*When discussing the evaluation of model performance, we introduced the method of dividing   
data into training vs. test set. In the illustration of the Caret package, we further demonstrate   
that the summary statistics of training and test are similar. The question for the discussion is, in   
machine learning, why should we divide the data into two sets of similar characteristics? And   
how do we do it?*

The introductory lecture for Module 3 lays out the primary objective of supervised machine learning models: to find a function *h* that maps the predictor variable *X* to the target variable *Y*. A brief skim of the lecture might leave the impression that the entire concept of machine learning boils down to just this simple formula – give the computer *X*, give the computer *Y*, and *voilà*, you’ve got a finished supervised model. But while the formula itself seems simple, what about the samplesof those *X* and *Y* values that are supplied to the program to help it find *h*? Do we just feed the computer a full dataset in CSV format and let the machine do the rest? As the subsequent materials of Module 3 establish, this is not the case. The reality is that quite a bit of work goes into creating the subsets of data that the supervised program uses to learn – and that work involves partitioning the population into *training* and *test* datasets.

The benefits of splitting the data in this way and the means by which that split is accomplished, are, I believe, best described through a real-world example – and for this, I’ll use an actual project that I’ve worked on in my role as Systems Analyst for Cleveland Clinic Philanthropy. Part of my job is to develop scoring systems that aim to predict how likely a given patient is to make a donation. To build these predictive models, I loosely follow the “steps to be taken in a data mining effort” outlined in Chapter 2 of Shmueli’s *Data mining for Business Analytics*. After first obtaining a dataset of over several million patients, I’ll explore the data to identify a host of predictor *X* variables - including estimated wealth holdings, number of patient experiences, and confirmed donations to other organizations – as well as our desired *Y* outputs, “donor” versus “non-donor”. I’ll then clean that data, replacing any missing values and breaking down any nominal categorical variables into binary “dummy”variables (for example, inpatient versus non-inpatient, outpatient versus non-outpatient, etc.).

Next comes the “split” – breaking the dataset into the *train* and *test* subsets (representing, ideally, about 70%-80% and 20%-30% of the overall data, respectively). The methodology behind how this is done depends on how the overall dataset reflects the actual population that is being tested. While *random sampling* ensures that each member of the population is equally likely to be pulled into the sample, for my models I need to use *stratified sampling*. Because only a small fraction of all patients ever become donors, there is a nonzero chance that a random sample from our patient pool will produce a sample that contains no donors – making it impossible to produce a final model that accurately measures the likelihood of making a gift. In practice, creating and splitting this sample can be accomplished a variety of ways – using the “createDataPartition()” function within the CARET R package for example, or using the “train\_test\_split()” function in the sklearn Python library (Shmueli 38). My method has typically been to extract patient records from the dataset using a randomly-ordered SQL query, and then to extract a randomly-ordered set of donor records as well, before finally combining the two subsets together to form a training set – and then repeating the same steps over again in order to create the additional test set (making sure to exclude any records that are already in the training set). Most important at this stage though is to ensure that the overall proportion of donors to patients in both samples reflects the proportion of the entire dataset, thereby preventing any bias. Once this is done, I am left with a full training sample to help the model learn, and a test model to evaluate its’ final performance.

While this explains the *how*, what is left is *why*. For one, splitting the overall dataset into train and test sets allows us to perform an out-of-sample evaluation – i.e. testing the final model over a subset of unseen data and giving us an additional check on the model’s accuracy. Dividing the dataset in this way also protects against *overfitting* and *underfitting*. If the subset of donors in the training set has an unusually high number of dermatology patients, for example, that “noise” might inadvertently be included as part of the actual model, leading to a prediction that dermatology patients are more likely to become donors than other types of patients; having a separate, unseen sample to test the model on helps to root out instances like this. These benefits - providing additional unseen values for testing, and protecting against bad fits - help to make our models as accurately predictive as they can possibly be, and support our overall objective of finding the *h* that explains the relationship between *X* and *Y*.

**References**

Shmueli, G., Bruce, P. C., Gedeck, P., & Patel, N. R. (2020). *Data mining for Business Analytics: Concepts, techniques and applications in Python*. John Wiley & Sons, Inc.

**Response**

Hello Brenden.

It was nice to see a mention of K-fold cross validation in your post. I’ve admittedly been racking my brain trying to think of what a real-world application of this method would be, as it sure seems like an interesting method of sampling. While my brain has unfortunately come up empty, I was able to locate an example on the internet that is based on around a hypothetical speech recognition dataset. Say you have 5 speakers, and 250 recordings from each speaker – “K-fold cross validation will allow you to train 5 different models, where in each model you are using one of the speakers for the testing dataset and the remaining for the training dataset” (KDnuggets). So the K-fold method seems to work particularly well in instances where we have data from 5 or 10 different subjects (5 and 10, of course, typically being the ideal number of folds used) and we want cycle through all of those data subsets during our training.

**References**

*Why use K-fold cross validation?* KDnuggets. (n.d.). Retrieved September 18, 2022, from https://www.kdnuggets.com/2022/07/kfold-cross-validation.html