Expected Goals (xG) Model

F Fisher Marks
10 min read · 23 hours ago

Fisher Marks & Nicholas Strickler

Introduction

The 2023 introduction of Lionel Messi to the roster of second-to-last Inter Miami CF saw the Major League Soccer club's valuation climb over 70% to over $1 billion dollars within a year. The 2022 World Cup earned FIFA a reported $7.5 billion. Last year Spanish club Real Madrid's yearly revenue passed the $800 million mark.

As the United States prepares to hold the 2026 World Cup, the soccer industry is growing rapidly both domestically and worldwide. And while the front office focuses on raking in millions off of sponsorships and advertising, coaches and players have to play their ever-important role in winning games and bringing home trophies. And for a game that is won solely based on the number of goals scored at the end of stoppage time, it is crucial to know which shots are worth taking and which aren't.

As such, the first point of discussion in every statistical report on a game is the number of Expected Goals (xG). Put simply, xG aims to estimate the number of goals a team should have had at the end of the game based on the shots that they took. For each shot, xG is calculated based on a wide variety of factors measured at the instant before the shot is taken. By understanding what makes a shot have high xG, teams can be more selective about what kinds of shot they take in order to maximize their number of goals.

Using StatsBomb's open data repository, we have employed a variety of machine learning methods to produce xG models that we compare against the results of StatsBomb model for each shot, alongside an analysis of what makes a good xG model.

Data

The StatsBomb open data repository includes an events folder, containing a json file for each match included in the dataset. In each json file, there is an object for every type of event that occurs within a match including what we care about: shots.

At this stage in the project we first had to decide what we wanted each of our feature vectors to look like. StatsBomb provides a wealth of information about each shot including information about the shot-taker, the shot position, information about the goalie, the goalie position, body part the shot was taken with, defender positions, the type of chance creation, and much more. The standard definition of xG is independent of the identities of players involved, and our dataset isn't large enough for such considerations regardless so we quickly ruled team and player info out.

Feature Engineering

One important feature we recognized the dataset was missing was the shot angle: the angle between the shot-taker and both of the goal posts. The wider this angle is, the wider the range of area the shot can be aimed at.

```
def angle(x, y):
    v1 = np.array([120, 36]) - float(x)
    v2 = np.array([120, 44]) - float(y)
    cos_t = np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
    cos_t = np.clip(cos_t, -1.0, 1.0)
    return Decimal(np.degrees(np.arccos(cos_t)))
```

Using the position of the goalposts, we can find this angle as another helpful measure of how good a shot position is.

After the exclusion of a few more categorical variables we thought weren't very relevant and would have required one-hot encodings that would dramatically increase the length of our feature vectors we were left with the the following:

```
0-1: shot coordinates
2: shot angle (engineered)
3: header (binary)
4: left foot (binary)
5: right foot (binary)
6-7: goalie coordinates
8-27: defender coordinates sorted by closeness to shot
28-47: teammate coordinates sorted by closeness to shot
```

The source dataset was quite large, so using standard json loading libraries and indexing for the desired info was an extremely slow process. Because

each json file in the matches folder contained many non-shot events we weren't interested in, we took the time to just write our own json-to-numpy parser that only loaded shot objects and skipped past everything else to make the process a lot faster. We went through several different versions of our feature vectors so streamlining this process in a way that allowed us to clean the data quickly and easily paid off immensely. The cleaned dataset with our chosen features is in our project repository, but the dataset parser is also included for selection of other features.

Logistic Regression

Given a problem where we have a bunch of features we want to use to predict probabilities a given item belongs to a particular class, logistic regression is an obvious place to start. Considering the size of the dataset and number of features, we went straight to implementing a logistic regression model with a gradient descent optimizer.

```
model_ours.fit(x_train,
               y_train,
               T=6000,
               alpha=8e-5,
               eta_decay_factor=1e-5,
               batch_size=8000,
               optimizer_type='stochastic_gradient_descent')
```

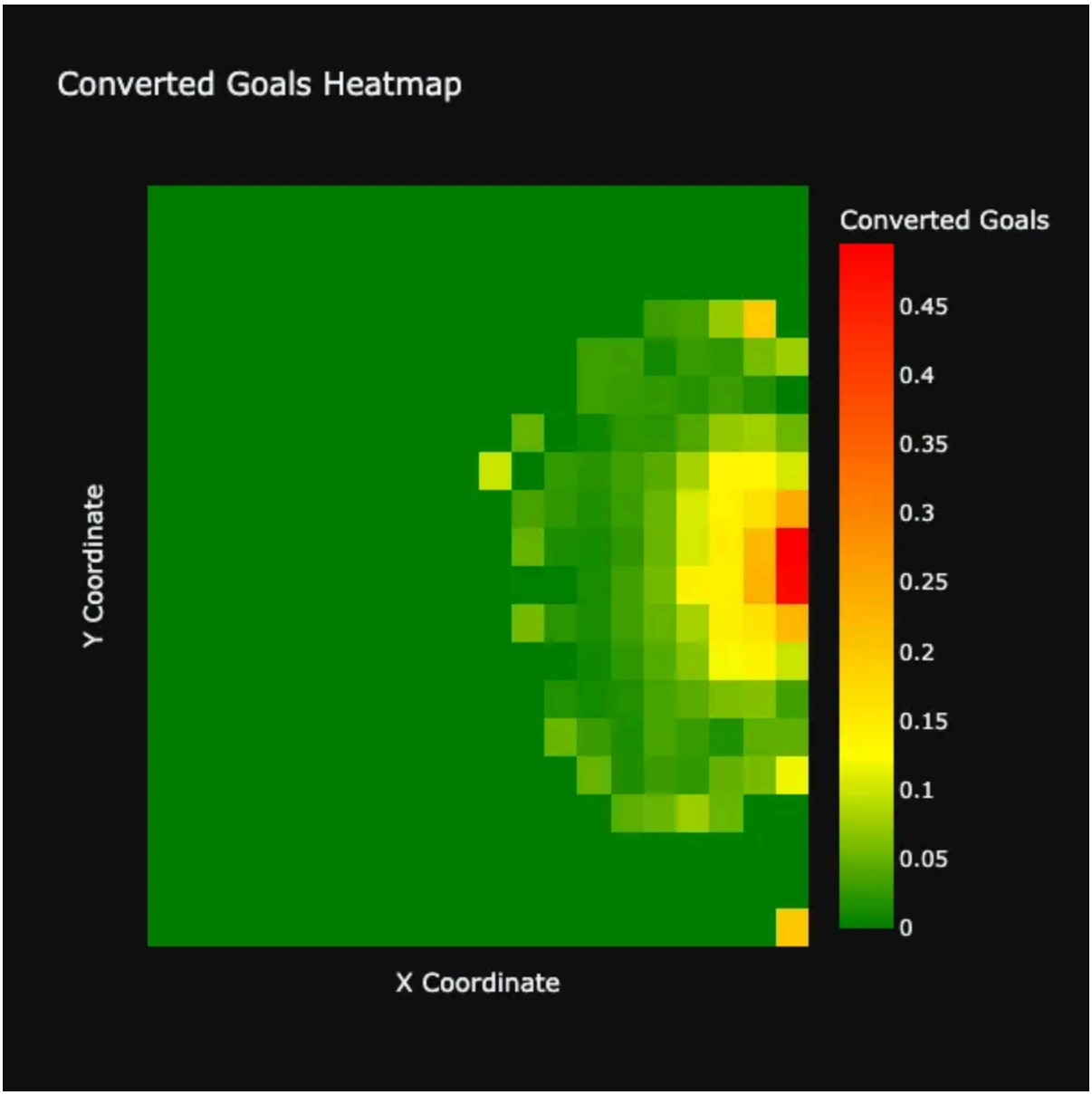
With a few quick hyper-parameter adjustments, we quickly approached the loss of StatsBomb's model using all 48 features of our cleaned dataset.

```
Total training time: 32.457503 seconds
Training Log-loss of Our Model: 0.2938
```

```
Training Log-loss of StatsBomb Model: 0.2633
Validation Log-loss of Our Model: 0.2888
Validation Log-loss of StatsBomb Model: 0.2561
Testing Log-loss of Our Model: 0.2974
Testing Log-loss of StatsBomb Model: 0.2665
```

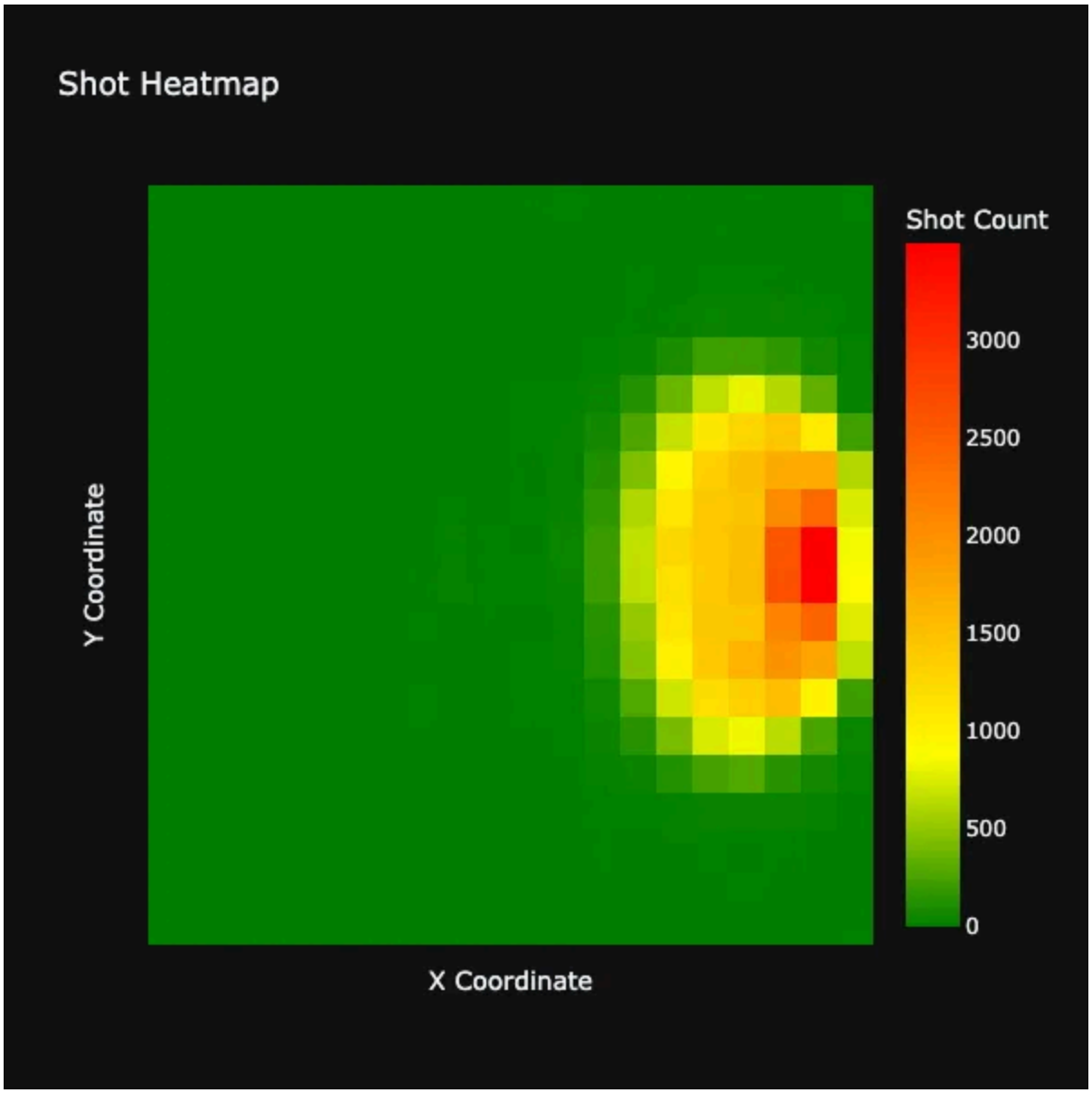
This was kind of surprising, given the complexity of predicting the xG of a given shot. We anticipated a very poor result from a Logistic Regression model, with the understanding that it only evaluates each feature independently. It shouldn't be able to say that a shot position is good relative to the shot position of the goalie. And for a game in which there are very few objectively good shot positions (only if you are right in front of the goal), and the success of a shot is completely dependent on where the shot taker is relative to the goalie, we were confused by these results.

More careful analysis showed our instincts about Logistic Regression's weaknesses for calculating xG were right; we were just ignoring the full context of the data.



Heatmap of shot conversion from our dataset. Code found in data_visualizations.ipynb

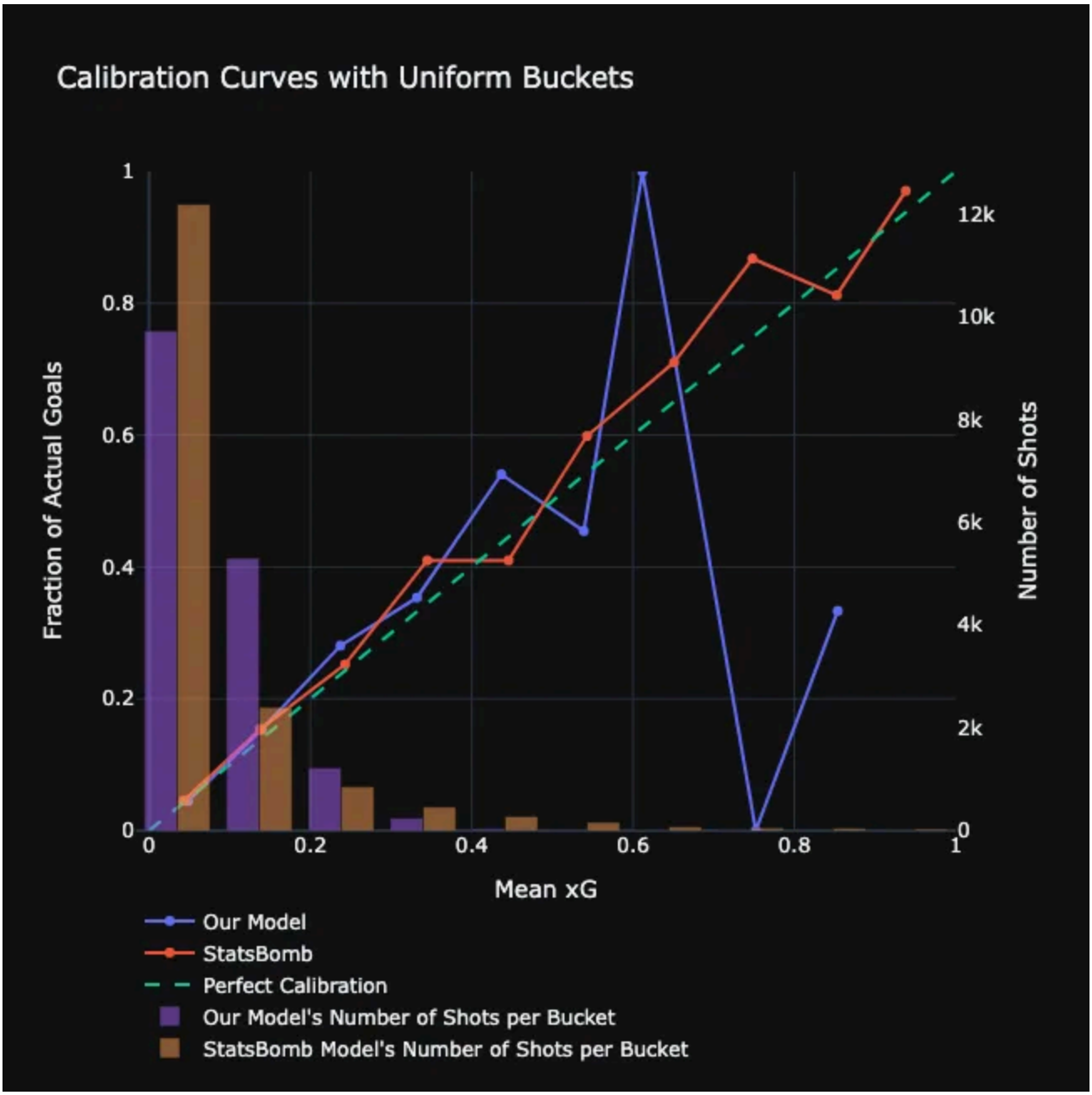
A heatmap of our dataset plotting shot position alongside the conversion rate from that position shows that there is a small area right in front of the goal that has a pretty good conversion rate. Other than that, most positions are pretty bad ignoring all other features.



Heatmap of shots in our dataset. Code found in data_visualizations.ipynb

When we compare this with a heatmap of where shots tend to be taken from in general, we start to see the big picture — we are dealing with a classification problem in which the vast majority of shots aren't goals. Even worse, making high confidence predictions for goals isn't straightforward unless the shot is taken right in front of the goal, a very rare occurrence.

To confirm our suspicions, we plotted our Logistic Regression's predictions onto a calibration curve.



Calibration curve for our Logistic Regression model. Code found in data_visualizations.ipynb.

Basically, this graph divides all our predictions into equal-width buckets. The purple line shows our model's calibration ie. for all the shots we predicted to have an xG between 0–0.1, 0.1–0.2, etc., the hope is that the actual proportion of goals in that bucket is close to the mean predicted xG of shots in that bucket. And we can see that our model is pretty well calibrated for low-xG shots, which makes sense given the heatmap showing that most shots taken from most locations have a low chance of going in, something that a Logistic Regression could easily pick up on.

We can see where the Logistic Regression begins to fail by looking at the calibration for higher xG shots. There are far fewer shots in these buckets, but the StatsBomb model still manages to stay well-calibrated despite the low sample size. With the properties of a Logistic Regression model in mind, we can hypothesize about where it is falling short.

As shown in the above heatmap, even shots taken right in front of the goal don't have much more than a coinflip chance of going in. This is largely because of the fact that more often than not there is a goalie in between the ball and the back of the net. Often the goalie isn't in a great spot to make a save, so the ball still goes in much of the time. Unfortunately, a Logistic Regression model isn't really equipped to know if such a thing is happening, as it evaluates each feature independently. In the context of our classification problem, a shot from the left is great if the goalie is way to the right, and vice versa. But if both are on the same side the ball is likely going straight into the goalie's hands.

To make high confidence predictions for goals and make significant improvements to the model, we will need to explore deep learning methods.

Neural Network

With the following hyper-parameters and hidden layer architecture, we quickly improved upon the Logistic Regression model.

```
# Batch size - number of shots within a training batch of one training iteration
N_BATCH = 200

# Training epoch - number of passes through the full training dataset
N_EPOCH = 35

# Learning rate - step size to update parameters
```

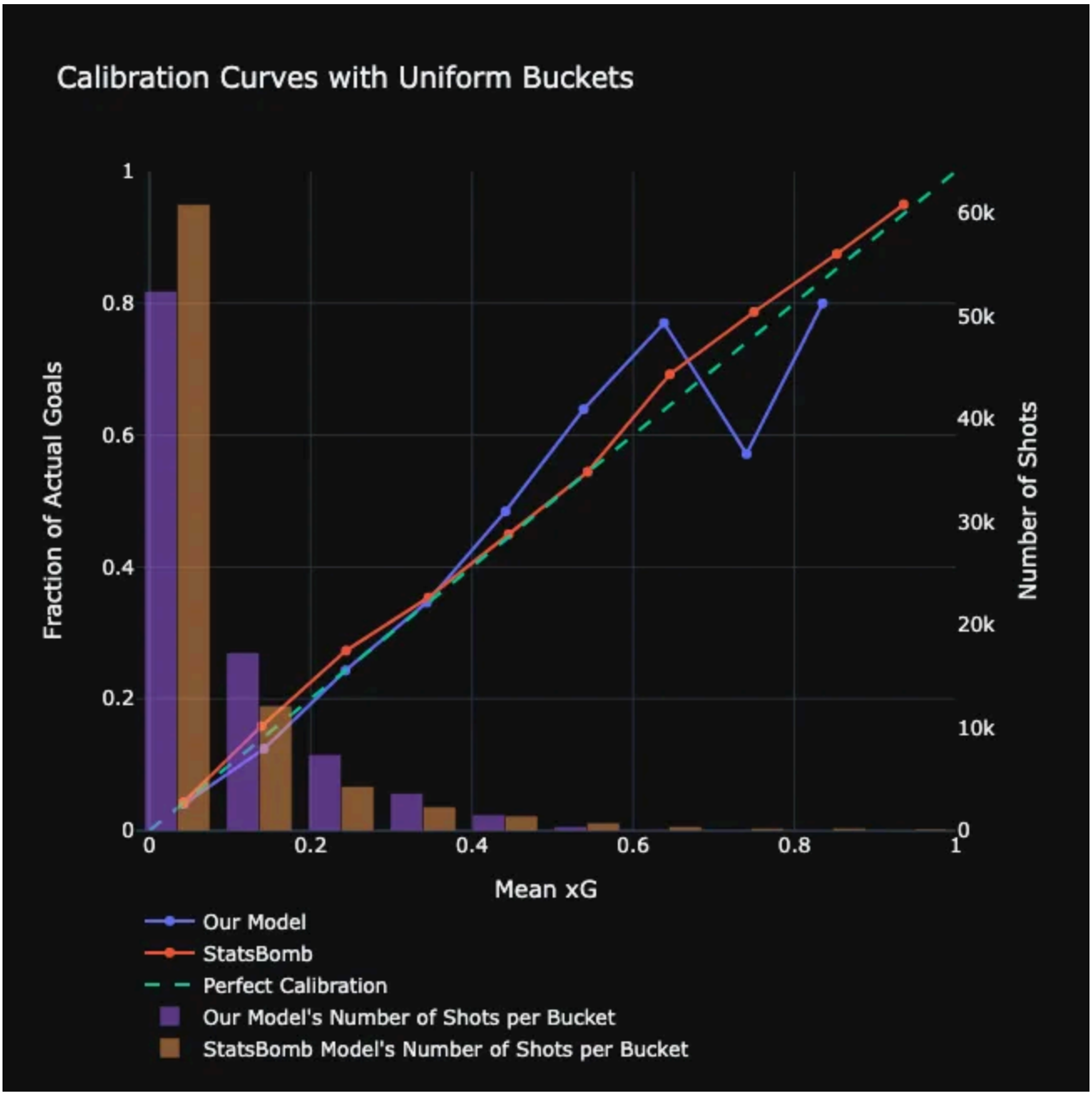
```
LEARNING_RATE = 0.01

# Learning rate decay - scaling factor to decrease learning rate at the end of e
LEARNING_RATE_DECAY = 0.75

# Learning rate decay period - number of epochs before reducing/decaying learnin
LEARNING_RATE_DECAY_PERIOD = 4
```

```
def __init__(self, n_input_feature, n_output):
        super(NeuralNetwork, self).__init__()

        self.fully_connected_layer_1 = torch.nn.Linear(n_input_feature, 128)
        self.fully_connected_layer_2 = torch.nn.Linear(128, 256)
        self.fully_connected_layer_3 = torch.nn.Linear(256, 512)
        self.fully_connected_layer_4 = torch.nn.Linear(512, 1024)

        self.output = torch.nn.Linear(1024, n_output)
```

As expected, a deep learning model was much better equipped to deal with the complex relationships between features. Without much effort, the loss quickly dropped much closer to that of the StatsBomb model.

```
Total training time: 80.880398 seconds
Testing Log-loss of Our Model: 0.2760
Testing Log-loss of StatsBomb Model: 0.2625
```

We were most happy with the results once the calibration curve loaded and we could see that the Neural Network was doing a much better job at assigning high confidence predictions to shots.



Calibration curve for our Neural Network model. Code found in data_visualizations.ipynb.

Not perfect, but far better than what we saw with the Logistic Regression.

Results

In our proposal, we discussed a measure of success as hoping to predict the proportion of goals converted accurately to within some percent (8% for LR and 5% for NN). Both of these targets were fairly consistently met for the Logistic Regression, and very consistently met for the Neural Network.

For the above Logistic Regression training run, we under-predicted goals by only 5.6632%. Having already seen the calibration curve for this particular run, a sizable under-prediction isn't surprising, and repeated training runs of the model consistently yield an under-prediction of varying magnitude (most often below the 8% target).

For the above Neural Network training run, we actually over-predicted goals by only 3.6831%. Repeated training runs of the model vary pretty evenly in whether it over or under-predicts, and consistently do so at a low rate below the 5% target.

While we were very happy with these results, shortly after submitting our proposal, we realized that this on its own wasn't the best measure of success. As alluded to previously, you can have a pretty poor xG model that meets this criteria — you could just always predict the base rate for every shot. As such, we moved towards these calibration curves as an additional measure of success; we don't want to just classify goals and non-goals at an accurate rate in the long term. Such a model would be useless for players and coaches that want to know what kinds of shots have the best chance of hitting the back of the net.

Revised Measures of Success

As such, while we met our initial measures of success, we also hold ourselves to this higher standard. Because we trained both models using Log-loss, we knew that we weren't going to run into a problem of always predicting the base rate, but we still wanted to measure how calibrated each of our models were. For our Logistic Regression model, to no fault of our own, we fall very short. It just doesn't have the tools required to be well-calibrated for high xG shots. In the 0.7–0.8 xG bucket, our model's predictions weren't within 50% of ground truth let alone 8%. There is a very low sample size for these shots,

but nonetheless Logistic Regression just doesn't cut it for this type of problem.

The Neural Network, on the other hand, did a lot better. When grouped by buckets it still fell a little short of our ambitious 5% target for a few of the high-probability buckets (again, low sample size), but still did a lot better than the Logistic Regression at making high-confidence predictions — something extremely important for this type of problem.

Conclusion

Unsurprisingly our model falls short of the StatsBomb model. However, its results are not far off. With a larger dataset, more feature engineering, and further hyper-parameter tuning, a better xG model is feasible. We suspect a Neural Network would be the best approach, especially with a larger dataset, but are also interested in trying other models such as a Random Forest classifier that would also allow for analysis of the importance of each of the features.

The biggest challenge we faced was getting a good dataset. After several attempts at working with other sources, we eventually stumbled across StatsBomb's dataset which still required a lot of time to load, engineer, and clean into the feature vectors we were looking for, but ended up being a great dataset with a lot of datapoints.

This classification problem was an excellent exercise in developing reasoning about the data involved, which models are good fits and why, and how to best measure the success of different machine learning methods.

https://github.com/fisherm123/xg_model

Unlisted

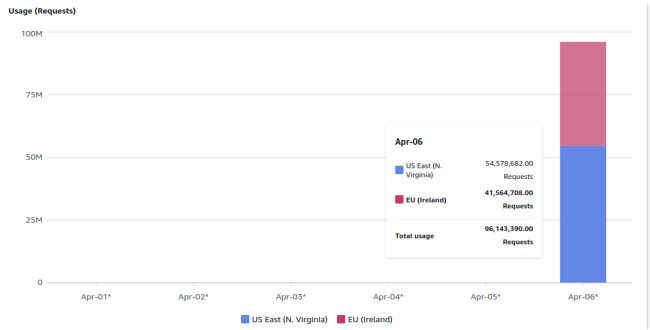
F

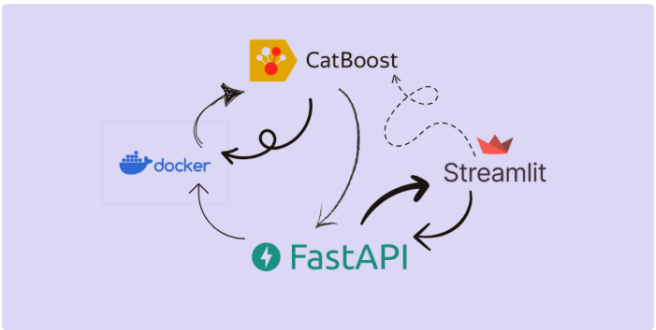
Written by Fisher Marks

Edit profile

0 Followers

Recommended from Medium





Maciej Pocwierz

Ramazan Olmez