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*. I've seen feature importance used effectively many times for every purpose in the list of use cases above.

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

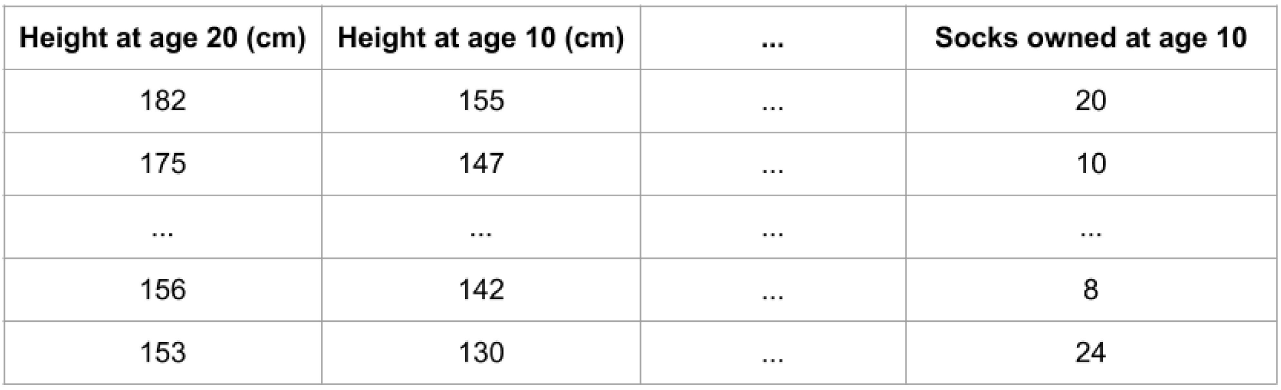
In this lesson, we'll focus on *permutation importance*. Compared to most other approaches, permutation importance is:

- Fast to calculate
- Widely used and understood
- 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:



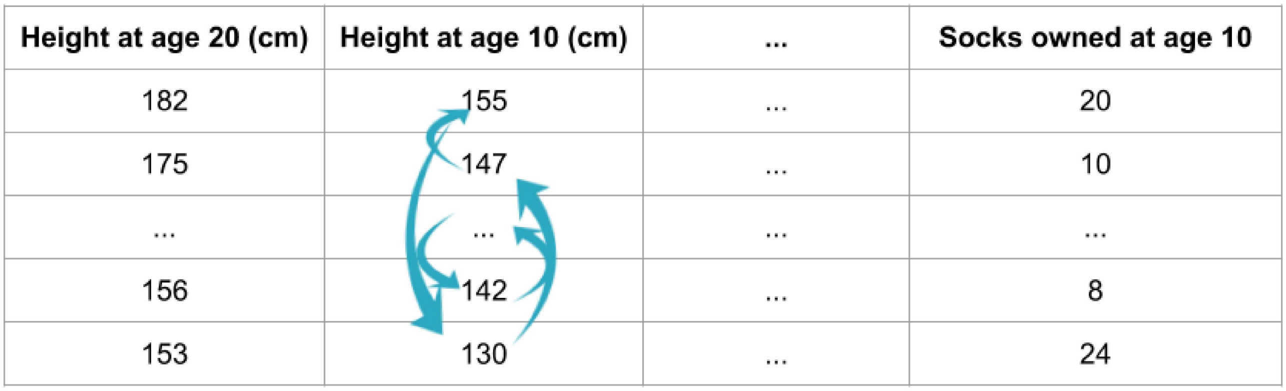
Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...	...	...	...
156	142	...	8
153	130	...	24

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 I 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 now-shuffled data?



Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...	...	...	...
156	142	...	8
153	130	...	24

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:

1. Get a trained model
2. Shuffle the values in a single column, make predictions using the resulting dataset. Use these predictions and the true target values to

In [1]:

```python
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(random_state=0).fit(train_X, train_y)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/forest.py:248: FutureWarning: The default value
of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

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

In [2]:

```python
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())
```

Out[2]:

```
         Weight    Feature
0.0750 ± 0.1159    Goal Scored
0.0625 ± 0.0791    Corners
0.0437 ± 0.0500    Distance Covered (Kms)
0.0375 ± 0.0729    On-Target
0.0375 ± 0.0468    Free Kicks
0.0187 ± 0.0306    Blocked
0.0125 ± 0.0750    Pass Accuracy %
0.0125 ± 0.0500    Yellow Card
0.0063 ± 0.0468    Saves
0.0063 ± 0.0250    Offsides
```

Weight	Feature
0.0063 ± 0.1741	Off-Target
0.0000 ± 0.1046	Passes
0 ± 0.0000	Red
0 ± 0.0000	Yellow & Red
0 ± 0.0000	Goals in PSO
-0.0312 ± 0.0884	Fouls Committed
-0.0375 ± 0.0919	Attempts
-0.0500 ± 0.0500	Ball Possession %

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/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.