Intermediate Machine Learning: 7th lesson – Data Leakage

Data leakage:

Data leakage happens when the training data contains information about the target, but similar data will not be available when the model is used for prediction. This leads to high performance on the training set (and possibly even the validation data), but the model will perform poorly in production. In other words, leakage causes a model to look accurate until the user start making decisions with the model, and then the model becomes very inaccurate.

There are two main types of leakage:

* Target leakage
* Target leakage occurs when predictors include data that will not be available at the time the user makes predictions. It is important to think about target leakage in terms of the timing or chronological order that data becomes available, not merely whether a feature helps make good predictions.
* An example will be helpful. Imagine the user want to predict who will get sick with pneumonia. The top few rows of that raw data look like this:

| got\_pneumonia | age | weight | male | took\_antibiotic\_medicine | ... |
| --- | --- | --- | --- | --- | --- |
| False | 65 | 100 | False | False | ... |
| False | 72 | 130 | True | False | ... |
| True | 58 | 100 | False | True | ... |

People take antibiotic medicines after getting pneumonia in order to recover. The raw data shows a strong relationship between those columns, but took\_antibiotic\_medicine is frequently changed after the value for got\_pneumonia is determined. This is target leakage.

* The model would see that anyone who has a value of False for took\_antibiotic\_medicine didn't have pneumonia. Since validation data comes from the same source as training data, the pattern will repeat itself in validation, and the model will have great validation (or cross-validation) scores. But the model will be very inaccurate when subsequently deployed in the real world, because even patients who will get pneumonia won't have received antibiotics yet when we need to make predictions about their future health.
* To prevent this type of data leakage, any variable updated (or created) after the target value is realized should be excluded.
* Train-test contamination
* A different type of leak occurs when users aren't careful to distinguish training data from validation data. Recall that validation is meant to be a measure of how the model does on data that it hasn't considered before. The user can corrupt this process in subtle ways if the validation data affects the preprocessing behavior. This is sometimes called train-test contamination.
* For example, imagine users run preprocessing (like fitting an imputer for missing values) before calling train\_test\_split(). The end result? The model may get good validation scores, giving users great confidence in it, but perform poorly when users deploy it to make decisions. After all, users incorporated data from the validation or test data into how predictions made, so the may do well on that particular data even if it can't generalize to new data. This problem becomes even more subtle (and more dangerous) when users do more complex feature engineering.
* If the validation is based on a simple train-test split, exclude the validation data from any type of fitting, including the fitting of preprocessing steps. This is easier if users use scikit-learn pipelines. When using cross-validation, it's even more critical that users do the preprocessing inside the pipeline!