SHAP Values

You've seen (and used) techniques to extract general insights from a machine learning model. But what if you want to break down how the model works for an individual prediction?

SHAP Values (an acronym from SHapley Additive exPlanations) break down a prediction to show the impact of each feature. Where could you use this?

- A model says a bank shouldn't loan someone money, and the bank is legally required to explain the basis for each loan rejection
- A healthcare provider wants to identify what factors are driving each patient's risk of some disease so they can directly address those risk factors with targeted health interventions

You'll use SHAP Values to explain individual predictions in this lesson. In the next lesson, you'll see how these can be aggregated into powerful model-level insights.

How They Work

SHAP values interpret the impact of having a certain value for a given feature in comparison to the prediction we'd make if that feature took some baseline value.

An example is helpful, and we'll continue the soccer/football example from the permutation importance and partial dependence plots lessons.

In these tutorials, we predicted whether a team would have a player win the *Man of the Game* award.

We could ask:

- How much was a prediction driven by the fact that the team scored 3 goals?

But it's easier to give a concrete, numeric answer if we restate this as:

- How much was a prediction driven by the fact that the team scored 3 goals, **instead of some baseline number of goals.**

Of course, each team has many features. So if we answer this question for `number of goals`, we could repeat the process for all other features.

SHAP values do this in a way that guarantees a nice property. When we make a prediction

```
sum(SHAP values for all features) = pred_for_team - pred_for_baseline_values
```

That is, the SHAP values of all features sum up to explain why my prediction was different from the baseline. This allows us to decompose a prediction in a graph like this:

If you want a larger view of this graph, here is a link

How do you interpret this?

We predicted 0.7, whereas the base_value is 0.4979. Feature values causing increased predictions are in pink, and their visual size shows the magnitude of the feature's effect. Feature values decreasing the prediction are in blue. The biggest impact comes from `Goal Scored` being 2. Though the ball possession value has a meaningful effect decreasing the prediction.

If you subtract the length of the blue bars from the length of the pink bars, it equals the distance from the base value to the output.

There is some complexity to the technique, to ensure that the baseline plus the sum of individual effects adds up to the prediction (which isn't as straightforward as it sounds.) We won't go into that detail here, since it isn't critical for using the technique. This blog

post has a longer theoretical explanation.

Code to Calculate SHAP Values

We calculate SHAP values using the wonderful Shap library.

For this example, we'll reuse the model you've already seen with the Soccer data.

In [1]:
```
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, 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 the code to get the SHAP values for a single prediction.

In [2]:
```
import shap  # package used to calculate Shap values

row_to_show = 5
data_for_prediction = val_X.iloc[row_to_show]  # use 1 row of data here. Could use multiple
 rows if desired
data_for_prediction_array = data_for_prediction.values.reshape(1, -1)

# Create object that can calculate shap values
explainer = shap.TreeExplainer(my_model)

# Calculate Shap values
shap_values = explainer.shap_values(data_for_prediction)
```

The shap_values object above is a list with two arrays. The first array in the list is the SHAP values for a negative outcome (don't win the award.) . The second array is the list of SHAP values for the positive outcome, which is how we usually think about predictions. It's cumbersome to review a raw array, but the shap package has a nice way to visualize the results. Before doing that, here are the predicted probabilities of each outcome.

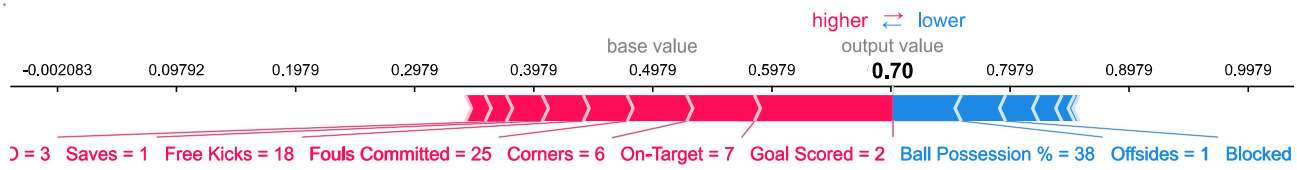
In [3]:
```
my_model.predict_proba(data_for_prediction_array)
```

Out[3]:
```
array([[0.3, 0.7]])
```

The team is 70% likely to have a player win the award. Now we visualize the SHAP values

In [4]:
```
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], data_for_prediction)
```

Out[4]:



If you look carefully at the code where we created the SHAP values, you'll notice we reference Trees in `shap.TreeExplainer(my_model)`. But the SHAP package has explainers for every type of model.

- `shap.DeepExplainer` works with Deep Learning models.
- `shap.KernelExplainer` works with all models, though it is slower than other Explainers and it offers an approximation rather than exact Shap values.

Here is an example using KernelExplainer to get the same results you saw above.

In [5]:
```
# use Kernel SHAP to explain test set predictions
k_explainer = shap.KernelExplainer(my_model.predict_proba, train_X)
k_shap_values = k_explainer.shap_values(data_for_prediction)
shap.force_plot(explainer.expected_value[1], shap_values[1], data_for_prediction)
```

Out[5]: