Prepare data: transformations

Transforming data (Recipe C.3) may be **the most important** step of the multi-step Recipe
!

It is often the case that the "raw" features given to us don't suffice

- we may need to create "synthetic" features.
- This is called **feature engineering**.

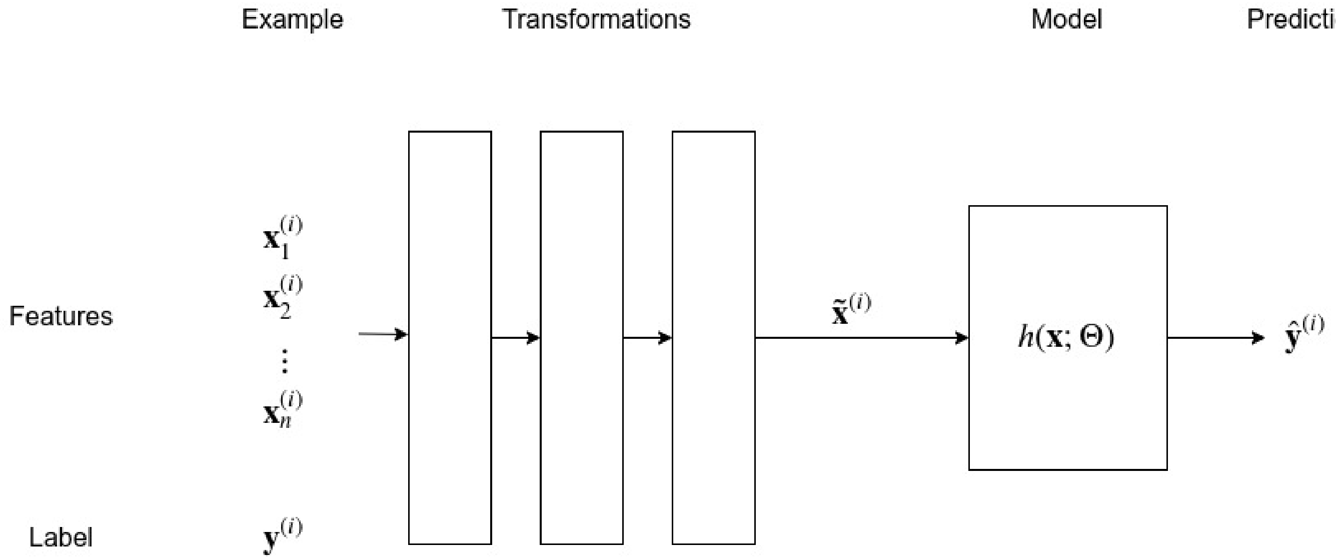
In the "curvy" data case, adding the squared feature was key to a better prediction.

Feature engineering, or transformations

- takes an example: vector $\mathbf{x^{(i)}}$ with n features
- produces a new vector $\tilde{\mathbf{x}}^{\mathbf{(i)}}$, with n' features

We ultimately fit the model with the transformed *training* examples.

Feature Engineering



Example

Transformations

Model

Prediction

Features

$\mathbf{x}_1^{(i)}$

$\mathbf{x}_2^{(i)}$

$\vdots$

$\mathbf{x}_n^{(i)}$

Label

$\mathbf{y}^{(i)}$

$\tilde{\mathbf{x}}^{(i)}$

$h(\mathbf{x};\Theta)$

$\hat{\mathbf{y}}^{(i)}$

- MIssing data inputation
- Standardization
- Discretization
- Categorical variable encoding

The above diagram shows multiple tranformations

- organized as a sequence (sometimes called a *pipeline*) of independent transformations $T_1, T_2, \ldots, T_t$
$$\begin{aligned} \tilde{\mathbf{x}}_{(1)} &= T_1(\mathbf{x}) \\ \tilde{\mathbf{x}}_{(2)} &= T_1(\tilde{\mathbf{x}}_{(1)}) \\ &\vdots \\ \tilde{\mathbf{x}}_{(l+1)} &= T_1(\tilde{\mathbf{x}}_{(l)}) \end{aligned}$$

We write the final transformed $\tilde{\mathbf{x}}$ as a function T that is the composition of each transformation function

$$\tilde{x} = T(\mathbf{x}) = T_t(\ T_{t-1}(\ldots T_1(\mathbf{x}) \ldots)\ )$$

The length of the final transformed vector $\tilde{\mathbf{x}}$ may differ from the n, the length of the input $\mathbf{x}$

- may add features
- may drop features

The model $\mathbf{h}(\mathbf{x}; \Theta)$ now takes $\tilde{x}$ as input

- The model is *fit* to *transformed* training examples
- Prediction must be performed on *transformed* test examples !
  - Examples must be treated consistently (e.g., have the same shape)
  - Regardless of whether the example is from training, validation or test

It is thus very important to apply the transformation pipeline to every example.

Examples

The first transformation we encountered added a feature ($\mathbf{x}^2$ term) that improved prediction.

Some transformations alter existing features rather than adding new ones.

Transformations in detail will be the subject of a separate lecture but let's cover the basics.

Let's consider a second reason for transformation: filling in (imputing) missing data for a feature.

$\boxed{\#}$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$
1	1.0	10
2	2.0	20
$\vdots$	$\vdots$	$\vdots$
i	2.0	NaN
$\vdots$	$\vdots$	$\vdots$
m	. . .	

In the above: feature $\mathbf{x}_2$ is missing a value in example i: $x^\ip_2 = \text{NaN} $

We will spend more time later discussing the various ways to deal with missing data imputation.

For now: let's adopt the common strategy of replacing it with the median of the defined values:

$$\text{median}(\mathbf{x}_2) = \text{median}(\{\mathbf{x}_2^{\mathbf{(i)}} | 1 \le i \le m, \mathbf{x}_2^{\mathbf{(i)}} \ne \text{NaN}\})$$

This imputation is a kind of data transformation: replacing an undefined value.

”Fitting” transformations

Transformations often have their own parameters $\Theta_{\text{transform}}$ that is separate from the Θ parameters of the model.

In the case of imputation: the mean/median of a feature j for a missing value.

In that case: $\Theta_{\text{transform}}$ must contain $\text{median}(\mathbf{x}_j)$

The process of Transformations is similar to fitting a model and predicting.

The parameters in $\Theta_{\text{transform}}$

- are "fit" by examining all training data $\mathbf{X}$
- once fit, we can transform ("predict") *any* example (whether it be training/validation or test)

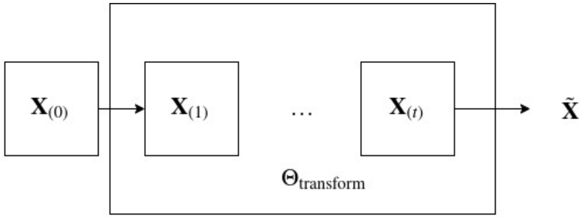
Applying transformations consistently

Note that the transformation of an example depends on $\Theta_{\text{transform}}$, which is fit *only on the training data*.

When transforming any example (including one *not* in $\mathbf{X}$) one uses $\Theta_{\text{transform}}$ from the transformation fitting

- You **do not** recalculate $\Theta_{\text{transform}}$ on test examples !
    - Just imagine that you are given each test example in isolation-- there is no summary statistic to compute!

Feature engineering: fit, then transform



Train

$\text{fit} : \mathbf{X}_{(0)} \mapsto \Theta_{\text{transform}}$

$\text{transform}(\mathbf{x}^{(\mathbf{i})}; \Theta_{\text{transform}}) \mapsto \tilde{\mathbf{x}}^{(i)}$

Test

$\text{transform}(\underline{\mathbf{x}}; \Theta_{\text{transform}}) \mapsto \underline{\mathbf{x}}'$

NO fitting at test time, re-use

$\Theta_{\text{transform}}$

$\Theta_{\text{transform}}$

- standardization: mean($\mathbf{X}$), std-dev($\mathbf{X}$)
- scaling: min($\mathbf{X}$), max($\mathbf{X}$)
- imputation: median($\mathbf{X}$)

To re-iterate:

- **No** fitting is applied to test examples only train !
- The $\Theta_\text{transform}$ obtained from training data is used in transforming *test* as well as *train* examples

There are several reasons not to re-fit on test examples

- it would be a kind of "cheating" to see all test examples (required to fit)
- you should assume that you only encounter one test example at a time, not as a group

Transformations are applied to both training and test examples

- training examples so that the model may be fit
- test examples in order to be able to predict
    - to the extent that transformations added features (e.g., $\mathbf{x}^2$) or changed features (imputation)
    - the test examples *must be transformed* the same way as training
        - otherwise they won't be similar to training examples, violating the fundamental assumption of ML

Using pipelines to avoid cheating in cross validation

We know that looking at test examples while fitting a model is "cheating".

That's one of the reasons that $\Theta_{\text{transform}}$ is fit only using training examples.

There is a subtle (but common and often overlooked) form of cheating that occurs when using cross-validation.

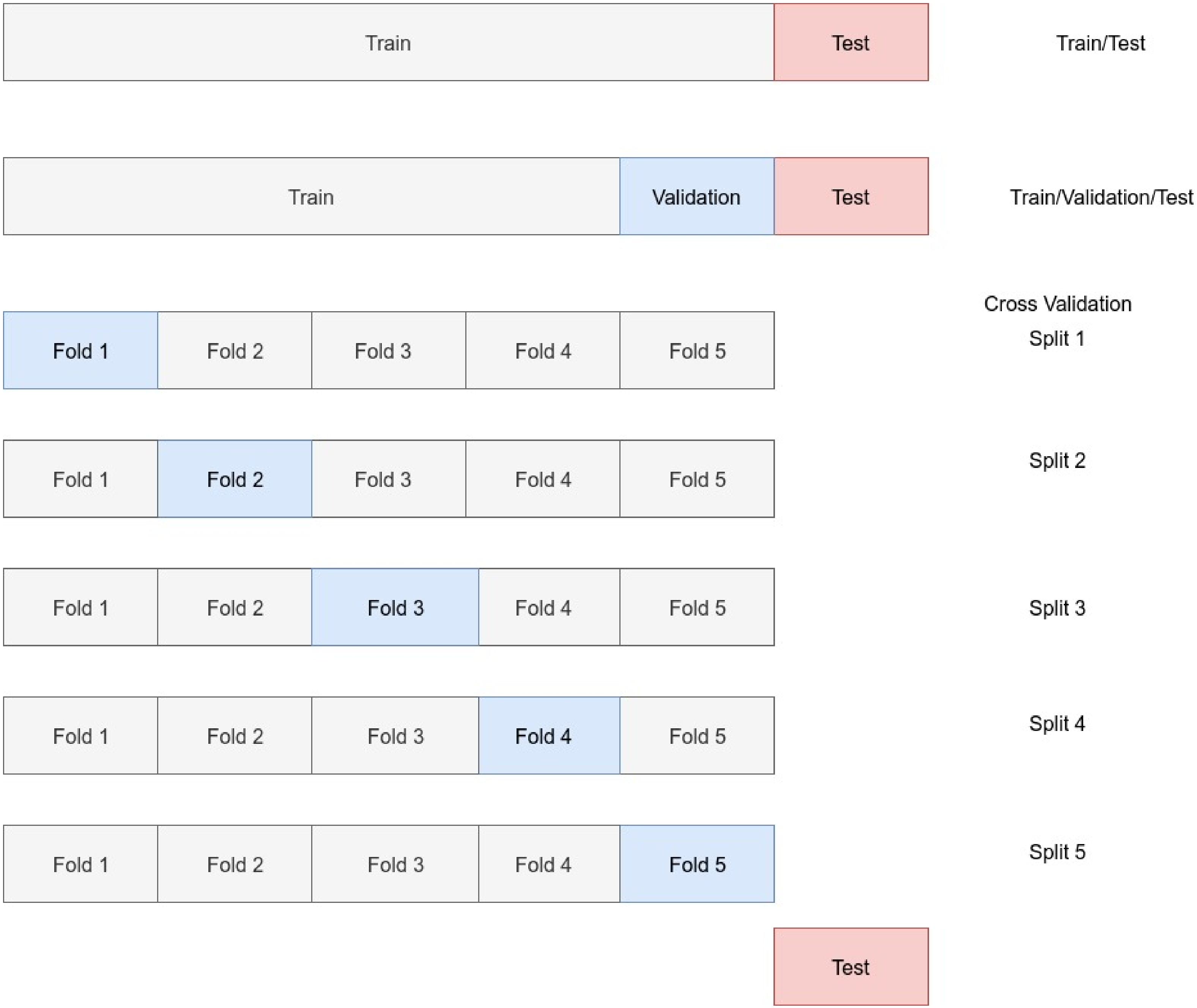
k-fold cross-validation:

- Allows us to use part of the training examples as "out of sample" for computing the Performance Metric
    - We can reuse these out of sample examples multiple times -- we couldn't do this with the test examples

The process of cross-validation

- Divides the training examples into k "folds"
- A model is fit k times
    - by selecting one of the folds to serve the role of "out of sample" (on which the Performance Metric is evaluated)
    - using the other $(k-1)$ folds as the training data
- This gives us k Performance Metrics, from which we can see the distribution

Cross Validation/Test split



Train | Test — Train/Test

Train | Validation | Test — Train/Validation/Test

Cross Validation

Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 — Split 1

Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 — Split 2

Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 — Split 3

Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 — Split 4

Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 — Split 5

Test

Consider the difference between

- Transforming *all* the training examples *before* applying cross-validation
- For each iteration of Cross-Validation: transforming only the examples from the $(k-1)$ folds used

In the first case, $\Theta_{\text{transform}}$ depends on *all* k folds

- Even though during each iteration of Cross Validation, one fold should be treated as out of sample
- In the first case, each iteration of Cross Validation could be influenced by out of sample examples

Performing the transformation within each iteration of Cross Validation avoids peeking at out of sample examples.

Perhaps the reason that this subtle cheating is overlooked is because it might make the use of cross validation burdensome.

`sklearn` has been engineered to make it easy to perform transformations in the proper manner.

We will see this in action within the notebook for Classification.

In [2]:

```
print("Done")
```

```
Done
```