FADML Scribe
Suyash Aakash (24BM6JP56)
7th April 2025

Remedy for Overfitting : Validation

In machine learning, our ultimate goal is to minimize the out-of-sample error (E_{out}), which reflects how well our model generalizes to unseen data. However, since $\mathbf{E_{out}}$ is not directly observable during training, we use strategies like regularization and validation to estimate and minimize it. Regularization controls model complexity by penalizing large weights, thereby reducing overfitting and is mathematically expressed as:

$$E_{\text{out}}(h) = E_{\text{in}}(h) + \underbrace{\text{overfit penalty}}_{\text{regularization estimates this quantity}},$$

This approach attempts to indirectly minimize $\mathbf{E_{out}}$ by keeping the penalty low. On the other hand, validation estimates $\mathbf{E_{out}}$ more directly by splitting the data into training and validation sets, training on one part, and evaluating on the other. The penalty term can arise due to various issues such as overfitting, poor model selection, or noise in the training data. Regularization restricts the solution space to within a bounded norm (like wT .w ≤ C) and minimizes the in-sample error within that constrained set. Validation complements this by emulating real-world deployment—by evaluating the model on unseen validation data. Together, regularization estimates the penalty term while validation estimates the full $\mathbf{E_{out}}$, and both work toward building a robust model that generalizes well.

What is Validation Set?

The validation set is a subset of data held out from the training set. Unlike the test set, which is used only at the final evaluation stage, the validation set plays an active role in the learning process—especially in decisions like hyper-parameter tuning, early stopping, and model selection.

The core idea is: since the validation set has not been used to train the model, its error can serve as a proxy for how the model will perform on unseen data—i.e., an estimator for the true generalization error **Eout**.While regularization estimates the penalty term, validation provides an empirical approximation of **Eout** directly:

Validation: **Eout**(h) ↓ (as close as possible).

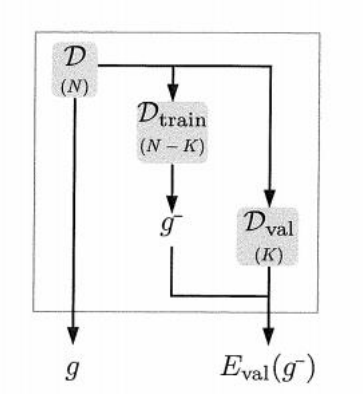
By splitting the dataset into training and validation sets, we can evaluate the model on unseen data and estimate how well it generalizes. The goal is to make

Ein ≈ Eout (Goal)

A small gap implies low over-fitting and good generalization. Validation is also used in hyper-parameter tuning, such as selecting the best value for the regularization parameter λ.

How does validation work?

Validation starts with the fundamental idea that validation data consists of points not used during model training. The error computed on these points helps estimate the out-of-sample error. The error function e(h(x), y) captures the deviation between predicted and actual values. For a single point, the expected out-of-sample error is $E_{out}(h) = E_x[e(h(x), y)]$, and its variance . Extending this idea to multiple points $\{(x_1, y_1), \ldots, (x_K, y_K)\}$, the empirical average over these errors gives the validation error Eval, which can also be viewed as an estimator of E_{out}. The more data points used for validation (larger K), the smaller the variance, leading to a more accurate estimate of generalization error. Thus, validation error gives practical insight into a model's real-world performance.



what a validation set is and how it differs from a test set. Although both sets are excluded from training, validation data can influence decisions during the learning process—such as selecting models or tuning hyper parameters—while test data should only be used at the final evaluation stage. The construction of a validation set is outlined: the dataset D is split into a training set $\mathbf{D_{train}}$ of size N − K and a validation set $\mathbf{D_{val}}$ of size K. Importantly, the method of splitting must not depend on the data values to maintain randomness and unbiasedness. The formula for computing validation error $\mathbf{E_{val(g)}}$ is given as the average error over the validation set. This section sets the stage for understanding how validation fits into the machine learning pipeline by providing a statistically sound means of evaluating model performance on unseen data

$$E_{\text{val}}(g^-) = \frac{1}{K} \sum_{\mathbf{x}_n \in \mathcal{D}_{\text{val}}} \mathsf{e}\left(g^-(\mathbf{x}_n), y_n\right),$$

Where **e(g(x), y)** is the point-wise error measure .For classification, **e(g(x), y) = g(x)≠ y** and for regression using squared error, **e(g(x), y)**

$$e(g(x), y) = (g(x) - y)^2.$$

The validation error is an unbiased estimate of $\mathbf{E_{out}}$ because the final hypothesis g − was created independently of the data points in the validation set. Indeed, taking the expectation of $\mathbf{E_{val}}$ with respect to the data points in $\mathbf{D_{val}}$,

$$\begin{aligned}\mathbb{E}_{\mathcal{D}_{\text{val}}}[E_{\text{val}}(g^-)] &= \frac{1}{K}\sum_{x_n \in \mathcal{D}_{\text{val}}} \mathbb{E}_{\mathcal{D}_{\text{val}}}[e(g^-(x_n), y_n)] \\ &= \frac{1}{K}\sum_{x_n \in \mathcal{D}_{\text{val}}} E_{\text{out}}(g^-) \\ &= E_{\text{out}}(g^-).\end{aligned}$$

The first step uses the linearity of expectation, and the second step follows because $\mathbf{e(g^-(x_n), y_n)}$ depends only on x_n, and so

$$\mathbb{E}_{\mathcal{D}_{\text{val}}}\left[\mathrm{e}\left(g^-(\mathbf{x}_n), y_n\right)\right] = \mathbb{E}_{\mathbf{x}_n}\left[\mathrm{e}\left(g^-(\mathbf{x}_n), y_n\right)\right] = E_{\text{out}}(g^-)$$

How reliable is $\mathbf{E_{val}}$ at estimating $\mathbf{E_{out}}$? In the case of classification, one can use the VC bound to predict how good the validation error is as an estimate for the out-of-sample error. We can view $\mathbf{D_{val}}$ as an 'in-sample' dataset on which we computed the error of the single hypothesis g −. We can thus apply the VC bound for a finite model with one hypothesis in it (using the Hoeffding bound). With high probability,

$$E_{\text{out}}(g^-) \leq E_{\text{val}}(g^-) + O\left(\tfrac{1}{\sqrt{K}}\right).$$

Q. How the variance of the validation error behaves as a function of the number of validation samples K, and why increasing the size of the validation set leads to a more reliable estimate of $\mathbf{E_{out}}$?

The variance of the error on a single validation point is given by

$$\mathrm{Var}\left[e(g(x_i), y_i)\right] = \sigma^2$$

This assumes that the prediction errors are independent and identically distributed (i.i.d.) with variance σ^2. Now, suppose we compute the validation error $\mathbf{E_{val}(g)}$ as the **average error** over K independent validation points. The variance of this average is given by:

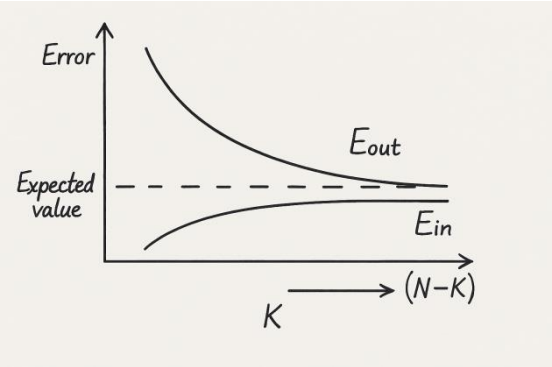
$$\mathrm{Var}\left[E_{\text{val}}(g)\right] = \frac{1}{K^2}\sum_{i=1}^{K} \mathrm{Var}\left[e(g(x_i), y_i)\right]$$

Since, each term has the same variance σ^2 we get the modified equation as:

$$\mathrm{Var}\left[E_{\mathrm{val}}(g)\right] = \frac{1}{K^2} \cdot K \cdot \sigma^2 = \frac{\sigma^2}{K}$$

The validation-accuracy trade off

Suppose we have N training data points and we use k of these points for our validation purposes. This leaves us with N-k data points to train our model. Now when we have a close look at the graph between our errors and the number of data-points we use for training, we get some interesting observations. Here,we observe that as we increase the k (and hence decrease N-k), the less accurate our model becomes worse. We estimate our out of sample error very well but we have a much less accurate model. Similarly for a small k, we train a better model but our estimation of out of sample error is bad.
The possible solution to this problem is that we use N-k data points for training, k data points for validation, and finally supply a model which was trained using all the N data points while using the out of sample error estimation of our previous model as the proxy for this model. As we can intuitively understand how with increasing k this proxy will become less and less accurate but practical experience has found out the values of k around ~ N/ 5 (80% training datasets and 20% test datasets) gives us good results , this is just a rule of thumb.

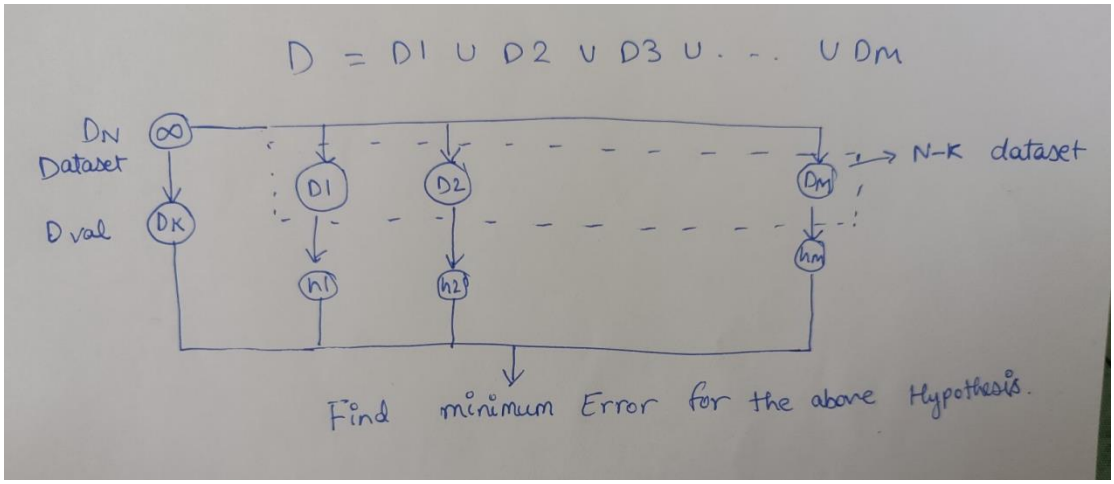


To understand this trade-off, imagine plotting two curves:

E_{in} (in-sample error): Error the model makes on the training data And **E_{out}** (true out-of-sample error): Expected error on unseen data.As we increase k (thus decrease N−k, i.e., training size becomes smaller), the model has less data to learn from. Consequently: E_{in} increases (worse training fit) and **E_{out}** decreases because we estimate it more reliably using more validation points.

In conclusion, if k is small (large training set), the model learns better (lower Ein , but the estimation of Eout becomes unreliable due to too few validation points.

Understanding Validation



To better understand how the validation process operates in model selection, let us assume that we are working with M different models, denoted as $H_1, H_2, \ldots, H_M$. These models are initially trained on a subset of the full dataset, called the training set $D_{\textbf{train}}$, which contains $N - k$ data points. Upon training, each model produces a hypothesis, resulting in M trained hypotheses: $h_1, h_2, h_3 \ldots..h_M$

Next, we evaluate the performance of each hypothesis using a separate validation set $D_{\textbf{val}}$, which contains the remaining k data points that were not included in training. For each hypothesis hi, we compute the validation error E_i, where $i = 1, 2, \ldots, M$. These errors serve as estimates of how well each hypothesis generalizes to unseen data. Among all $E_1, E_2, \ldots, E_M$, we select the hypothesis with the minimum validation error, denoted as E_{m*} . This corresponds to the model H_{m*} which appears to perform best based on validation.

To obtain the final hypothesis for deployment or evaluation, we retrain the selected model H_{m*} on the full datasets D, now using all N data points. This yields the final hypothesis h_{m*} ,which is expected to offer strong generalization due to optimal model choice and complete data utilization.

This entire workflow can be visualized as follows (refer to the diagram):

• The full datasets D is split into parts: $D_1, D_2, \ldots, D_M$

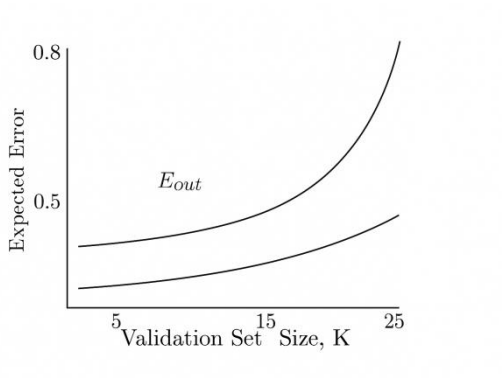
• Each part is used to train a model yielding hypotheses $h_1, h_2, \ldots, h_M$

• A validation set D_K (held out) is used to compute validation errors

• The hypothesis with the minimum validation error is identified

• That model is then retrained on the full dataset D to obtain h_{m*} (Best Hypothesis)

MODEL SELECTION



The graph illustrates the trade-off between training set size and validation set size (k)with respect to the estimated and actual out-of-sample errors. As k increases, the size of the validation set grows, which improves the accuracy of the validation error $\mathbf{E_{val}}$ as an estimator for $\mathbf{E_{out}}$**.** However, the training set becomes smaller, leading to poorer model performance and a rise in $\mathbf{E_{out}}$ for the selected hypothesis h_m* .The two curves—$\mathbf{E_{out}}$ and $\mathbf{E_{val}}$—eventually converge as k becomes large because the validation estimate becomes more reliable. However, due to less training data, the actual model performance drops. This demonstrates a key dilemma: larger k improves estimation accuracy but reduces learning performance. A common rule of thumb is to choose **k = N / 5** to balance both needs effectively.

Cross Validation

Let us consider a dataset:Dn = {(x_1, y_1), (x_2, y_2), . . . , (x_N , y_N)}Out of this, we hold out one point for validation and use the remaining N − 1 points for training.

$$\mathcal{D}_n = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n-1}, y_{n-1}), \cancel{(\mathbf{x}_n, y_n)}, (\mathbf{x}_{n+1}, y_{n+1}), \ldots, (\mathbf{x}_N, y_N)$$

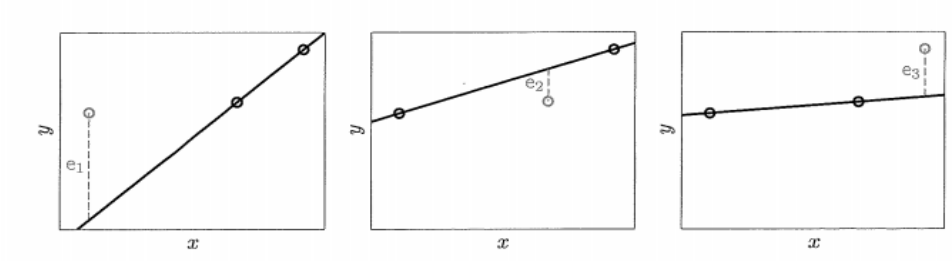
We now use cross-validation, particularly the leave-one-out variant (where K = 1):

Each time, one data point is set aside as validation.The model is trained on the remaining N − 1 data points.This process is repeated for all N data points, producing N validation errors e_1, e_2, . . . , e_N .

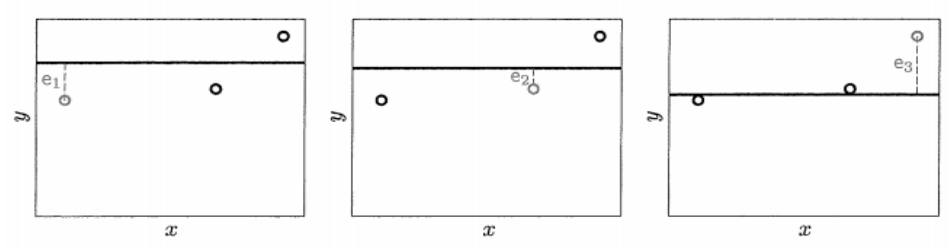
The cross-validation estimate of the out-of-sample error is:

$$E_{\text{cv}} = \frac{1}{N} \sum_{n=1}^{N} \mathrm{e}_n.$$

This technique gives a more reliable and balanced approximation of the true out-of-sample error, especially when the full datasets is small.



Leave one out cross validation: Linear Hypothesis Fitting



Leave one out cross validation: Constant Fit

we want to fit a linear hypothesis for these three points. The cross validation experiment suggests that we leave one point out, train a model for the other two and find the corresponding validation error in the manner shown in the above 5plots. Final the Cross Validation Error will be:

$$E_{CV} = \frac{e_1 + e_2 + e_3}{3}$$

Now we fit a constant hypothesis for these three points. The cross validation experiment suggests that we leave one point out, train a model for the other two and find the corresponding validation error in the manner shown in the above plots. Depending on the Cross Validation error of both the hypothesis we may decide

which hypothesis fits the datasets better. But, usually The linear model gives $\mathbf{E_{cv}}$ more error as compared to the constant model .

Also, Despite being more flexible in case of the linear model over-fits the small datasets. The constant model, though simpler, generalizes better in this case. Hence, cross-validation identifies the better model based on true generalization performance, not just in-sample fit.

Aspect	Linear Hypothesis	Constant Fit
Model Flexibility	Can capture trends	Cannot capture trends
Leave-One-Out Error Behavior	Varies more; sensitive to the subset	More consistent but may be high
Best when	Data has trend (linear relationship)	Data is centered around a constant mean
Bias-Variance	Lower bias, higher variance	Higher bias, lower variance

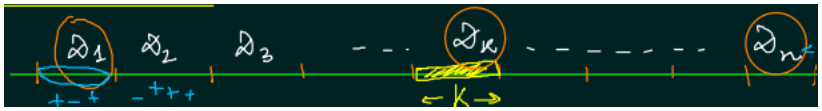
Why there is a need of K - Fold Validation?

When building machine learning models, it's important to evaluate how well your model will perform on unseen data. A simple way to do this is by splitting your dataset into a **training set** and a **validation set**—this is called **hold-out validation**. While it's quick and easy, it can be quite unreliable. The performance you observe heavily depends on how the data was split. If the split is unlucky (e.g., not representative), your evaluation will be misleading.

To fix this, we use **cross-validation**. In its most extreme form—**leave-one-out cross-validation (LOOCV)**—we leave out one data point at a time for validation and train on the rest. We repeat this for every point in the datasets, and then average the validation errors. This gives a much more stable and reliable estimate of the model's true performance.

But here's the catch: if you have a huge datasets, this means training the model N times, which can be computationally expensive and slow. Hence, we use another method called the **K-Fold Cross Validation** method to calculate Cross Validation Error.

KFold Cross Validation Method:



We first segregate the data into folds of equal number of points i.e. k number of points. Let the folds be $D_1, D_2, ..., D_k$ ..., D_n. The Common choices are 5 or 10.So if you pick K=5, we 'll split your datasets into 5 equal parts.

We randomly shuffle your datasets to mix up the order (so things don't get biased).Then, you split it into K parts—let's call them $D_1, D_2, ..., D_k$. Each part is called a fold, and they're all about the same size.

For each fold D_k:Use it **as your validation set** (the data you test the model on).Use all the other folds **as your training set** (the data you teach the model with).Train your model on the training data.Test it on the validation fold and record the error

Once we're done training and testing across all folds, we'll have K error values—one from each fold.We simply take the average of these errors. This gives you a final cross-validation score that tells us well about the model how likely to perform on unseen data.