7 Model selection and validation

Date	@November 5, 2024
Topic	Theory

Model selection and validation

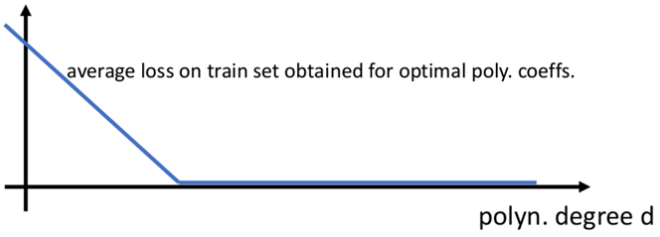
You can select different models by selecting the exponent of the polynomials, as seen in the previous lessons. Each model with a greater degree includes, in terms of sets, a model with a smaller degree. In fact, we talk speak about nested models.

$$\mathcal{H}^{(0)} \subseteq \mathcal{H}^{(1)} \subseteq \mathcal{H}^{(2)} \subseteq \mathcal{H}^{(3)} \subseteq \ldots$$

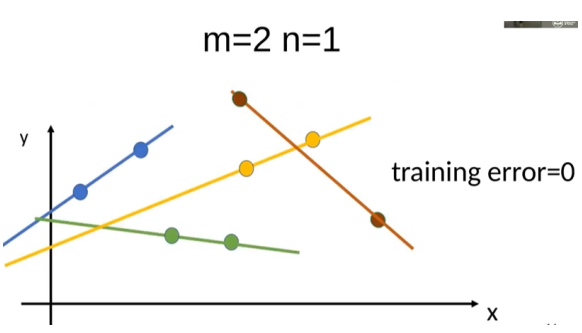
Train error vs degree

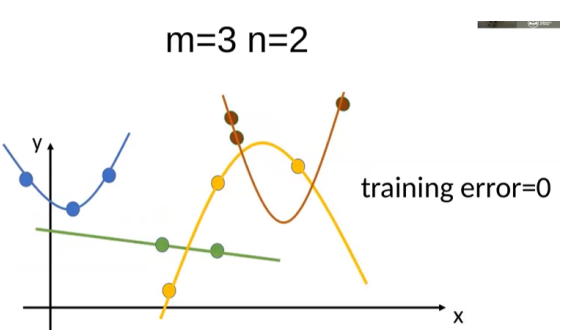
If you're considering polynomials you can almost always reach a null empirical risk. This is because if you have 10 points, a polynomial that goes exactly through them will be a 9 degree polynomial. Thus, if you consider the degree of the polynomial , which is the size of H, and then plotting the empirical loss for each different degree, you reach a point in which increasing the degree will lead to 0.

So we can perfectly fit (almost) any m data points using polynomials of degree n as soon as $n \geq m - 1$.

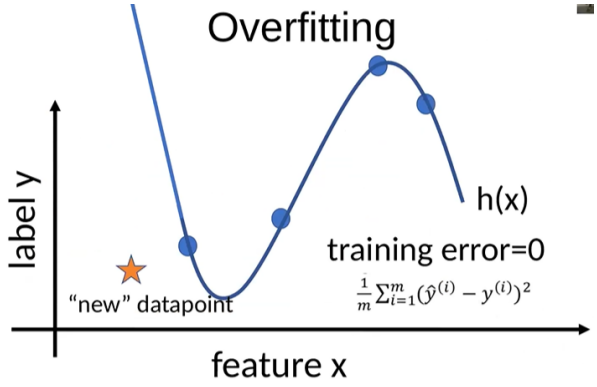


For example:





Choosing a model that is large enough to reach training error = 0 is not always desirable.



In the example above we've chosen a degree 4 polynomial that allows us to obtain a training error equal to 0. However, we want our model to be applied to samples for which we don't know the data and if we add a new data point the loss on it will be huge.

Small training error does not guarantee good performance outside the training set, that means for new/other data points. Small training error only indicates that we have solved the ERM optimization problem. Obtaining training error = 0 just means that the model is flexible enough and that the algorithm is working well to minimize ERM.

Ps: training error is empirical risk on training data after ERM.

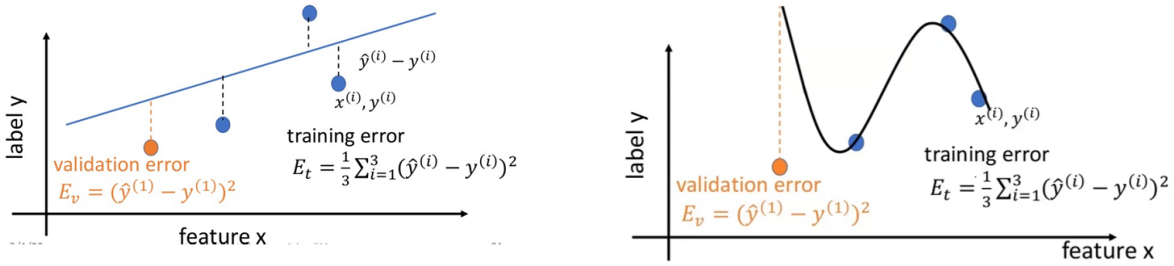
Basic idea of validation

We can divide data points into two sets, the training set, used to learn a hypothesis, and the validation set, used to probe outside training set, that is estimating the expected loss.

Regarding the previous example, the blue points, used to minimize the ERM, are the training set, while the new data point (the star), used to estimate the loss on new points, is the validation set.

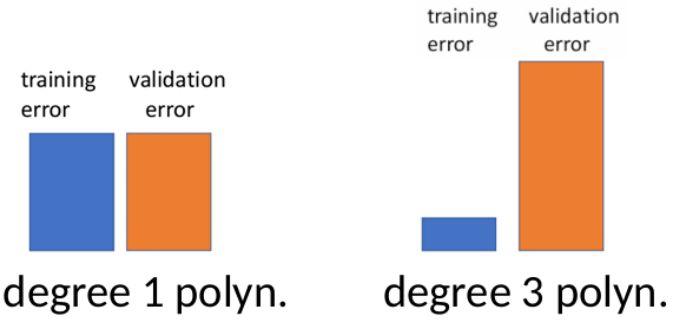
In python:

So we will construct two ERMs, one on training and one on validation, for each model.

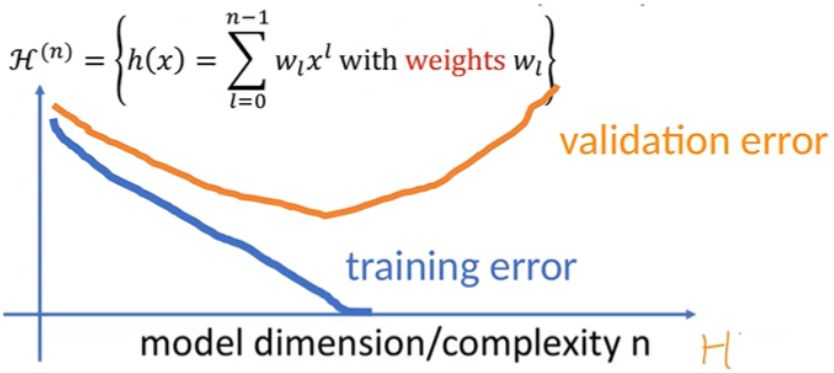


Then we could obtain these “graphs” that show that if we are taking the ERM on the training set we have the blue bars and for degree 1 polynomials we obtain a worst training error, while if we are taking the EMR on validation set we can see that the validation error is bigger for degree 3 polynomial. This means that degree 3 polynomial is reaching lower error on training because the model is more flexible and it can adapt to the specific points. However, when we observe the performance on you points we have the orange bar and we can see that the smaller model works better for new points. This demonstrate that the ERM on training is not a reliable estimation of the expected lost. Since we want a reliable estimation we will need to say that the degree 1 polynomials are better than the degree 3 polynomials.

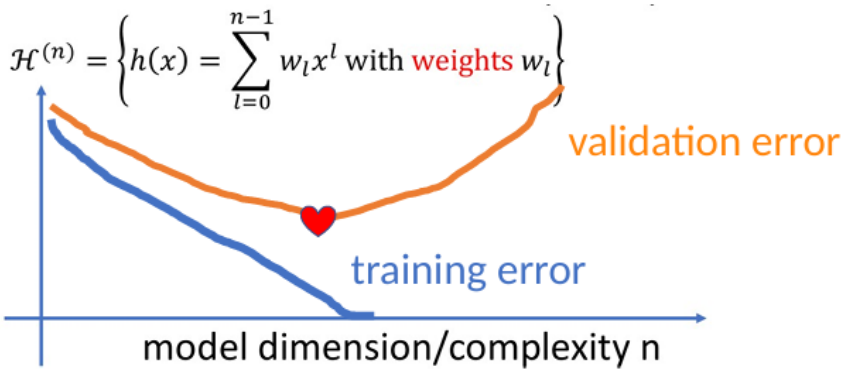
Degree 3 polynomials problem, in this case, is known as overfitting, because we have a model which is too flexible and that can fit very well to training data, but it performs poorly on the validation data.



Training/validation error vs model complexity



We can see that increasing the model size (on the x-axis) the training error tends to be 0, but larger the model size, worst the validation error. This means that the model that we need to choose is the one that allows us to obtain the hypothesis for which we obtain the minimum empirical risk on validation.



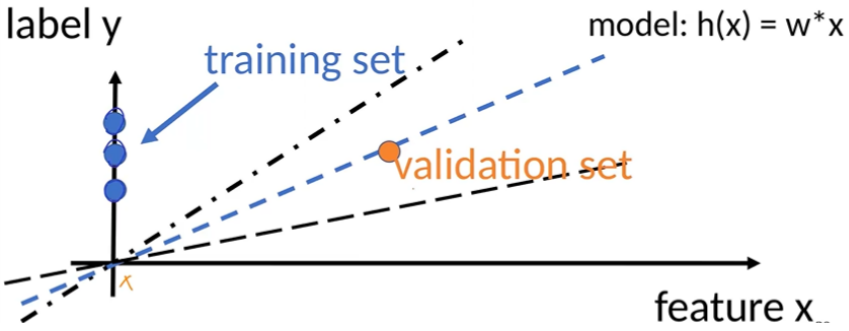
When the errors are diverging we speak about overfitting, while when they converge we speak about underfitting, and, in absolute values, they are very large.

In python:

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.4, random_state=42)
```

```
model.fit(X_train, y_train)
training_error = mean_squared_error(y_train,model.predict(X_train))
validation_error = mean_squared_error(y_val,model.predict(X_val))
```

Unlucky split into train and val set



In some case, since the splitting is random, you may have some unlucky splits, for example in this case, where al the training set share the same x.

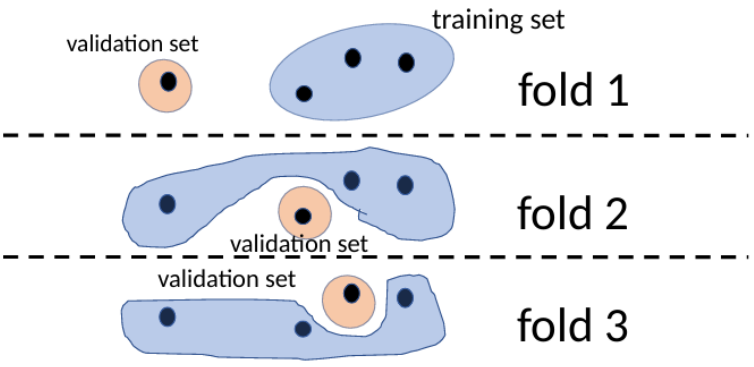
k-fold cross validation

One solution is to perform the splitting multiple time, and it is called k-fold cross validation. The idea is to repeat training and validation procedures multiple times.

sklearn.model_selection.KFold

class sklearn.model_selection.**KFold**(*n_splits=5, *, shuffle=False, random_state=None*)

For example, a 3-fold cross validation goes as follows:



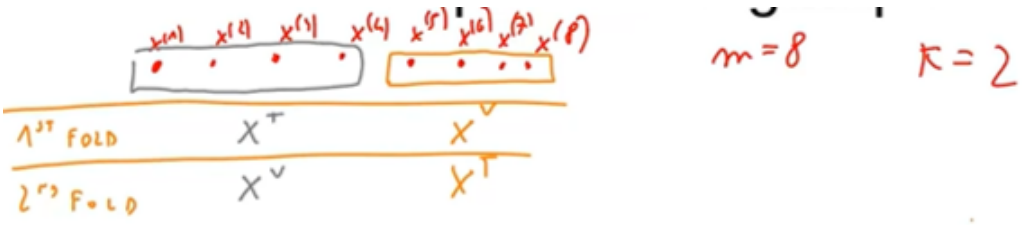
This means you're solving the ERM on different folds, obtaining different h^*, since the training set changes each time.

In k-fold cross validation you have to compute the average empirical loss over the k training folds in order to obtain the training error, and to compute the average empirical loss over the k validation folds to obtain validation error.

To choose the number of folds, you need to have a sufficient large train fold, in order to avoid overfitting, as well as the validation folds, in order to get reliable estimate of generalization. On the contrary, choosing a too large set will results in more computations to be done.

k-fold cross validation requires a method to split into folds. The most basic method is to evenly divide the data set into k parts, and then taking k-1 sections for training and the last for validation, and then repeat.

For example



This method works if data is i.i.d. (independent and identically distributed). If data points are grouped or ordered, this method fails.

Imbalanced classes and group structure

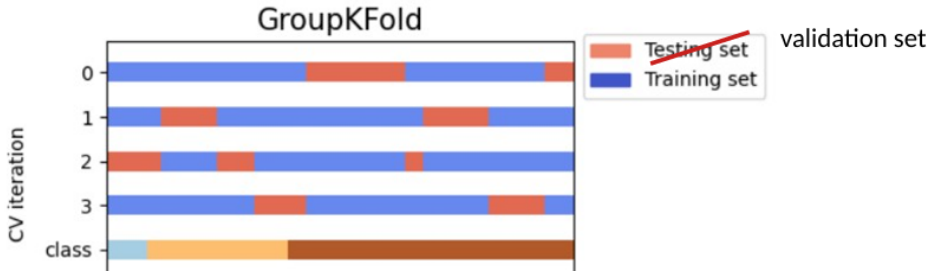
If points are not i.i.d you can group them together in a smarter way. For example, data can have different labels or can be obtained at consecutive time instants. The idea is that we want our train to represent all the data an to break the dependencies.

If we want to preserve the proportion of data classes, we can perform the so called stratified splitting.

sklearn.model_selection.StratifiedKFold

class sklearn.model_selection.**StratifiedKFold**(*n_splits=5, *, shuffle=False, random_state=None*)

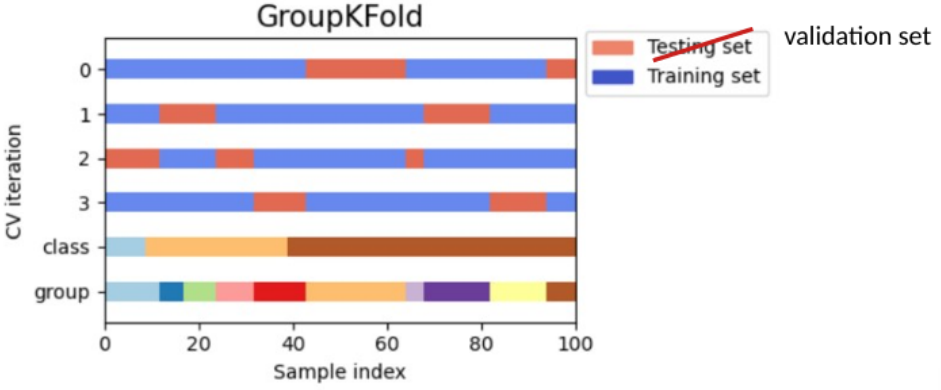
It means that we take the first class (the blue one) and evenly distribute it among the k (4) folds. Then we take the orange and the brown classes and can do the same.



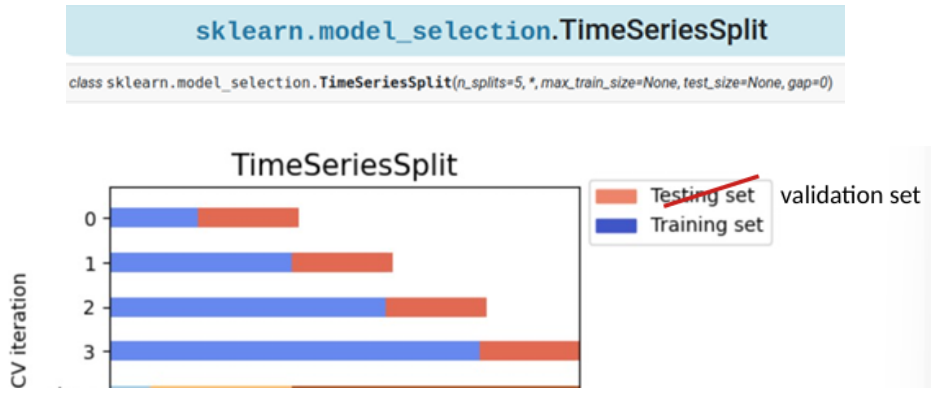
If you want to preserve the proportion of data groups, we can perform the so called group k-fold. In this case we don't want each group to be split evenly among the folds. We have to make sure that one group stays all in one fold.

sklearn.model_selection.GroupKFold

class sklearn.model_selection.**GroupKFold**(*n_splits=5*)



For time series split we have to consider another thing, because also time series, if things are collected over time, data are not independent. In this case, it's important to be sure that what's used for the training is the past and not the future, as well as we want the validation to be the future and not the past. If you are repeating multiple times, you are not doing the k-fold cross validation as previously. So what you can do is to make a second fold in which you can expand your time frame considering the previous train and validation and adding a new validation. In this ways we do 4 splits all with different training set and validation set.



Test set

The test set is an additional set to the training and validation ones. It is needed because once we choose the model with the lowest validation error, the latter is again a poor performance indicator and in some cases it could be too optimistic. So we need the test set, different from the training and validation sets, so that the training set is used for ERM, the validation set is used to choose the model and the test set is used to estimate the expected loss.

Model selection recipe

So to choose a model the steps are:

1. get a list of candidate models
2. for each candidate model
    a. you perform ERM on training to obtain the training error
    b. you compute the validation error on the validation set
3. you compare the models and choose hypothesis with the minimum validation error
4. compute test error on the test set