12: Model Selection

Model Selection: Tuning Hyperparameters

Remember ERM

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\ell(h_{\mathbf{w}}(\mathbf{x}_i), y_i)}_{Loss} + \underbrace{\lambda r(w)}_{Regularizer}$$

Here, $\mathbf{w}$ denotes the *parameters* of the model, which are learned. And λ is a *hyper-parameter* that regulates bias/variance. Hyper-parameters are not learned and need to be "by hand". How do we know what good values of λ could be? A general rule is Occam's Razor, which generally states that *the simplest explanation is the best*. I.e. we are trying to find a model to explain our training data that is "as simple as possible, but not simpler".

Overfitting and Underfitting

There are two problematic cases which can arise when learning a classifier on a data set: underfitting and overfitting, each of which relate to the degree to which the data in the training set is extrapolated to apply to unknown data:

Underfitting: The classifier learned on the training set is not expressive enough (i.e. too simple) to even account for the data provided. In this case, both the training error and the test error will be high, as the classifier does not account for relevant information present in the training set.

Overfitting: The classifier learned on the training set is too specific, and cannot be used to accurately infer anything about unseen data. Although training error continues to decrease over time, test error will begin to increase again as the classifier begins to make decisions based on patterns which exist only in the training set and not in the broader distribution.

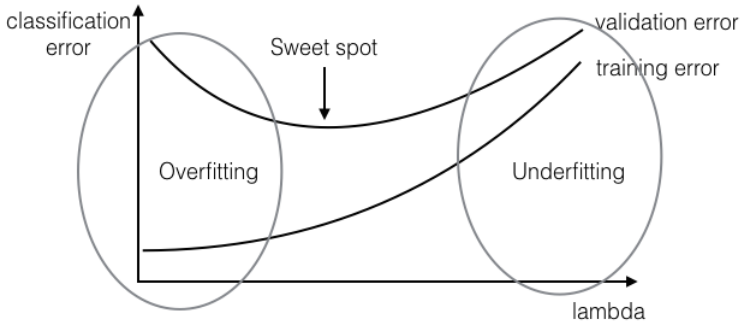


Figure 1: overfitting and underfitting

Identify the Sweet Spot

Divide data into training and validation portions. Train your algorithm on the "training" split and evaluate it on the "validation" split, for various value of λ (Typical values: 10^{-5} 10^{-4} 10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} ...).

k-fold cross validation

Divide your training data into k partitions. Train on $k-1$ of them and leave one out as validation set. Do this k times (i.e. leave out every partition exactly once) and average the validation error across runs. This gives you a good estimate of the validation error (even with standard deviation). In the extreme case, you can have $k = n$, i.e. you only leave a single data point out (this is often referred to as LOOCV- Leave One Out Cross Validation).

LOOCV is important if your data set is small and cannot afford to leave out many data points for evaluation .

Telescopic search

Do two searches: 1st, find the best order of magnitude for λ; 2nd, do a more fine-grained search around the best λ found so far. For example, first you try $\lambda = 0.01, 0.1, 1, 10, 100$. It turns out 10 is the best performing value. Then you try out $\lambda = 5, 10, 15, 20, 25, \ldots, 95$ to test values "around" 10.

Grid and Random search

If you have multiple parameters (e.g. λ and also the kernel width σ in case you are using a kernel) a simple way to find the best value of both of them is to fix a set of values for each hyper-parameter and try out every combination. One downside of this method is that the number of settings you need to try out grows exponentially with the number of hyper-parameters. Also, if your model is insensitive to one of the parameters, you waste a lot of computation by trying out many different settings for it. An variant of grid-search is random search. Instead of selecting hyper-parameters on a pre-defined grid, we select them randomly within pre-defined intervals (See Figure 2). One advantage is that if for example the algorithm is somewhat insensitive to exact values of λ and sensitive to σ, then by performing e.g. 6×6 grid search you will learn little from varying λ six times and you only explore six values of σ. However, if you do random search, again the changes in λ may not matter too much, but now you are exploring 36 (!) different values of σ.

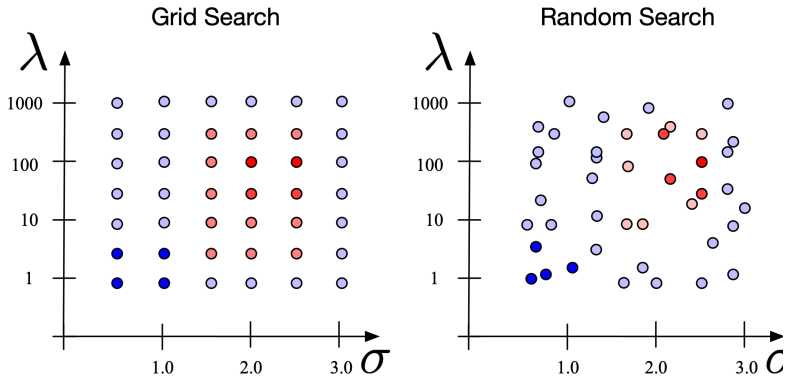
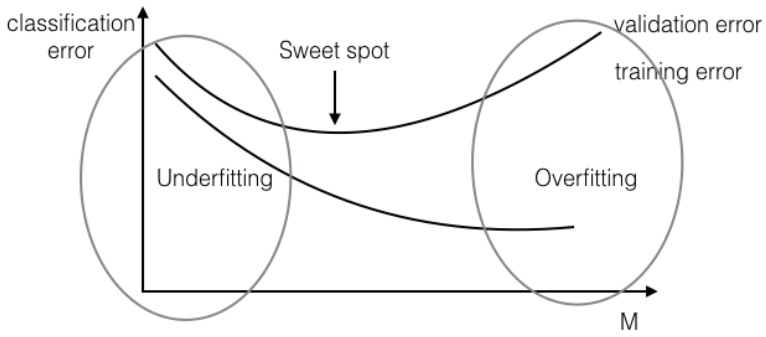


Figure 2: Grid Search vs Random search (red indicates lower loss, blue indicates high loss). One advantage of Random Search is that significantly more values of each individual hyper-parameter are explored.

Early Stopping

Stop your optimization after M (>= 0) number of gradient steps, even if optimization has not converged yet.



Cross-Validation Notes

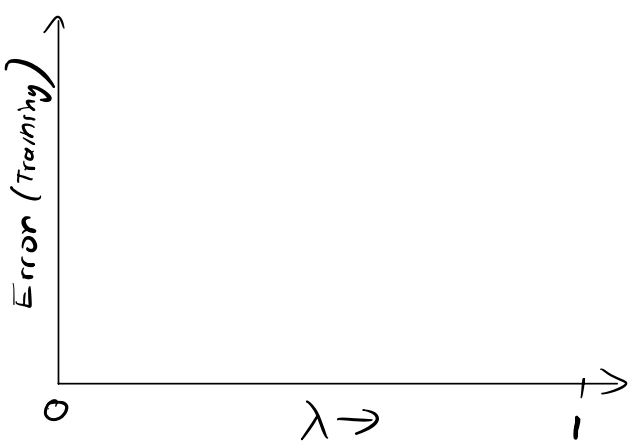
$\lambda = \{0.01, 0.02, 0.03, \ldots, 1\}$

$$\hat{w}_\lambda, \hat{b}_\lambda = \underset{w,b}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i + b - y_i)^2 + \lambda \|w\|_2^2$$

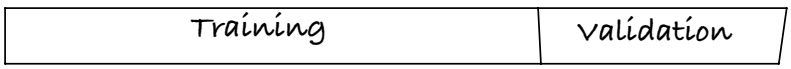
How do we pick λ ?

Q1. Pick λ that minimizes training error, is this a good strategy?

Q2. What does the curve of trining error Vs λ look like?



Validation Set



1. If validation set is too small we dont get good estimate of error
2. But if it is very big then training set size is small

K- fold cross validation

Partition data into K folds $D_1,\ldots,D_K$

For $\lambda \in \{0.01, 0.02, \ldots, 1\}$

  For $k = 1$ to K:

    W_k, b_k = Ridge regression(D_{-k}, λ)

    $\varepsilon_{k,\lambda} =$

  $\varepsilon_\lambda = \frac{1}{K}\sum_{k=1}^{K} \varepsilon_{k,\lambda}$ $\quad \frac{1}{|D_k|} \sum_{(x,y)\in D_k} (w_k^T x - y)^2$

$\lambda^* = \arg\min_\lambda \varepsilon_\lambda$

% train on all data But D_k

% evaluate on D_k

% Take average validation error

% Pick best λ