example to demonstrate the point that  
small empirical loss  $\nRightarrow$  small population loss

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  $f_{\mathcal{S}}(\mathbf{x})$  is wrong.

The issue with both this model, and the decision tree that had no restriction on depth, is that the **variance** of the hypothesis class is too large.

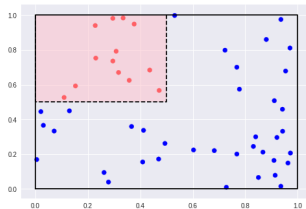


Figure: Upper-left square labeled -1



## Bias-Variance Trade-off

Suppose that we have chosen a hypothesis class (a class of parameterized functions, say), and a procedure that, given a training sample  $\mathcal{S}$  from the population, determines a prediction function  $f_{\mathcal{S}}$ .

In order to understand the *expected* population loss, we want to consider the variance of  $f_{\mathcal{S}}(\mathbf{x})$ , not only over the input space but over possible training sets  $\mathcal{S}$ . That is, over all  $(\mathbf{x}, y)$  and  $\mathcal{S}$ , the expected value  $\mathbb{E}[(f_{\mathcal{S}}(\mathbf{x}) - y)^2]$ .

Let  $\bar{h}$  be the expected prediction function, over training sets.<sup>1</sup> In practice, you could approximate  $\bar{h}$  in the following way: for samples  $\mathcal{S}_1, \dots, \mathcal{S}_N$ , each with the same number of points, drawn i.i.d. from the population, then for sufficiently large  $N$ ,

$$\bar{h}(\mathbf{x}) \approx \frac{1}{N} \sum_{i=1}^N h_{\mathcal{S}_i}(\mathbf{x}).$$

---

<sup>1</sup>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

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

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

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

A similar argument shows that

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

And so

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

Variance  $\uparrow$

Bias<sup>2</sup>  $\uparrow$

Noise  $\uparrow$