Machine Learning Explainability: 2nd lesson – Permutation Importance

One of the most basic questions we might ask of a model is, what features have the biggest impact on predictions?

This concept is called **feature importance**. There are multiple ways to measure feature importance. Some approaches answer subtly different versions of the question above. Other approaches have documented shortcomings.

In this lesson, we'll focus on permutation importance. Compared to most other approaches, permutation importance is fast to calculate, widely-used and understood, and consistent with properties we would want a feature importance measure to have.

***How it works?***

Permutation importance uses models differently than anything you've seen so far, and many people find it confusing at first. So we'll start with an example to make it more concrete. Consider data with the following format:

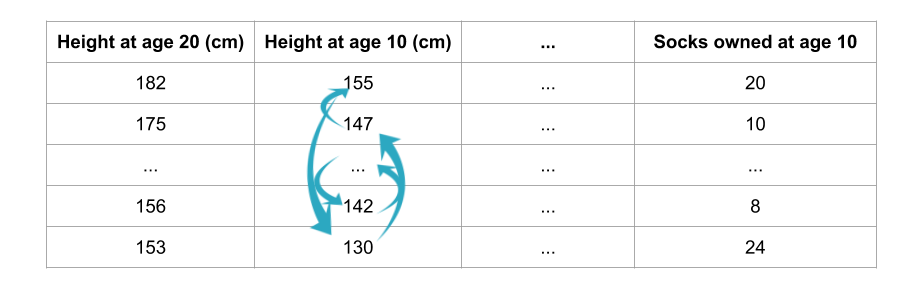


We want to predict a person’s height when they become 20-years old, using data that is available at age 10. Our data includes useful features (*height at age 10*), features with little predictive power (*socks owned*), as well as some other features we won't focus on in this explanation.

**Permutation importance** is calculated after a model has been fitted. So we won't change the model or change what predictions we'd get for a given value of height, sock-count, etc.

Instead, we will ask the following question:

If an user randomly shuffle a single column of the validation data, leaving the target and all other columns in place, how would that affect the accuracy of predictions in that currently-shuffled data?



Randomly re-ordering a single column should cause less accurate predictions, since the resulting data no longer corresponds to anything observed in the real world. Model accuracy especially suffers if we shuffle a column that the model relied on heavily for predictions. In this case, shuffling *height at age 10* would cause terrible predictions. If we shuffled *socks owned* instead, the resulting predictions wouldn't suffer nearly as much.

With this insight, the process is as follows:

* Acquire a trained model.
* Shuffle the values in a single column, make predictions using the resulting dataset. Use these predictions and the true target values to calculate how much the loss function suffered from shuffling. That performance deterioration measures the importance of the variable that has been shuffled.
* Return the data to the original order, undoing the shuffle from the 2nd step. Now repeat the 2nd step with the next column in the dataset, until calculating the importance of each column.

Code example:

Our example will use a model that predicts whether a soccer/football team will have the "*Man of the Game*" winner based on the team's statistics. The "*Man of the Game*" award is given to the best player in the game. Model-building isn't our current focus, so the cell below loads the data and builds a rudimentary model.

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

data = pd.read\_csv('../input/fifa-2018-match-statistics/FIFA 2018 Statistics.csv')

y = (data['Man of the Match'] == "Yes") *# Convert from string "Yes"/"No" to binary*

feature\_names = [i for i **in** data.columns if data[i].dtype **in** [np.int64]]

X = data[feature\_names]

train\_X, val\_X, train\_y, val\_y = train\_test\_split(X, y, random\_state=1)

my\_model = RandomForestClassifier(n\_estimators=100,

random\_state=0).fit(train\_X, train\_y)

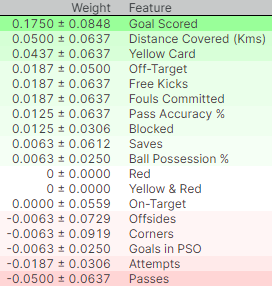
Here is how to calculate and show importances with the eli5 library:

import eli5

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(my\_model, random\_state=1).fit(val\_X, val\_y)

eli5.show\_weights(perm, feature\_names = val\_X.columns.tolist())



Interpreting permutation importances:

The values towards the top are the most important features and those towards the bottom matter least. The first number in each row shows how much model performance decreased with a random shuffling (in this case, using "accuracy" as the performance metric). Like most things in data science, there is some randomness to the exact performance change from a shuffling a column. We measure the amount of randomness in our permutation importance calculation by repeating the process with multiple shuffles. The number after the ± measures how performance varied from one-reshuffling to the next.

You'll occasionally see negative values for permutation importances. In those cases, the predictions on the shuffled (or noisy) data happened to be more accurate than the real data. This happens when the feature didn't matter (should have had an importance close to 0), but random chance caused the predictions on shuffled data to be more accurate. This is more common with small datasets, like the one in this example, because there is more room for luck or chance. In our example, the most important feature was *Goals scored*. That seems sensible. Soccer fans may have some intuition about whether the orderings of other variables are surprising or not.