APPENDIX D

Supervised learning in a nutshell

In which we try to describe the outlines of the "lifecycle" of supervised learning, including hyperparameter tuning and evaluation of the final product.

D.1 General case

We start with a very generic setting.

D.1.1 Minimal problem specification

Given:

- Space of inputs $\mathcal{X}$
- Space of outputs $\mathcal{Y}$
- Space of possible *hypotheses* $\mathcal{H}$ such that each $h \in \mathcal{H}$ is a function $h : \mathcal{X} \to \mathcal{Y}$
- Loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$

a *supervised learning algorithm* $\mathcal{A}$ takes as input a *data set* of the form

$$\mathcal{D} = \left\{ \left(x^{(1)}, y^{(1)} \right), \ldots, \left(x^{(n)}, y^{(n)} \right) \right\} \ ,$$

where $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ and returns an $h \in \mathcal{H}$.

D.1.2 Evaluating a hypothesis

Given a problem specification and a set of data $\mathcal{D}$, we evaluate hypothesis h according to average loss, or *error*,

$$\mathcal{E}(h, \mathcal{L}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{\mathcal{D}} \mathcal{L}(h(x^{(i)}), y^{(i)})$$

If the data used for evaluation *were not used during learning of the hypothesis* then this is a reasonable estimate of how well the hypothesis will make additional predictions on new data from the same source.

D.1.3 Evaluating a supervised learning algorithm

A *validation strategy* $\mathcal{V}$ takes an algorithm $\mathcal{A}$, a loss function $\mathcal{L}$, and a data source $\mathcal{D}$ and produces a real number which measures how well $\mathcal{A}$ performs on data from that distribution.

D.1.3.1 Using a validation set

In the simplest case, we can divide $\mathcal{D}$ into two sets, $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{val}}$, train on the first, and then evaluate the resulting hypothesis on the second. In that case,

$$\mathcal{V}(\mathcal{A}, \mathcal{L}, \mathcal{D}) = \mathcal{E}(\mathcal{A}(\mathcal{D}^{\text{train}}), \mathcal{L}, \mathcal{D}^{\text{val}}) \ .$$

D.1.3.2 Using multiple training/evaluation runs

We can't reliably evaluate an algorithm based on a single application of it to a single training and test set, because there are many aspects of the training and testing data, as well as, sometimes, randomness in the algorithm itself, that cause *variance* in the performance of the algorithm. To get a good idea of how well an algorithm performs, we need to, multiple times, train it and evaluate the resulting hypothesis, and report the average over K executions of the algorithm of the error of the hypothesis it produced each time.

We divide the data into 2K random non-overlapping subsets: $\mathcal{D}_1^{\text{train}}, \mathcal{D}_1^{\text{val}}, \ldots, \mathcal{D}_K^{\text{train}}, \mathcal{D}_K^{\text{val}}$. Then,

$$\mathcal{V}(\mathcal{A}, \mathcal{L}, \mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{E}(\mathcal{A}(\mathcal{D}_k^{\text{train}}), \mathcal{L}, \mathcal{D}_k^{\text{val}}) \ .$$

D.1.3.3 Cross validation

In *cross validation*, we do a similar computation, but allow data to be re-used in the K different iterations of training and testing the algorithm (but never share training and testing data for a single iteration!). See Section 2.7.2.2 for details.

D.1.4 Comparing supervised learning algorithms

Now, if we have two different algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, we might be interested in knowing which one will produce hypotheses that generalize the best, using data from a particular source. We could compute $\mathcal{V}(\mathcal{A}_1, \mathcal{L}, \mathcal{D})$ and $\mathcal{V}(\mathcal{A}_\in, \mathcal{L}, \mathcal{D})$, and prefer the algorithm with lower validation error. More generally, given algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_M$, we would prefer

$$\mathcal{A}^* = \arg\min_m \mathcal{V}(\mathcal{A}_M, \mathcal{L}, \mathcal{D}) \ .$$

D.1.5 Fielding a hypothesis

Now what? We have to deliver a hypothesis to our customer. We now know how to find the algorithm, $\mathcal{A}^*$, that works best for our type of data. We can apply it to all of our data to get the best hypothesis we know how to create, which would be

$$h^* = \mathcal{A}^*(\mathcal{D}) \ ,$$

and deliver this resulting hypothesis as our best product.

D.1.6 Learning algorithms as optimizers

A majority of learning algorithms have the form of optimizing some objective involving the training data and a loss function.

So for example, (assuming a perfect optimizer which doesn't, of course, exist) we might say our algorithm is to solve an optimization problem:

$$\mathcal{A}(\mathcal{D}) = \arg\min_{h \in H} \mathcal{J}(h; \mathcal{D}) \quad .$$

Interestingly, this loss function is *not always the same* as the loss function that is used for evaluation! We will see this in logistic regression.

Our objective often has the form

$$\mathcal{J}(h; \mathcal{D}) = \mathcal{E}(h, \mathcal{L}, \mathcal{D}) + \mathcal{R}(h) \quad ,$$

where $\mathcal{L}$ is a loss to be minimized during training and $\mathcal{R}$ is a regularization term.

D.1.7 Hyperparameters

Often, rather than comparing an arbitrary collection of learning algorithms, we think of our learning algorithm as having some parameters that affect the way it maps data to a hypothesis. These are not parameters of the hypothesis itself, but rather parameters of the *algorithm*. We call these *hyperparameters*. A classic example would be to use a hyperparameter λ to govern the weight of a regularization term on an objective to be optimized:

$$\mathcal{J}(h; \mathcal{D}) = \mathcal{E}(h, \mathcal{L}, \mathcal{D}) + \lambda \mathcal{R}(h) \quad .$$

Then we could think of our algorithm as $\mathcal{A}(\mathcal{D}; \lambda)$. Picking a good value of λ is the same as comparing different supervised learning algorithms, which is accomplished by validating them and picking the best one!

D.2 Concrete case: linear regression

In linear regression the problem formulation is this:

- $\mathcal{X} = \mathbb{R}^d$
- $\mathcal{Y} = \mathbb{R}$
- $\mathcal{H} = \{\theta^T x + \theta_0\}$ for values of parameters $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$.
- $\mathcal{L}(g, y) = (g - y)^2$

Our learning algorithm has hyperparameter λ and can be written as:

$$\mathcal{A}(\mathcal{D}; \lambda) = \Theta^*(\lambda, \mathcal{D}) = \arg\min_{\theta, \theta_0} \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} (\theta^T x + \theta_0 - y)^2 + \lambda \|\theta\|^2 \quad .$$

For a particular training data set and parameter λ, it finds the best hypothesis on this data, specified with parameters $\Theta = (\theta, \theta_0)$, written $\Theta^*(\lambda, \mathcal{D})$.

Picking the best value of the hyperparameter is choosing among learning algorithms. We could, most simply, optimize using a single training / validation split, so $\mathcal{D} = \mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{val}}$, and

$$\begin{aligned}
\lambda^* &= \arg\min_{\lambda} \mathcal{V}(\mathcal{A}_\lambda, \mathcal{L}, \mathcal{D}^{\text{val}}) \\
&= \arg\min_{\lambda} \mathcal{E}(\Theta^*(\lambda, \mathcal{D}^{\text{train}}), \text{mse}, \mathcal{D}^{\text{val}}) \\
&= \arg\min_{\lambda} \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x,y) \in \mathcal{D}_{\text{val}}} (\theta^*(\lambda, \mathcal{D}^{\text{train}})^T x + \theta_0^*(\lambda, \mathcal{D}^{\text{train}}) - y)^2
\end{aligned}$$

It would be much better to select the best λ using multiple runs or cross-validation; that would just be a different choices of the validation procedure $\mathcal{V}$ in the top line.

Note that we don't use regularization here because we just want to measure how good the output of the algorithm is at predicting values of *new* points, and so that's what we measure. We use the regularizer during training when we don't want to focus only on optimizing predictions on the training data.

Finally! To make a predictor to ship out into the world, we would use all the data we have, $\mathcal{D}$, to train, using the best hyperparameters we know, and return

$$\begin{aligned}\Theta^* &= \mathcal{A}(\mathcal{D}; \lambda^*) \\ &= \Theta^*(\lambda^*, \mathcal{D}) \\ &= \arg\min_{\theta,\theta_0} \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} (\theta^T x + \theta_0 - y)^2 + \lambda^* \|\theta\|^2\end{aligned}$$

Finally, a customer might evaluate this hypothesis on their data, which we have never seen during training or validation, as

$$\begin{aligned}\mathcal{E}^{\text{test}} &= \mathcal{E}(\Theta^*, \text{mse}, \mathcal{D}^{\text{test}}) \\ &= \frac{1}{|\mathcal{D}^{\text{test}}|} \sum_{(x,y)\in\mathcal{D}^{\text{test}}} (\theta^{*T} x + \theta_0^* - y)^2\end{aligned}$$

Here are the same ideas, written out in informal pseudocode.

```
# returns theta_best(D, lambda)
define train(D, lambda):
    return minimize(mse(theta, D) + lambda * norm(theta)**2, theta)

# returns lambda_best using very simple validation
define simple_tune(D_train, D_val, possible_lambda_vals):
    scores = [mse(train(D_train, lambda), D_val)
                  for lambda in possible_lambda_vals]
    return possible_lambda_vals[least_index[scores]]

# returns theta_best overall
define theta_best(D_train, D_val, possible_lambda_vals):
  return train(D_train + D_val,
               simple_tune(D_train, D_val, possible_lambda_vals))

# customer evaluation of the theta delivered to them
define customer_val(theta):
    return mse(theta, D_test)
```

D.3 Concrete case: logistic regression

In binary logistic regression the problem formulation is as follows. We are writing the class labels as 1 and 0.

- $\mathcal{X} = \mathbb{R}^d$

- $\mathcal{Y} = \{+1, 0\}$
- $\mathcal{H} = \{\sigma(\theta^T x + \theta_0)\}$ for values of parameters $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$.
- $\mathcal{L}(g, y) = \mathcal{L}_{01}(g, h)$
- Proxy loss $\mathcal{L}_{\text{nll}}(g, y) = -(y \log(g) + (1 - y) \log(1 - g))$

Our learning algorithm has hyperparameter λ and can be written as:

$$\mathcal{A}(\mathcal{D}; \lambda) = \Theta^*(\lambda, \mathcal{D}) = \arg\min_{\theta, \theta_0} \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \mathcal{L}_{\text{nll}}(\sigma(\theta^T x + \theta_0), y) + \lambda \|\theta\|^2 \quad .$$

For a particular training data set and parameter λ, it finds the best hypothesis on this data, specified with parameters $\Theta = (\theta, \theta_0)$, written $\Theta^*(\lambda, \mathcal{D})$ according to the proxy loss $\mathcal{L}_{\text{nll}}$.

Picking the best value of the hyperparameter is choosing among learning algorithms based on their actual predictions. We could, most simply, optimize using a single training / validation split, so $\mathcal{D} = \mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{val}}$, and *we use the real 01 loss*:

$$\begin{aligned}
\lambda^* &= \arg\min_{\lambda} \mathcal{V}(\mathcal{A}_\lambda, \mathcal{L}_{01}, \mathcal{D}^{\text{val}}) \\
&= \arg\min_{\lambda} \mathcal{E}(\Theta^*(\lambda, \mathcal{D}^{\text{train}}), \mathcal{L}_{01}, \mathcal{D}^{\text{val}}) \\
&= \arg\min_{\lambda} \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x,y) \in \mathcal{D}_{\text{val}}} \mathcal{L}_{01}(\sigma(\theta^*(\lambda, \mathcal{D}^{\text{train}})^T x + \theta_0^*(\lambda, \mathcal{D}^{\text{train}})), y)
\end{aligned}$$

It would be much better to select the best λ using multiple runs or cross-validation; that would just be a different choices of the validation procedure $\mathcal{V}$ in the top line.

Finally! To make a predictor to ship out into the world, we would use all the data we have, $\mathcal{D}$, to train, using the best hyperparameters we know, and return

$$\Theta^* = \mathcal{A}(\mathcal{D}; \lambda^*)$$

Study Question: What loss function is being optimized inside this algorithm?

Finally, a customer might evaluate this hypothesis on their data, which we have never seen during training or validation, as

$$\mathcal{E}^{\text{test}} = \mathcal{E}(\Theta^*, \mathcal{L}_{01}, \mathcal{D}^{\text{test}})$$

The customer just wants to buy the right stocks! So we use the real $\mathcal{L}_{01}$ here for validation.