The mechanics of transformations

We briefly introduced transformations in [the overview of the Prepare the data step of the Recipe for ML (Prepare_data_Overview.ipynb)](Prepare_data_Overview.ipynb).

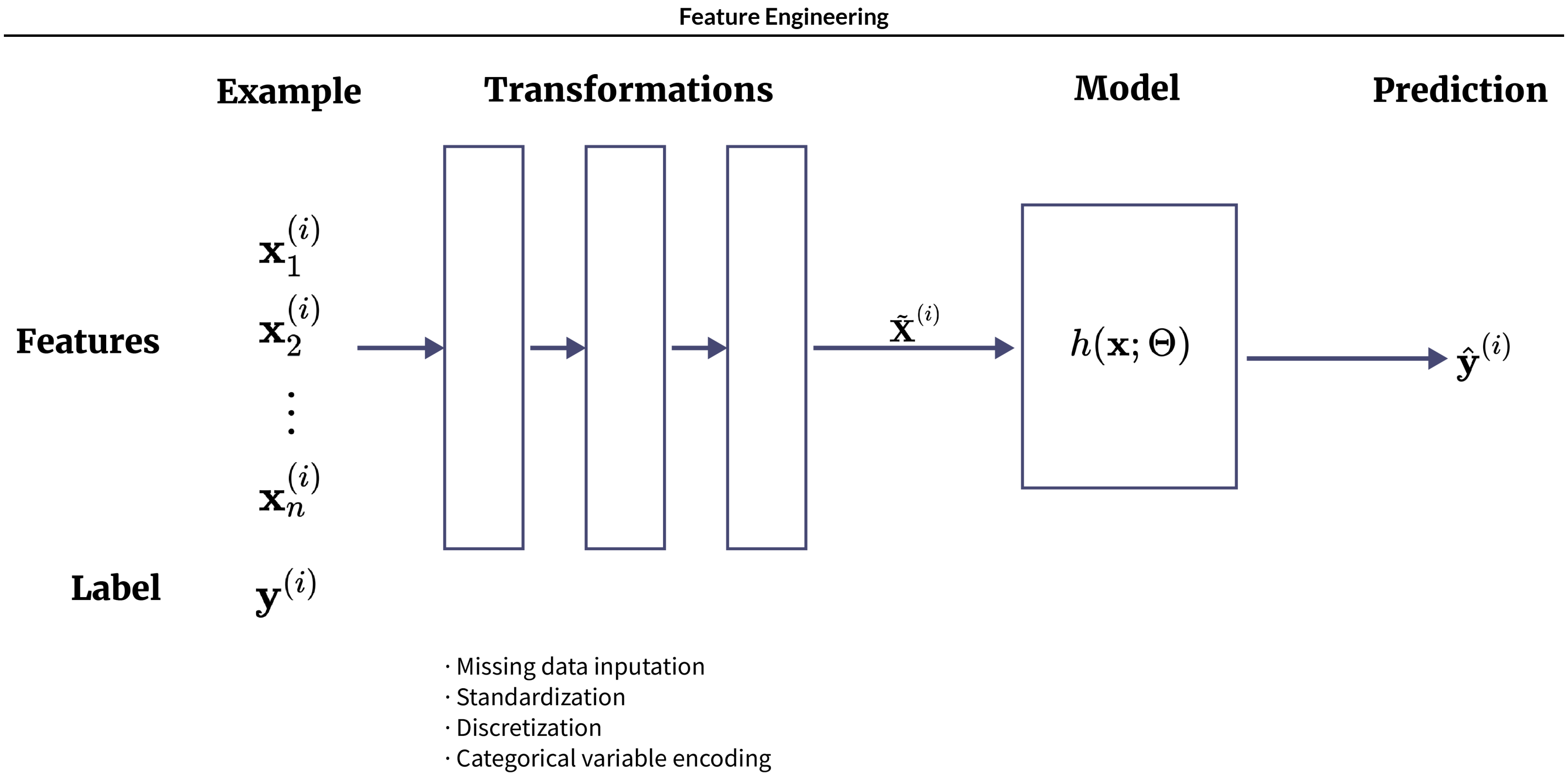
We recap the key points:

Fitting transformations

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
Label
$\mathbf{x}_1^{(i)}$
$\mathbf{x}_2^{(i)}$
$\mathbf{x}_n^{(i)}$
$\mathbf{y}^{(i)}$
$\tilde{\mathbf{x}}^{(i)}$
$h(\mathbf{x}; \Theta)$
$\hat{\mathbf{y}}^{(i)}$
· Missing data inputation
· Standardization
· Discretization
· Categorical variable encoding

Transformations have parameters $\Theta_{\text{transform}}$ distinct from the model's parameters Θ.

- Example: Missing data imputation for a feature substitutes the mean/median feature value
- $\Theta_{\text{transform}}$ stores this value

Our prediction is thus

$$\begin{aligned}
\hat{\mathbf{y}} &= h_\Theta(\tilde{\mathbf{x}}) \\
&= h_\Theta(\, T_{\Theta_{\text{transform}}}(\mathbf{x})\,)
\end{aligned}$$

Transformations can be applied to the target as well. For example

- One Hot Encoding a categorical target for a Classification task
- Scaling the target (e.g., pixel intensities from a range $[0 \ldots 255]$ to a range $[-1 \ldots +1]$)

If we transform the target $\mathbf{y}$ into new units, the predicted $\hat{\mathbf{y}}$ will also be in the new units

- If we want to report our prediction in original units
- We must be able to invert the transformation

For example:

- Logistic Regression transforms the target into Log Odds
- We want to report our prediction in terms of one class of the Categorical variable

Apply transformations consistently

Suppose you transform your raw training set

$$\langle \mathbf{X}, \mathbf{y} \rangle$$

to

$$\langle \mathbf{X}', \mathbf{y}' \rangle$$

In order to satisfy the Fundamental Assumption of Machine Learning

- you must apply the **identical** transformation to
- validation examples
- test examples

By wrapping up all your transformations in an `sklean Pipeline`

- you can ensure that your transformations are applied consistently to each example
- regardless of its source

But remember

- the transformation parameters $\Theta_{\text{transform}}$
- are fit to the **training examples** only
- never re-fit to test examples

One simple way to remember this

- assume you can look at your test examples only **one at a time** rather than as a collection
- it doesn't make sense to "fit" a transformation on a singleton

Transformed targets: remember to invert your prediction !

Suppose you transform your raw training set

$$\langle \mathbf{X}, \mathbf{y} \rangle$$

to

$$\langle \mathbf{X}', \mathbf{y}' \rangle$$

where f is the transformation applied to targets

$$\mathbf{y}' = f(\mathbf{y})$$

The units of $\mathbf{y}$ change from u (e.g., dollars) to u' (e.g., dimensionless z-score)

Then your model's predictions

$$\hat{\mathbf{y}}'$$

are in units of u' **not** u.

You must **invert** the transformed predicted target $\hat{\mathbf{y}}'$ back to units of u

$$\hat{\mathbf{y}} = f^{-1}(\hat{\mathbf{y}}')$$

For example

$$\mathbf{y} \mapsto \frac{\mathbf{y} - \mu}{\sigma}$$

where μ, σ are the mean and standard deviation of the *training* examples $\mathbf{y}$

Then

$$\hat{\mathbf{y}}' \mapsto \sigma * \hat{\mathbf{y}}' + \mu$$

`sklearn` transformers provide an `inverse_transform` method to facilitate this.

Transformers in `sklearn`

A transformer in `sklearn` provides the following methods

- `fit`: using training examples, compute $\Theta_{\text{transform}}$
- `transform`: map an example $[\mathbf{x^{(i)}}, \mathbf{y^{(i)}}]$ into transformed example $[\tilde{\mathbf{x}}^{\mathbf{(i)}}, \tilde{\mathbf{y}}^{\mathbf{(i)}}]$
- `inverse_transform`: map a transformed example $[\tilde{\mathbf{x}}^{\mathbf{(i)}}, \tilde{\mathbf{y}}^{\mathbf{(i)}}]$ back to its source example $[\mathbf{x^{(i)}}, \mathbf{y^{(i)}}]$

In [3]:

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

Done