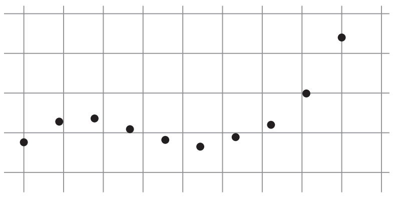
11

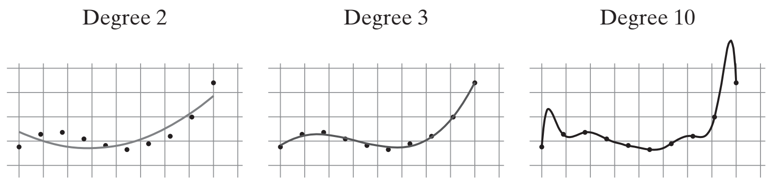
Model Selection and Validation

In the previous chapter we have described the AdaBoost algorithm and have shown how the parameter T of AdaBoost controls the bias-complexity tradeoff. But how do we set T in practice? More generally, when approaching some practical problem, we usually can think of several algorithms that may yield a good solution, each of which might have several parameters. How can we choose the best algorithm for the particular problem at hand? And how do we set the algorithm's parameters? This task is often called *model selection*.

To illustrate the model selection task, consider the problem of learning a one dimensional regression function, $h : \mathbb{R} \to \mathbb{R}$. Suppose that we obtain a training set as depicted in the figure.



We can consider fitting a polynomial to the data, as described in Chapter 9. However, we might be uncertain regarding which degree d would give the best results for our data set: A small degree may not fit the data well (i.e., it will have a large approximation error), whereas a high degree may lead to overfitting (i.e., it will have a large estimation error). In the following we depict the result of fitting a polynomial of degrees 2, 3, and 10. It is easy to see that the empirical risk decreases as we enlarge the degree. However, looking at the graphs, our intuition tells us that setting the degree to 3 may be better than setting it to 10. It follows that the empirical risk alone is not enough for model selection.



In this chapter we will present two approaches for model selection. The first approach is based on the Structural Risk Minimization (SRM) paradigm we have described and analyzed in Chapter 7.2. SRM is particularly useful when a learning algorithm depends on a parameter that controls the bias-complexity tradeoff (such as the degree of the fitted polynomial in the preceding example or the parameter T in AdaBoost). The second approach relies on the concept of *validation*. The basic idea is to partition the training set into two sets. One is used for training each of the candidate models, and the second is used for deciding which of them yields the best results.

In model selection tasks, we try to find the right balance between approximation and estimation errors. More generally, if our learning algorithm fails to find a predictor with a small risk, it is important to understand whether we suffer from overfitting or underfitting. In Section 11.3 we discuss how this can be achieved.

11.1 MODEL SELECTION USING SRM

The SRM paradigm has been described and analyzed in Section 7.2. Here we show how SRM can be used for tuning the tradeoff between bias and complexity without deciding on a specific hypothesis class in advance. Consider a countable sequence of hypothesis classes $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \ldots$. For example, in the problem of polynomial regression mentioned, we can take $\mathcal{H}_d$ to be the set of polynomials of degree at most d. Another example is taking $\mathcal{H}_d$ to be the class $L(B,d)$ used by AdaBoost, as described in the previous chapter.

We assume that for every d, the class $\mathcal{H}_d$ enjoys the uniform convergence property (see Definition 4.3 in Chapter 4) with a sample complexity function of the form

$$m^{\mathrm{UC}}_{\mathcal{H}_d}(\epsilon,\delta) \le \frac{g(d)\log(1/\delta)}{\epsilon^2}, \tag{11.1}$$

where $g:\mathbb{N}\to\mathbb{R}$ is some monotonically increasing function. For example, in the case of binary classification problems, we can take $g(d)$ to be the VC-dimension of the class $\mathcal{H}_d$ multiplied by a universal constant (the one appearing in the fundamental theorem of learning; see Theorem 6.8). For the classes $L(B,d)$ used by AdaBoost, the function g will simply grow with d.

Recall that the SRM rule follows a "bound minimization" approach, where in our case the bound is as follows: With probability of at least $1-\delta$, for every $d \in \mathbb{N}$ and $h \in \mathcal{H}_d$,

$$L_{\mathcal{D}}(h) \le L_S(h) + \sqrt{\frac{g(d)(\log(1/\delta) + 2\log(d) + \log(\pi^2/6))}{m}}. \tag{11.2}$$

This bound, which follows directly from Theorem 7.4, shows that for every d and every $h \in \mathcal{H}_d$, the true risk is bounded by two terms – the empirical risk, $L_S(h)$, and a complexity term that depends on d. The SRM rule will search for d and $h \in \mathcal{H}_d$ that minimize the right-hand side of Equation (11.2).

Getting back to the example of polynomial regression described earlier, even though the empirical risk of the 10th degree polynomial is smaller than that of the 3rd degree polynomial, we would still prefer the 3rd degree polynomial since its complexity (as reflected by the value of the function $g(d)$) is much smaller.

While the SRM approach can be useful in some situations, in many practical cases the upper bound given in Equation (11.2) is pessimistic. In the next section we present a more practical approach.

11.2 VALIDATION

We would often like to get a better estimation of the true risk of the output predictor of a learning algorithm. So far we have derived bounds on the estimation error of a hypothesis class, which tell us that for *all* hypotheses in the class, the true risk is not very far from the empirical risk. However, these bounds might be loose and pessimistic, as they hold for all hypotheses and all possible data distributions. A more accurate estimation of the true risk can be obtained by using some of the training data as a validation set, over which one can evalutate the success of the algorithm's output predictor. This procedure is called *validation*.

Naturally, a better estimation of the true risk is useful for model selection, as we will describe in Section 11.2.2.

11.2.1 Hold Out Set

The simplest way to estimate the true error of a predictor h is by sampling an additional set of examples, independent of the training set, and using the empirical error on this validation set as our estimator. Formally, let $V = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{m_v}, y_{m_v})$ be a set of fresh m_v examples that are sampled according to $\mathcal{D}$ (independently of the m examples of the training set S). Using Hoeffding's inequality (Lemma 4.5) we have the following:

Theorem 11.1. *Let h be some predictor and assume that the loss function is in* $[0,1]$. *Then, for every $\delta \in (0,1)$, with probability of at least $1-\delta$ over the choice of a validation set V of size m_v we have*

$$\left|L_V(h) - L_{\mathcal{D}}(h)\right| \leq \sqrt{\frac{\log(2/\delta)}{2m_v}}.$$

The bound in Theorem 11.1 does not depend on the algorithm or the training set used to construct h and is tighter than the usual bounds that we have seen so far. The reason for the tightness of this bound is that it is in terms of an estimate on a fresh validation set that is independent of the way h was generated. To illustrate this point, suppose that h was obtained by applying an ERM predictor with respect to a hypothesis class of VC-dimension d, over a training set of m examples. Then, from the fundamental theorem of learning (Theorem 6.8) we obtain the bound

$$L_{\mathcal{D}}(h) \le L_S(h) + \sqrt{C\,\frac{d+\log(1/\delta)}{m}},$$

where C is the constant appearing in Theorem 6.8. In contrast, from Theorem 11.1 we obtain the bound

$$L_{\mathcal{D}}(h) \le L_V(h) + \sqrt{\frac{\log(2/\delta)}{2m_v}}.$$

Therefore, taking m_v to be order of m, we obtain an estimate that is more accurate by a factor that depends on the VC-dimension. On the other hand, the price we pay for using such an estimate is that it requires an additional sample on top of the sample used for training the learner.

Sampling a training set and then sampling an independent validation set is equivalent to randomly partitioning our random set of examples into two parts, using one part for training and the other one for validation. For this reason, the validation set is often referred to as a *hold out* set.

11.2.2 Validation for Model Selection

Validation can be naturally used for model selection as follows. We first train different algorithms (or the same algorithm with different parameters) on the given training set. Let $\mathcal{H} = \{h_1, \ldots, h_r\}$ be the set of all output predictors of the different algorithms. For example, in the case of training polynomial regressors, we would have each h_r be the output of polynomial regression of degree r. Now, to choose a single predictor from $\mathcal{H}$ we sample a fresh validation set and choose the predictor that minimizes the error over the validation set. In other words, we apply $\mathrm{ERM}_{\mathcal{H}}$ over the validation set.

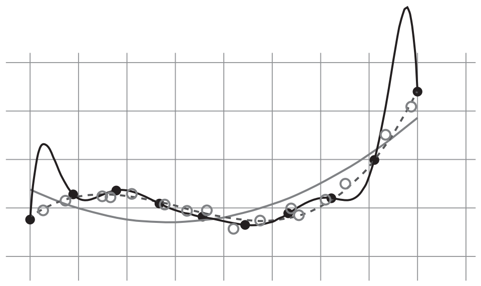
This process is very similar to learning a finite hypothesis class. The only difference is that $\mathcal{H}$ is not fixed ahead of time but rather depends on the training set. However, since the validation set is independent of the training set we get that it is also independent of $\mathcal{H}$ and therefore the same technique we used to derive bounds for finite hypothesis classes holds here as well. In particular, combining Theorem 11.1 with the union bound we obtain:

Theorem 11.2. *Let $\mathcal{H} = \{h_1, \ldots, h_r\}$ be an arbitrary set of predictors and assume that the loss function is in $[0,1]$. Assume that a validation set V of size m_v is sampled independent of $\mathcal{H}$. Then, with probability of at least $1-\delta$ over the choice of V we have*

$$\forall h \in \mathcal{H},\ \left|L_{\mathcal{D}}(h) - L_V(h)\right| \le \sqrt{\frac{\log(2|\mathcal{H}|/\delta)}{2m_v}}.$$

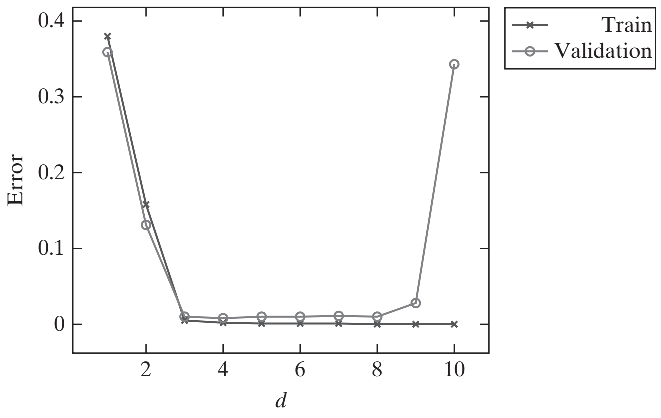
This theorem tells us that the error on the validation set approximates the true error as long as $\mathcal{H}$ is not too large. However, if we try too many methods (resulting in $|\mathcal{H}|$ that is large relative to the size of the validation set) then we're in danger of overfitting.

To illustrate how validation is useful for model selection, consider again the example of fitting a one dimensional polynomial as described in the beginning of this chapter. In the following we depict the same training set, with ERM polynomials of degree 2, 3, and 10, but this time we also depict an additional validation set (marked as unfilled circles). The polynomial of degree 10 has minimal training error, yet the polynomial of degree 3 has the minimal validation error, and hence it will be chosen as the best model.



11.2.3 The Model-Selection Curve

The model selection curve shows the training error and validation error as a function of the complexity of the model considered. For example, for the polynomial fitting problem mentioned previously, the curve will look like:



As can be shown, the training error is monotonically decreasing as we increase the polynomial degree (which is the complexity of the model in our case). On the other hand, the validation error first decreases but then starts to increase, which indicates that we are starting to suffer from overfitting.

Plotting such curves can help us understand whether we are searching the correct regime of our parameter space. Often, there may be more than a single parameter to tune, and the possible number of values each parameter can take might be quite large. For example, in Chapter 13 we describe the concept of *regularization*, in which the parameter of the learning algorithm is a real number. In such cases, we start with a rough grid of values for the parameter(s) and plot the corresponding model-selection curve. On the basis of the curve we will zoom in to the correct regime and employ a finer grid to search over. It is important to verify that we are in the relevant regime. For example, in the polynomial fitting problem described, if we start searching degrees from the set of values $\{1, 10, 20\}$ and do not employ a finer grid based on the resulting curve, we will end up with a rather poor model.

11.2.4 *k*-Fold Cross Validation

The validation procedure described so far assumes that data is plentiful and that we have the ability to sample a fresh validation set. But in some applications, data is scarce and we do not want to "waste" data on validation. The k-fold cross validation technique is designed to give an accurate estimate of the true error without wasting too much data.

In k-fold cross validation the original training set is partitioned into k subsets (folds) of size m/k (for simplicity, assume that m/k is an integer). For each fold, the algorithm is trained on the union of the other folds and then the error of its output is estimated using the fold. Finally, the average of all these errors is the estimate of the true error. The special case $k = m$, where m is the number of examples, is called *leave-one-out* (LOO).

k-Fold cross validation is often used for model selection (or parameter tuning), and once the best parameter is chosen, the algorithm is retrained using this parameter on the entire training set. A pseudocode of k-fold cross validation for model selection is given in the following. The procedure receives as input a training set, S, a set of possible parameter values, Θ, an integer, k, representing the number of folds, and a learning algorithm, A, which receives as input a training set as well as a parameter $\theta \in \Theta$. It outputs the best parameter as well as the hypothesis trained by this parameter on the entire training set.

k-Fold Cross Validation for Model Selection

```
input:
    training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
    set of parameter values $\Theta$
    learning algorithm $A$
    integer $k$
partition $S$ into $S_1, S_2, \ldots, S_k$
foreach $\theta \in \Theta$
    for $i = 1 \ldots k$
        $h_{i,\theta} = A(S \setminus S_i; \theta)$
    error($\theta$) $= \frac{1}{k} \sum_{i=1}^{k} L_{S_i}(h_{i,\theta})$
output
  $\theta^\star = \mathrm{argmin}_\theta\,[\mathrm{error}(\theta)]$
  $h_{\theta^\star} = A(S; \theta^\star)$
```

The cross validation method often works very well in practice. However, it might sometime fail, as the artificial example given in Exercise 11.1 shows. Rigorously understanding the exact behavior of cross validation is still an open problem. Rogers and Wagner (Rogers & Wagner 1978) have shown that for k local rules (e.g., k Nearest Neighbor; see Chapter 19) the cross validation procedure gives a very good estimate of the true error. Other papers show that cross validation works for stable algorithms (we will study stability and its relation to learnability in Chapter 13).

11.2.5 Train-Validation-Test Split

In most practical applications, we split the available examples into three sets. The first set is used for training our algorithm and the second is used as a validation set for model selection. After we select the best model, we test the performance of the output predictor on the third set, which is often called the "test set." The number obtained is used as an estimator of the true error of the learned predictor.

11.3 WHAT TO DO IF LEARNING FAILS

Consider the following scenario: You were given a learning task and have approached it with a choice of a hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on a test set. The test results, unfortunately, turn out to be unsatisfactory. What went wrong then, and what should you do next?

There are many elements that can be "fixed." The main approaches are listed in the following:

- Get a larger sample
- Change the hypothesis class by
  - Enlarging it
  - Reducing it
  - Completely changing it
  - Changing the parameters you consider
- Change the feature representation of the data
- Change the optimization algorithm used to apply your learning rule

In order to find the best remedy, it is essential first to understand the cause of the bad performance. Recall that in Chapter 5 we decomposed the true error of the learned predictor into approximation error and estimation error. The approximation error is defined to be $L_{\mathcal{D}}(h^\star)$ for some $h^\star \in \operatorname{argmin}_{h\in\mathcal{H}} L_{\mathcal{D}}(h)$, while the estimation error is defined to be $L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^\star)$, where h_S is the learned predictor (which is based on the training set S).

The approximation error of the class does not depend on the sample size or on the algorithm being used. It only depends on the distribution $\mathcal{D}$ and on the hypothesis class $\mathcal{H}$. Therefore, if the approximation error is large, it will not help us to enlarge the training set size, and it also does not make sense to reduce the hypothesis class. What can be beneficial in this case is to enlarge the hypothesis class or completely change it (if we have some alternative prior knowledge in the form of a different hypothesis class). We can also consider applying the same hypothesis class but on a different feature representation of the data (see Chapter 25).

The estimation error of the class does depend on the sample size. Therefore, if we have a large estimation error we can make an effort to obtain more training examples. We can also consider reducing the hypothesis class. However, it doesn't make sense to enlarge the hypothesis class in that case.

Error Decomposition Using Validation

We see that understanding whether our problem is due to approximation error or estimation error is very useful for finding the best remedy. In the previous section we saw how to estimate $L_{\mathcal{D}}(h_S)$ using the empirical risk on a validation set. However, it is more difficult to estimate the approximation error of the class. Instead, we give a different error decomposition, one that can be estimated from the train and validation sets.

$$L_{\mathcal{D}}(h_S) = (L_{\mathcal{D}}(h_S) - L_V(h_S)) + (L_V(h_S) - L_S(h_S)) + L_S(h_S).$$

The first term, $(L_{\mathcal{D}}(h_S) - L_V(h_S))$, can be bounded quite tightly using Theorem 11.1. Intuitively, when the second term, $(L_V(h_S) - L_S(h_S))$, is large we say that our algorithm suffers from "overfitting" while when the empirical risk term, $L_S(h_S)$, is large we say that our algorithm suffers from "underfitting." Note that these two terms are not necessarily good estimates of the estimation and approximation errors. To illustrate this, consider the case in which $\mathcal{H}$ is a class of VC-dimension d, and $\mathcal{D}$ is a distribution such that the approximation error of $\mathcal{H}$ with respect to $\mathcal{D}$ is 1/4. As long as the size of our training set is smaller than d we will have $L_S(h_S) = 0$ for every ERM hypothesis. Therefore, the training risk, $L_S(h_S)$, and the approximation error, $L_{\mathcal{D}}(h^\star)$, can be significantly different. Nevertheless, as we show later, the values of $L_S(h_S)$ and $(L_V(h_S) - L_S(h_S))$ still provide us useful information.

Consider first the case in which $L_S(h_S)$ is large. We can write

$$L_S(h_S) = (L_S(h_S) - L_S(h^\star)) + (L_S(h^\star) - L_{\mathcal{D}}(h^\star)) + L_{\mathcal{D}}(h^\star).$$

When h_S is an ERM$_{\mathcal{H}}$ hypothesis we have that $L_S(h_S) - L_S(h^\star) \le 0$. In addition, since $h^\star$ does not depend on S, the term $(L_S(h^\star) - L_{\mathcal{D}}(h^\star))$ can be bounded quite tightly (as in Theorem 11.1). The last term is the approximation error. It follows that if $L_S(h_S)$ is large then so is the approximation error, and the remedy to the failure of our algorithm should be tailored accordingly (as discussed previously).

Remark 11.1. It is possible that the approximation error of our class is small, yet the value of $L_S(h_S)$ is large. For example, maybe we had a bug in our ERM implementation, and the algorithm returns a hypothesis h_S that is not an ERM. It may also be the case that finding an ERM hypothesis is computationally hard, and our algorithm applies some heuristic trying to find an approximate ERM. In some cases, it is hard to know how good h_S is relative to an ERM hypothesis. But, sometimes it is possible at least to know whether there are better hypotheses. For example, in the next chapter we will study convex learning problems in which there are optimality conditions that can be checked to verify whether our optimization algorithm converged to an ERM solution. In other cases, the solution may depend on randomness in initializing the algorithm, so we can try different randomly selected initial points to see whether better solutions pop out.

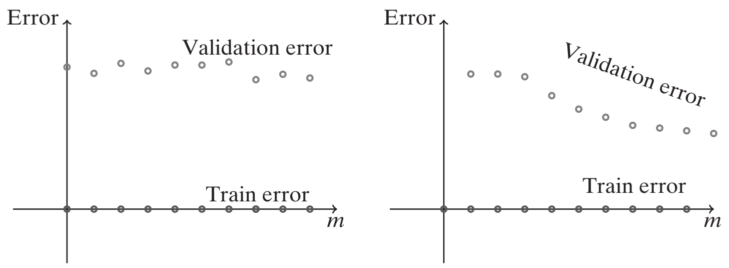


Figure 11.1. Examples of learning curves. Left: This learning curve corresponds to the scenario in which the number of examples is always smaller than the VC dimension of the class. Right: This learning curve corresponds to the scenario in which the approximation error is zero and the number of examples is larger than the VC dimension of the class.

Next consider the case in which $L_S(h_S)$ is small. As we argued before, this does not necessarily imply that the approximation error is small. Indeed, consider two scenarios, in both of which we are trying to learn a hypothesis class of VC-dimension d using the ERM learning rule. In the first scenario, we have a training set of $m < d$ examples and the approximation error of the class is high. In the second scenario, we have a training set of $m > 2d$ examples and the approximation error of the class is zero. In both cases $L_S(h_S) = 0$. How can we distinguish between the two cases?

Learning Curves

One possible way to distinguish between the two cases is by plotting *learning curves*. To produce a learning curve we train the algorithm on prefixes of the data of increasing sizes. For example, we can first train the algorithm on the first 10% of the examples, then on 20% of them, and so on. For each prefix we calculate the training error (on the prefix the algorithm is being trained on) and the validation error (on a predefined validation set). Such learning curves can help us distinguish between the two aforementioned scenarios. In the first scenario we expect the validation error to be approximately 1/2 for all prefixes, as we didn't really learn anything. In the second scenario the validation error will start as a constant but then should start decreasing (it must start decreasing once the training set size is larger than the VC-dimension). An illustration of the two cases is given in Figure 11.1.

In general, as long as the approximation error is greater than zero we expect the training error to grow with the sample size, as a larger amount of data points makes it harder to provide an explanation for all of them. On the other hand, the validation error tends to decrease with the increase in sample size. If the VC-dimension is finite, when the sample size goes to infinity, the validation and train errors converge to the approximation error. Therefore, by extrapolating the training and validation curves we can try to guess the value of the approximation error, or at least to get a rough estimate on an interval in which the approximation error resides.

Getting back to the problem of finding the best remedy for the failure of our algorithm, if we observe that $L_S(h_S)$ is small while the validation error is large, then in any case we know that the size of our training set is not sufficient for learning the class $\mathcal{H}$. We can then plot a learning curve. If we see that the validation error is starting to decrease then the best solution is to increase the number of examples (if we can afford to enlarge the data). Another reasonable solution is to decrease the complexity of the hypothesis class. On the other hand, if we see that the validation error is kept around 1/2 then we have no evidence that the approximation error of $\mathcal{H}$ is good. It may be the case that increasing the training set size will not help us at all. Obtaining more data can still help us, as at some point we can see whether the validation error starts to decrease or whether the training error starts to increase. But, if more data is expensive, it may be better first to try to reduce the complexity of the hypothesis class.

To summarize the discussion, the following steps should be applied:

1. If learning involves parameter tuning, plot the model-selection curve to make sure that you tuned the parameters appropriately (see Section 11.2.3).
2. If the training error is excessively large consider enlarging the hypothesis class, completely change it, or change the feature representation of the data.
3. If the training error is small, plot learning curves and try to deduce from them whether the problem is estimation error or approximation error.
4. If the approximation error seems to be small enough, try to obtain more data. If this is not possible, consider reducing the complexity of the hypothesis class.
5. If the approximation error seems to be large as well, try to change the hypothesis class or the feature representation of the data completely.

11.4 SUMMARY

Model selection is the task of selecting an appropriate model for the learning task based on the data itself. We have shown how this can be done using the SRM learning paradigm or using the more practical approach of validation. If our learning algorithm fails, a decomposition of the algorithm's error should be performed using learning curves, so as to find the best remedy.

11.5 EXERCISES

11.1 **Failure of k-fold cross validation** Consider a case in that the label is chosen at random according to $\mathbb{P}[y=1]=\mathbb{P}[y=0]=1/2$. Consider a learning algorithm that outputs the constant predictor $h(\mathbf{x})=1$ if the parity of the labels on the training set is 1 and otherwise the algorithm outputs the constant predictor $h(\mathbf{x})=0$. Prove that the difference between the leave-one-out estimate and the true error in such a case is always 1/2.

11.2 Let $\mathcal{H}_1,\ldots,\mathcal{H}_k$ be k hypothesis classes. Suppose you are given m i.i.d. training examples and you would like to learn the class $\mathcal{H}=\cup_{i=1}^k\mathcal{H}_i$. Consider two alternative approaches:

- Learn $\mathcal{H}$ on the m examples using the ERM rule
- Divide the m examples into a training set of size $(1-\alpha)m$ and a validation set of size αm, for some $\alpha\in(0,1)$. Then, apply the approach of model selection using validation. That is, first train each class $\mathcal{H}_i$ on the $(1-\alpha)m$ training examples using the ERM rule with respect to $\mathcal{H}_i$, and let $\hat{h}_1,\ldots,\hat{h}_k$ be the resulting hypotheses. Second, apply the ERM rule with respect to the finite class $\{\hat{h}_1,\ldots,\hat{h}_k\}$ on the αm validation examples.

Describe scenarios in which the first method is better than the second and vice versa.