Bookies Beating Us to the Draw

A study of whether a machine learning based betting strategy can exploit a bias in bookmakers' odds for football draws to earn positive returns. It can't.

Contents

1. Introduction: How do we intend to beat the bookies?

With an increasing number of online sports betting firms, the ease of which one can bet on sporting events has increased considerably through the last years. In a high risk setting such as sports betting some gamblers are bound to make a profit, whereas others will serve a loss. But one thing is certain, for the market to exist it has to be the case that bookies win on average, otherwise there is no profit to be gained, which would result in a market break down. To ensure an average expected profit the bookmakers must have a reasonable good prediction model, that helps them set the right probability of a given outcome. But might it be possible to beat the prediction model of the bookies, and earn a profit on some specific bets?

With a toolbox of easily implementable machine learning models, one is tempted to use the sheer amount of available data to develop an algorithmic betting strategy which attempts this. However, it is wise to consider whether it is really realistic to develop a better model than professional bookies, who likely have access to both better data and equally skilled data scientists. One would need a strategy for how to beat them.

In this paper we evaluate one such. Particularly, we investigate a hypothesis put forward by the mathematician David Sumpter that bookies deliberately set odds too high for draws, because people dislike betting on draws rather than one team winning. We expand further on this hypothesis in section 1.1. If we can locate matches where odds are too high, we should be able to earn money by betting on them. Locating them requires developing probability estimates of our own. However, these only need to match those of bookies, rather than to beat them, if there truly is a bias. Section 2.1. develops a simple statistical model for football draws based on teams propensity to score goals and their relative skill, as well as external features such as league and weather. The key question of the paper is then: Betting based on the probability estimates of the model, can we make money?

We investigated the hypothesis on 25,000 football matches across several European leagues from 2008-2016. The data covers both match specific statistics, as well as odds from several online betting platforms. Using the match specific statics, we compute performance metrics for the different teams. As additional covariates, the so-called Elo-rating is used as a measure for the team's relative strength. In addition, we include a measure for the weather condition on the different match days, to test whether draws are more likely when it rains.

Going from data to probabilities requires estimating models. We estimate three machine learning models of varying complexity on the data: Logistic regression, random forests and neural networks. We assume familiarity with these algorithms, so descriptions of their inner workings are relegated to appendix B. Our modelling strategy is described in some detail in section 3. The overall goal of modelling was to maximize the pay-off implied by a betting strategy based on the algorithm. We ensured a good estimate of each strategy's generalization performance by setting aside a pristine test set before we started trying out models. We then found optimal versions of the algorithms on the training data by standardizing and imputing data, and finding optimal hyper-parameters using cross-validated grid search.

Section 4 presents our main results: The approach does not make money. Out-of-sample performance of all chosen strategies produced negative results. We also investigated whether this might be different for Premier League matches separately, as suggested by the article author, but

again found negative profits on the test set. We believe our models provide decent probability estimates given the statistical model and data used. This suggests either that bookie odds are not biased as hypothesized, or that a more complex model is necessary to take advantage of this.

Our analysis does not conclusively state that no bias exists. The probability distributions implied by odds could be consistent with both biased and unbiased odds. However, as our models replicate several specific features of the implied odds, we believe they provide a good estimate of true probabilities. Since our results do not indicate biased odds, we do not believe odds are truly biased. However, one could try more complex models to verify this. Until then, we would not suggest betting indiscriminately on draws.

1.1. Motivation: Bookies might deliberately set odds for football draws too high

Is it feasible to beat the bookies? The odds of bookies are developed based on expert knowledge and, one would expect, quite sophisticated statistical models. Developing probability estimates which are more accurate than those underlying the offered odds might not be easy. However, it may be that bookies have other goals besides accuracy - particularly, they are interested in spreading bets across outcomes, to minimize the risk of big payouts. We intend to explore one systematic way that bookies, seeking to minimize risk, might set odds differently than implied by their probability estimates.

It was suggested by David Sumpter, a mathematician writing for The Economist (2016). He investigated several possible biases of bookies, and found the most reliable to be that 'punters don't like backing draws in big matches' (ibid., p. 4). The notion is that betters like to see a win in one direction or the other, and hence dislike betting on draws. To make up for this mismatch, the bookmakers will adjust accordingly by increasing the odds, hoping to draw betters there.

Another way to formulate this is by considering that odds reflect the bookmakers' view on the outcome's likelihood. Imagine, for instance, you place a bet on an outcome with odds 3, corresponding to the fractional odds 2/1. Here, the odds imply that the bookmakers think that the probability of a winning outcome is $1/(2+1) = 1/3$, i.e. the number of favorable outcomes relative to the number of outcomes. If there in fact is a bias against backing draws, we would expect the odds to be too high, corresponding to a too low implied probability.

One could exploit this bias by predicting the probabilities for draws and betting on matches that appear to be overvalued, i.e. where the predicted probability exceeds the probability implied by the odds. Sumpter claims to have earned positive profits mainly by backing draws (ibid., p. 4). This is the idea this paper picks up on. Using different more or less advanced prediction models, we will take up the challenge of modeling the probability of draws in football matches in order to set up a betting strategy that aims at exploiting the bias against draws. Since we are exploiting a bias, our models do not actually have to beat those of bookies - we just have to match their underlying probability estimates, figure out where they offer biased odds, and bet there. If the bias exists, we should make money.

2. Statistical model & data: Which probabilities are we interested in?

Our hypothesis is that for some matches, the odds are greater than implied by the true probability. If we find and bet on them, we should make money in expectation. However, this requires finding the actual probabilities, so we can compare them with the odds. The first section below develops a statistical model with this goal, and the second substantiates the theoretical concepts with actual data and variables.

2.1. Statistical model of draws: Goals & skill

Football matches may end in three ways: A draw, a win for the home team or a win for the away team. Given our hypothesis, we are solely interested in the first outcome. We have therefore chosen to model this is a binary prediction problem, rather than multiclass[1].

$$y_i = \begin{cases} 1, & \text{Draw} \\ 0, & \text{Not draw} \end{cases} \qquad p_i = \mathrm{P}\left(y_i = 1|\boldsymbol{x_i}\right) = \mathrm{E}\left[y_i|\boldsymbol{x_i}\right] = F\left(g(\boldsymbol{x_i})\right) \tag{2.1}$$

The distribution of binary data is necessarily Bernoulli - There must exist a probability of success p_i, and the complementary probability is bounded by $1 - p_i$ (Cameron and Trivedi 2005, p. 463). This implies that we are interested in the conditional expectation, which is defined for any $\boldsymbol{x_i}$ whether relevant or not (if irrelevant, we would simply have $p_i = E[y_i|\boldsymbol{x_i}] = E[y_i]$). In traditional econometrics, the functional forms of $F(\cdot)$ and $g(\cdot)$ would also be of interest, likely to fuel some maximum likelihood estimation. However, in data science we mainly focus on providing empirically validated approximations, and exact functional forms are of little interest. The tricky part then is to define an interesting conditional expectation function (CEF) to estimate[2].

So, which features might be interesting to include? The first thing to consider is how the outcome of a game is determined. In football, it only depends on the goals scored:

$$\mathrm{P}(y_i = 1|\boldsymbol{x_i}) = \mathrm{P}(\text{home goals} = \text{away goals}|\boldsymbol{x_i}) \tag{2.2}$$

Goal propensity. One obvious thing to model then is how likely each team is to score goals. Consider a simple example: Let each team have 25 pct. of scoring each outcome between 0-3 goals, and let goals be independent from one another. The probability of a draw is then 25 pct.[3] If they instead have 20 pct. chance of scoring 0-4 goals, the probability of a draw is 20 pct. Spreading the possible goal combinations decreases draw probabilities. Thus, matches where teams are more likely to score goals should be less likely to end in draws.

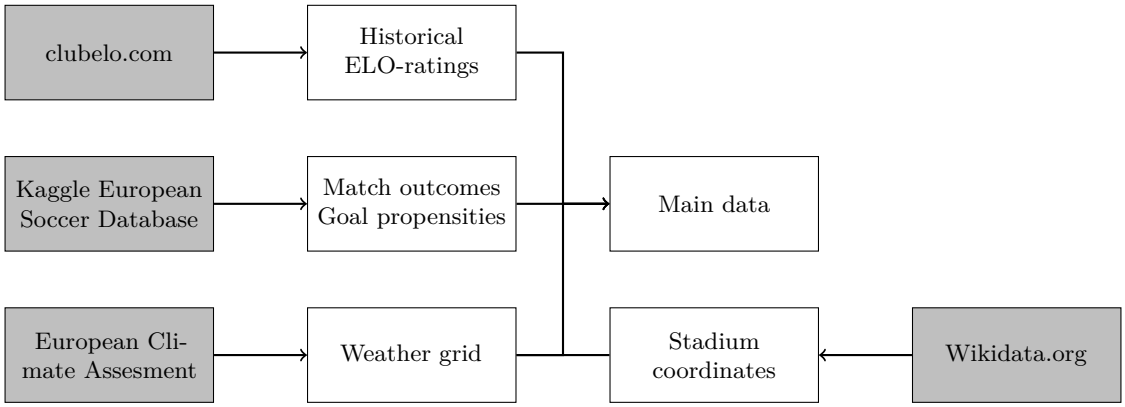
There are two main ways goal propensities could vary. First, some teams may tend to score more goals than others, either because they are better at football, or more *aggressive*. Similarly, some teams may have strong *defense*, and tend to prevent other teams from scoring on them.

[1] It is not given this will yield better results. It may be that modelling home and away wins separately allows an algorithm to realize something about the underlying DGP otherwise missed, and hence provide better draw probabilites. Or it might force the algorithm to learn an needlessly complex function, and hence perform worse.

[2] The difference from microeconometrics is that here interesting simply implies maximizing predictive power, without the usual concerns of exogenous variation to fuel estimation of causal effects (Angrist and Pischke 2008).

[3] The probability of any one draw is the product of probabilities $\forall i : \mathrm{P}(\text{Home} = i \wedge \text{Away} = i) = \mathrm{P}(\text{Home} = i)\,\mathrm{P}(\text{Away} = i) = 1/8$. Summing over $i \in \{0, 1, 2, 3\}$ yields the total probability.

Figure 2.1: *Data sources*



Secondly, there may be reasons outside the team which affect propensities. One candidate is that teams in some *leagues* tend to score more goals. We have also formulated a slightly more 'interesting' hypothesis: It may be that goals are less likely in bad weather, as more water on the field makes the game more physically challenging, lowering the number of shots on target[4]. If so, observing (a forecast of) match weather could inform as to the draw probability.

Relative skill. Another obvious contender is how well teams are matched. A mid level team can score a lot of goals on average, but still have a low probability of beating a high level team. The model should therefore account for whether there is a clear favorite. To keep the model parsimonious, we focus on these main features in our statistical model.

2.2. Data: Match outcomes, odds, average goals, Elo-ratings & whether it rained

Even with these few theoretical features, we could easily have flooded the model with data. Uncritical data scientists might for instance look at 'relative skill', and simply feed the algorithm every performance measure they could find. This is a recipe for overfitting. We instead again chose a parsimonious approach, and looked for a few summary measures of each. This section briefly describes the data sources and gathering process, as well as the final data set. It is summarized in figure 2.1, with gray boxes indicating external data sources.

2.2.1. Match outcomes & odds

Fortunately, the popularity of football ensures easy data access. To obtain data on match outcomes and odds, we used a database gathered by users of *Kaggle.com*, a popular site for data science contests[5]. It includes about 25,000 matches from 11 European leagues between 2008-2016. Outcomes were provided for all matches, and betting odds were available from a number of betting provides. Assuming full information, one would always pick the best available odds - We therefore use the highest odds for each match to evaluate betting strategies.

2.2.2. Goal propensities & leagues

The database includes an abundance of information about the teams. There is far more data than necessary for our approach, so we only extracted part of the database. To get a measure of

[4]One could also argue that goals could be more likely, if goalies/defense are more affected.
[5]https://www.kaggle.com/hugomathien/soccer

the propensity of each team to score goals (of their general 'aggresiveness', if you will), one can look at average goals scored. However, it is important to remember that in order to be relevant for betting, data should be available before the match. We therefore calculated an $m - period$ running average of goals scored in previous matches. For team i at time t this would be:

$$\text{Agg}_{\text{i,t}} = \frac{1}{m} \sum_{j=1}^{m} \text{Goals}_{i,t-j} \tag{2.3}$$

To get a measure of their defensiveness, the exact same measure is calculated with goals against the team, rather than by the team. Higher values of both measures indicate greater goal propensities, and should lower the probability of draws.

The running average allows us to exploit cross-sectional variance across teams. But a neat bonus feature of the running average is that it let's us discard *out of date* observations. Since there is a fair amount of change on a given team, a teams current success may be more or less independent of their success ten years ago. Discarding older periods from the averages allows us to exploit this more. We therefore set a maximum period of $m = 20$ matches.

The database also provided data on which league each game was played in (there is no data on cross-league games). There seemed to be differences in the share of averages in each league, so we included dummies for 11 leagues in the modelling.

2.2.3. Elo ratings

We also needed a measure of the general ability of each team. We could possibly have estimated this based on historical performance, but this would be troublesome, and require discarding data for a burn-in period. We instead looked to *Elo-ratings*. They provide a summary measure of the relative skill of each football team at a given point in time[6].

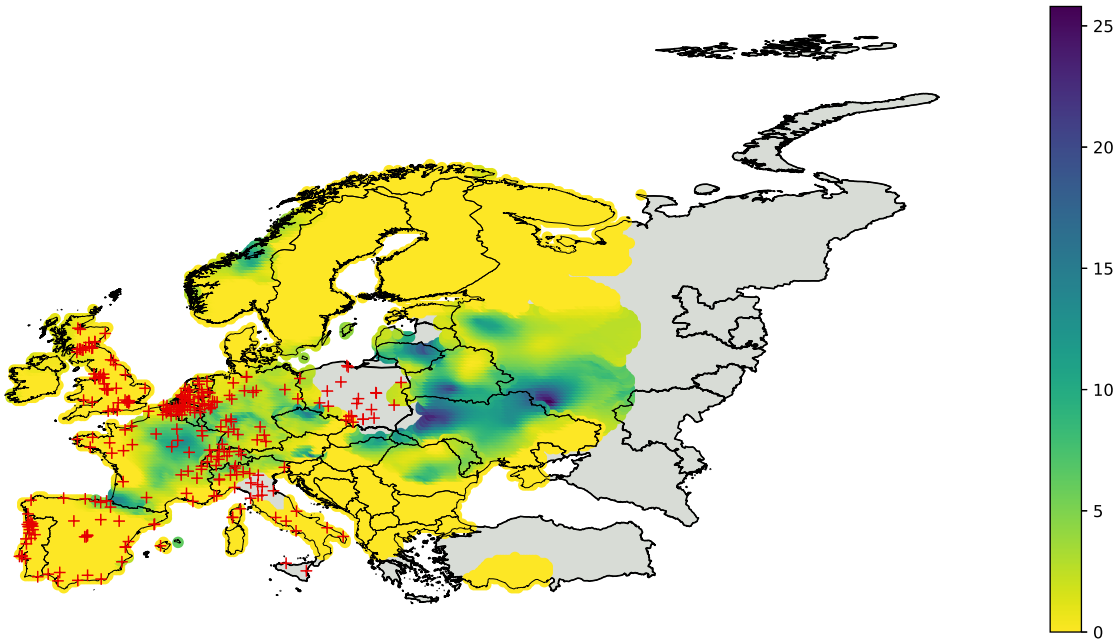
We scraped Elo-ratings from *clubelo.com*. The site provides historical Elo-ratings going back far more years than we need. Each covered team has their own page, so we designed a simple scraper which visited the site of each of a list of teams (also collected from the site), and stored the results. We encountered some difficulties in merging the data together since the names of football teams are typically logged slightly differently on different sites. We used some *regex* code to remove parts of club names which might differ (say, whether you write F.C., FC or exclude it), but the process still required significant manual matching. In the end, we managed to provide Elo-ratings for a large number of matches. Matches in lower leagues are not featured on *clubelo*, so we do have a missing data problem. We return to this in the section on imputation below.

2.2.4. Weather data

Finally, we looked at whether we could gather weather data for the matches. To keep things simple, we simply focused on how much it rained during each. Figuring out whether it rained during 25,000 football matches is slightly tricky, but easier than one might think.

[6]If two teams with the same Elo compete, each should have a 50 pct. chance of winning. After the match, the Elo-rating of the winning team increases, and the losing team's decreases. Elo-ratings increase more if you defeat a team with a higher Elo-rating, and decrease more if you are bested by a lower-ranked team.

Figure 2.2: *Weather data & stadiums for July 1st 2017*



Note: Each red dot represents each stadium in our data. The grid is colored by how many millimeters of rain each area got on July 1st 2017. Grey areas indicate missing data. Wow. What a plot! (cockney accent).

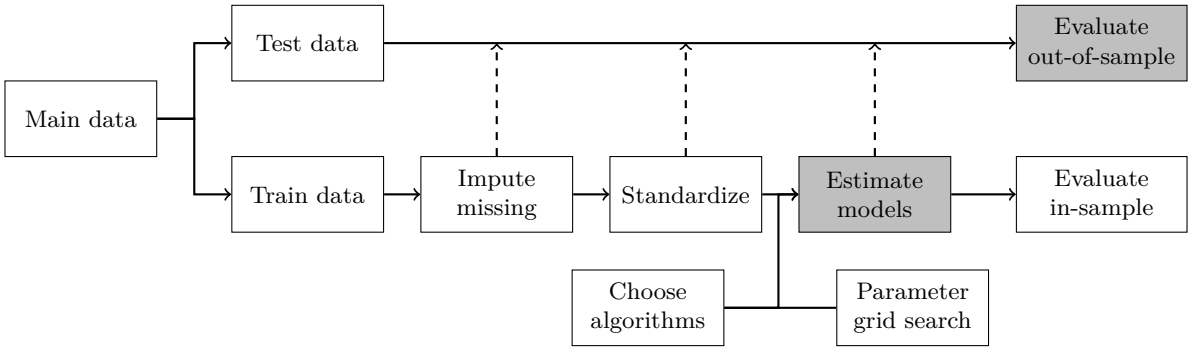
The European Climate Assesment & Dataset project provides data from various weather stations in Europe, leading to a dataset with daily precipation on a 0.25 latitude-longitude grid basis). We matched this with matches by finding the coordinates of the football stadium of the home team using *Wikidata.org*. Again, there were some issues with matching team names, but we persevered. In the end, stadium coordinates were located for all but a few teams. Some grid point had missing values – In these cases, we allowed values values to be carried over from neighbouring grids within a one degree distance. In the end, we obtained precipitation for about four fifths of the matches. An example is shown in figure 2.2. It shows the weather in Europe on a random date (2017-07-01), where it seems to have rained in France and Central Europe. Gray areas denote countries for which we do not have weather data. We have also indicated the stadiums for which there are matches in our datasets. We have similar data for every day in our data period, allowing us to state match weather based on the stadium location.

Table 2.1: *Data overview*

Feature	Variable	Variation across			
		Team	Time	League	Match
Relative skill	Elo rating	x	x		
Goal propensity	Average goals scored	x	x		
(internal)	Average goals suffered	x	x		
Goal propensity	League			x	
(external)	Precipitation		x		x

Notes: There are separate variables for home and away team for Elo and avg. goals.

Figure 3.1: *Modelling pipeline*



3. Modelling: How do we ensure that we model probabilities well?

The purpose of the analysis is to investigate whether a parsimonious algorithm betting on draws can confidently expect to make money. The first section defines a relevant performance measure based on this. The subsequent sections go through our modelling pipeline as shown in figure 3.1.

3.1. Evaluating performance using a custom scorer (betting pay-off)

Draws are low-probability events: Since draws are rare it is likely that the true probability is rarely above 50 pct. Pure classification is not very useful then, since almost all matches are designated as non-draws. When evaluating a given model's performance we will hence depart from using common accuracy metrics and use a more probabilistic measure instead. To this end, following the hypothesis of biased odds, we consider the net profit implied by a betting strategy that aims at exploiting the bias. We will place a bet on match i if the predicted draw probability,p^i_{pred}, exceeds the probability implied by the odds, p^i_{imp}. In total, the number of placed bets, b, is then given by

$$b = \sum_{i=1}^{n} \mathbb{1}\left\{p^i_{\text{pred}} > p^i_{\text{imp}}\right\}, \tag{3.1}$$

where $\mathbb{1}\{\texttt{x}\}$ is an indicator function with value one if condition `x` is met. If the outcome of match i goes our way, $y_i = 1$, the bet will trigger a payoff equal to the corresponding odds, odds_i. The total earning of the betting strategy can be computed as

$$\text{Earnings} = \sum_{i=1}^{n} \text{odds}_i \times \mathbb{1}\left\{p^i_{\text{pred}} > p^i_{\text{imp}}\right\} \times \mathbb{1}\left\{y_i = 1\right\}. \tag{3.2}$$

Now, if we suppose that we in each bet place one Euro, the average net profit implied by the betting strategy is given by

$$\Pi = \underbrace{\frac{1}{b}\sum_{i=1}^{n} \text{odds}_i \times \mathbb{1}\left\{p^i_{\text{pred}} > p^i_{\text{imp}}\right\} \times \mathbb{1}\left\{y_i = 1\right\}}_{\text{Average earnings}} \underbrace{- 1}_{\text{Average cost}} \tag{3.3}$$

If a bias against draws exists and if the model is able to provide more accurate probabilities, we should earn a positive net profit, as we systematically will bet on matches that are overvalued.

3.2. Measuring generalization performance

A key goal of machine learning is to ensure good generalization performance. If we encounter a new sample of (unlabelled) data from the same data process and use our algorithm to bet on these, then we'd like the algorithm to perform well[7]. The following sections will discuss how to ensure this. First, however, we should discuss how to measure generalization performance, because a critical step occurs at the beginning of the analysis.

The basic point is that measuring performance on the data an algorithm is fitted on, overestimates performance because complex algorithms may **overfit** the data. Loosely speaking, this implies that a model learns to replicate the noise in observed data rather than the underlying data-generating process[8]. The easiest way to check for overfitting is to evaluate on data the model has not seen yet. Before we fitted any models, we divided our dataset randomly, keeping 70 pct. to train the algorithms on, and setting aside 30 pct. in a **test set**. Since we did not use the test data for any part of our algorithm tuning, performance on it provides an unbiased estimate of generalization performance (Abu-Mostafa, Magdon-Ismail, and Lin 2012a, pp. 59-60).

We also need a way to choose between algorithms. One could use a similar approach, and set aside a validation set of the train data. Given enough data, holding out a small sample to validate models on wouldn't be a problem, but often data is sparse, and we might miss important features by excluding more data from the estimation process. We instead chose to use **K-fold cross validation**, which provides an alternate method for estimating prediction errors and is a useful tool to evaluate how well our model predicts in expectation (Hastie, Tibshirani, and Friedman 2001, pp. 241-249). Specifically, K-fold cross validation partitions the data into K equal folds, then for $k = 1, \ldots, K$ estimate the model an all other folds than k, and compute the prediction error for the observations in fold k.

3.3. Preprocessing data

Data is rarely perfect, and one typically needs to tweak it slightly to ensure modelling goes smoothly. We perform two preprocessing steps: Imputation & standardization.

3.3.1. Imputation

Out of our 18,151 unique matches some information is missing in 7,632 of them, corresponding to 42% of all games. If left alone, the data we train on will only use a little more than half of our available data. Thus, some consideration should be spend on missing data.

Data can either be *missing at random* or missing due to some underlying feature of the data, e.g. if a teams Elo-rating is missing in one match, it's missing in all of the matches, because the measure is missing for the lower league teams. The least complex way to impute missing data is to either use the mean or median on each of the nonmissing features. As soon as the feature are moderately dependent, a better approach is to estimate a model that predicts the missing features from the known elements in data (ibid., pp. 332-333). The most common way to do this is multiple imputation by chained equations (MICE), though it's important to note that this

[7] More precisely, for good generalization performance we'd just like the algorithm to perform *as well as* on the training data. Bad performance can generalize well.

[8] For a more thorough definition, see Goodfellow, Bengio, and Courville 2016, pp. 105-113 or Abu-Mostafa, Magdon-Ismail, and Lin 2012a, pp. 119-126.

method shouldn't be used if the *missing at random* condition is not fulfilled. As this is the case with some of our data, we will be using the simple imputation method. Specifically, we add the mean of each co-variate when data is missing. For the team specific observations, such as home aggressiveness, we add the conditional mean of each team, instead of the unconditional mean. The imputations is done after the data is split into test and training sets, otherwise the training data is contaminated with information that should have been unknown. When the test set is evaluated, we use the imputation values from the training set, rather than imputing on the test set, again because the test set should remain unknown to.

3.3.2. Standardization

Several algorithms are vulnerable to the scale of input variables. The issue is that optimization may fail to find the optimal version of the algorithm. Very large features may induce quite small weights, leading to them to be approximated by zero (*underflow*) and getting the optimizer stuck. Alternatively, small features may yield very large weights and possible *overflow*.

There are two main approaches. First, we may *normalize* the data, either by scaling it to a $[0, 1]$ range or to a norm length of one. However, this does not prevent very small values (and may even cause it, particularly with outliers). We instead employ *standardization*, where the data is scaled to mean-zero, unit variance. This was done for all variables (except dummies).

3.4. Managing model complexity

Now the fun part! This section details our approach to modelling draw probabilities. There are basically two goals (Abu-Mostafa, Magdon-Ismail, and Lin 2012a, p. 26):

1. Ensure that the model performs well (minimize underfitting)
2. Ensure good generalization performance (minimize overfitting)

Resolution of these is done through proper management of **model complexity**: More complex models can fit more complex relationships (preventing overfitting), but could also fit the algorithm to noise rather than signal (creating overfitting)[9]. A first consideration is then that complexity should match the complexity of the statistical relationship. This is too simple however - if data is limited or very noisy, it may be better to fit a simple approximation quite precisely, than a complex noise-repeater. A more precise answer therefore is that 'model complexity should match the *quantity* and *quality* of the data we have (ibid., p. 122). We have plenty data but as seen above, there are some issues with it. We take a pragmatic approach: Try various levels of model complexity, and evaluate with cross-validation.

3.4.1. Choice of algorithms

Different algorithms naturally encompass different levels of model complexity. The toolbox of machine learning ranges simple linear estimators (which tends to underfit, but generalizes well) to non-parametric universal approximators (e.g. neural networks). We went with three popular approaches of increasing complexity (see appendix B for a more precise definition of each):

[9]Formally, the conflict is often summarized in the *bias-variance tradeoff*: Complex models are less biased since they can properly model the underlying relationship, but exhibit higher variance because they are more affected by small data fluctuations (for a more formal definition, see Hastie, Tibshirani, and Friedman 2001, pp. 223-228).

Logistic regression (Logit) provides a simple linear estimator. It works well if the relationship is linear, or a linear approximation is the best the data can do.
Random forests (RF) are an ensemble method based on decision trees. It can model complex relationships in the data amd usually provides good performance with relatively little tuning.
Artificial neural networks (NN) provide a universal approximator (i.e. allows infinitely complex modelling). They typically require significant domain-specific tuning, but may obtain very good results if optimal configurations are found.

The algorithms were picked because they are popular (which, in turn, they are because they work well). However, they were also picked because they provide decent probability estimates, rather than simple classification (e.g. 0-1 loss). As previously mentioned, pure classification is not very useful in the context of predicting draws, since almost all matches are designated as non-draws (This is why we discarded SVM's for instance, which provide a high-dimensional linear separator, but where probability output is less natural). Each of the chosen algorithms minimize a loss function which is probability based[10].

3.4.2. Regularization

The allowed complexity is not entirely determined by the choice of algorithm. Typically, we can alter the capacity of the algorithm so the effective capacity is less than the full representational capacity of the algorithm, which may help prevent overfitting (Goodfellow, Bengio, and Courville 2016, p. 110). A popular tool for doing so is *regularization*[11]. Classically, regularization alters the objective (loss) function of the algorithm with a penalty term[12]:

$$J^R(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) = J(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) + \alpha\Omega(\boldsymbol{w}) \tag{3.4}$$

Two classical approaches are *L1* regularization (adding the term $\Omega(\boldsymbol{w}) = \sum_i |w_i|$) and *L2* regularization (adding the term $\Omega(\boldsymbol{w}) = 0.5||\boldsymbol{w}||_2^2$) (ibid., pp. 224-230) Both are examples of *weight decay* which penalize larger weights, and hence encourage the algorithm not to fit too closely to the data. We define the types of regularization used below.[13]

3.4.3. Hyperparameter selection through grid search

Some algorithms provide further ways to tune the effective capacity. Generally, parameters which are set by the data scientist before estimation are called *hyperparameters*. We concern ourselves mainly with those which directly affect the effective capacity of the model by altering the optimal solution.[14] The choice of regularizer $\Omega(\boldsymbol{w})$ and the weights α above are ann example of such. The architecture of a neural network (number of hidden layers and nodes) are another. Deeper and larger nets can approximate more general functions.

[10]Logistic regression minimizes the negative log-likelihood, which corresponds exactly to minimizing the cross-entropy as in neural networks and random forest (Goodfellow, Bengio, and Courville 2016, p. 129).

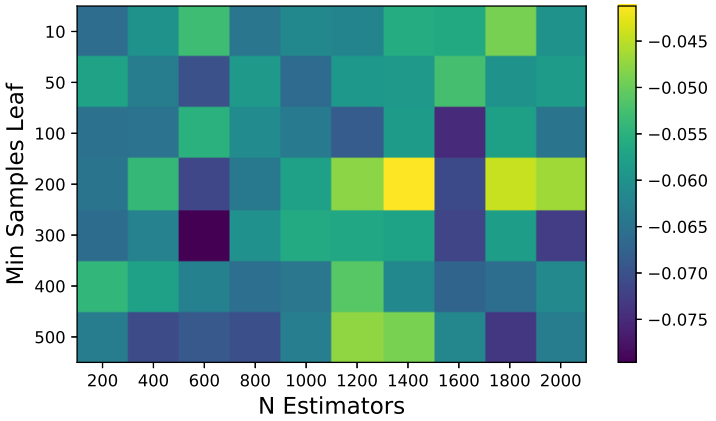
[11]For an indepth discussion of regularization in the context of neural networks (but relevant for other approaches) see ibid., pp. 221-309. For a general introduction, see (Abu-Mostafa, Magdon-Ismail, and Lin 2012a, pp. 126-137)

[12]Other approaches to regularization exist (e.g. *dropout* for NN's). Generally, regularization is any modification to an algorithm intended to reduce generalization error (Goodfellow, Bengio, and Courville 2016, p. 117).

[13]Random forest does not use regularization. Instead, it uses *out-of-bag* samples to combat overfitting.

[14]There are also hyperparameters which affects the optimizing of the estimator. These should only affect training times, but may also affect solutions, particularly if there are many similar local minima.

Figure 3.2: *Random forest grid search*



Note: The heat map only shows 2/3 hyperparameters, strictly due to visualization purposes. It's colored by our custom made performance measure: The expected turn from betting on all bets suggested by the algorithm.

Optimal hyper-parameters are typically domain-specific, and choosing them is hard. We choose a simple approach which works decently in practice: *Grid search* (Goodfellow, Bengio, and Courville 2016, p. 420). We define a range of possible values for each hyperparameter, and then train and validate the model using each combination of these. Hyperparameters were chosen to maximize the betting pay-off using our custom scorer. This is typically the combination with highest pay-off, unless the combination seems an outlier from other likely optima. There is no guarantee this provides the optimal solution, but it will typically yield an algorithm which is 'good enough' (and ideally, small changes should not affect betting choices much). The chosen hyperparameters can be seen in the next section. To see how it works, consider an example from random forests in figure 3.2. The yellow square marks the combination with best performance, which does not appear an outlier.

3.4.4. Estimated models

We ended up with six different specifications of the algorithms. Different versions again allows us to test various levels of model complexity. Table 3.1 summarizes the differences between the algorithms. All models were fitted using *scikit-learn* (Pedregosa et al. 2011). Details on the algorithms and their fitting can be found in appendix B.

Table 3.1: *Model descriptions*

Model	Description
Logit (I)	L1 regularized logistic regression with $1/\alpha = C \approx 1.2$ (55.0).
Logit (II)	Adds polynomial features. L2 (L1) regularized with $C \approx 18.3$ (0.14).
RF (I)	1600 (600) trees with $\sqrt{k}$ ($\log_2(k)$) features in each and at least 100 (500) leaves.
RF (II)	600 (1800) trees with $\sqrt{k}$ features in each and at least 200.
NN (I)	One hidden layer with 29 (32) nodes. L2-regularized with $\alpha = 0.0001$ (0.01).
NN (II)	Two hidden layers with 13/20 (13/40). L2-regularized with $\alpha = 0.001$ (0.01).

Parentheses indicate different hyperparameters chosen when modelling only matches in Premier League.

Table 4.1: *Profits from betting based on each algorithm*

	All	Logit (I)	Logit (II)	RF (I)	RF (II)	NN (I)	NN (II)
In-sample	-0.052	-0.056	0.052	0.305	0.135	0.119	0.117
Cross-validated	.	-0.092	-0.045	-0.058	-0.063	-0.059	-0.057
Out-of-sample	-0.042	-0.09	-0.051	-0.114	-0.086	-0.093	-0.076

Notes: Values indicate expected earnings from betting one euro spread across all bets suggested by the strategy. 'All' bets that all matches are draws. Algorithms bet if they set probability higher than implied by odds. In-sample bets on train data. Cross-validation is 10-fold on train data. Out-of-sample is on pristine test set.

4. Results: Can we make money betting on draws?

This section presents our main results - can we make money by betting on draws? The first section shows the results on the entire dataset. The second one investigates Premier League separately. Finally, we delve into how important each variable is to the algorithms.

4.1. Betting on all matches based on algorithms loses money

Table 4.1 presents the results from betting based on the various algorithms. Values indicate the expected return from betting spread out across all bets suggested by the algorithm. 'All' indicates a baseline result from betting on a draw in every match offered. If odds were fair, this should yield zero in expectation. The negative values thus indicate the bookies mark-up or fee: In the training set for instance, if you bet one euro you should expect to lose five cents. Achieving a better rate than this, even if you still lose money, would mean the model has more predictive power than bookies. However, you need positive numbers to actually make money.

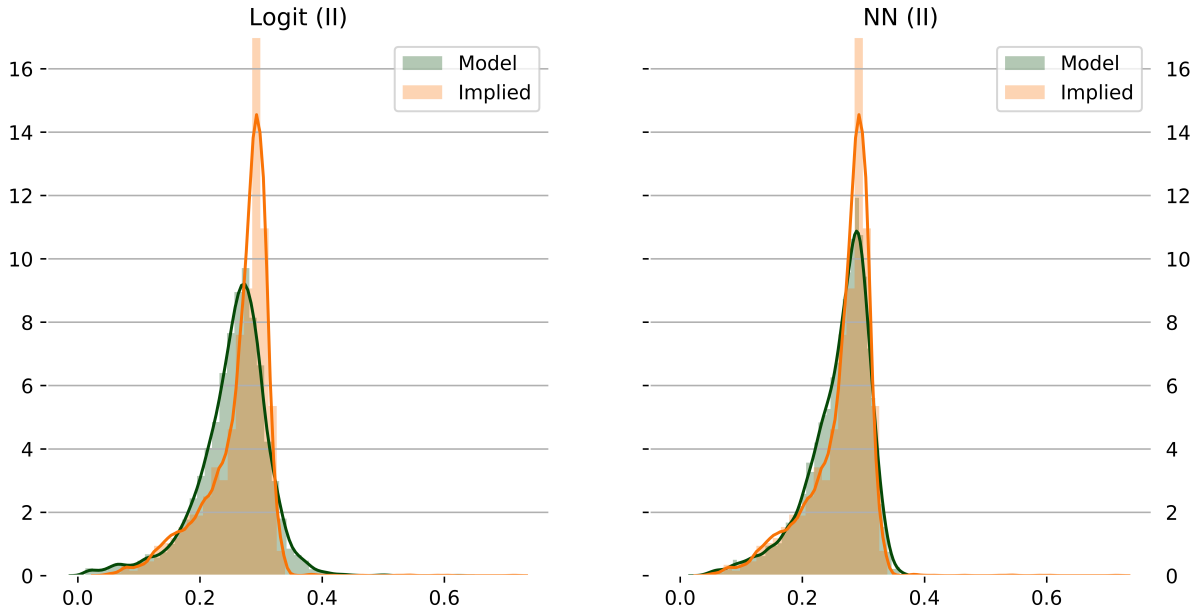
In-sample all models except the basic logistic regression are capable of making money. The random forests in particular suggest strong returns (one promises to return 30 pct. on the invested euro). However, as noted above, in-sample performance is upwardly biased due to overfitting. Hence we used cross-validation measures to choose the optimal hyperparameters, and these values are also indicated here. No algorithms expect to make money when properly measured. Most of them do however seem to perform about as well as the baseline.

The true test is out-of-sample performance on the test set. Since the training data is used during cross-validation (particularly as one runs more and more validation checks), these measures also become biased. The test set was kept entirely pristine, so it does represent a good performance check. Again, no models make money, and most actually perform quite worse than the baseline. This may partly be due to overfitting, as in-sample measures differ greatly from out-of-sample. However, given the varied algorithms and approaches tried, we believe that the algorithms do provide the best expectation given the data at hand and the included features. The problem is simply that the features, measured as they are, simply cannot predict draws well enough to make money.

4.1.1. Why do the algorithms lose money?

This leaves us with two possible conclusions. First, it may be that we do not have enough (or the correct) variables to estimate probabilities. If this is the case, a more complex approach

Figure 4.1: *Distribution of draw probabilities from algorithms and implied by odds*



might fare better. Alternatively, it may be that our hypothesized bias among bookies do not exist: There are no matches where the odds are greater than those implied by the underlying probabilities. In this case, we would have to compute better estimates than the bookies to make money. Indeed, they would have to be quite superior to conquer the bookies' fees.

It is difficult to distinguish between these cases, as we cannot know how the model would fare with more and/or better data. However, we can look at the probability distribution of draws, and try to discern whether there appears to be a possible bias in the odds of bookies. Figure 4.1 shows the out-of-sample distributions of our modelled probabilities and the implied probabilities. The key point to notice is that the probabilities are highly centered around the mean, which a thick left tail but hardly any probability mass right of the peak. This indicates that most matches have a similar chance of ending in a draw, with some matches exhibiting a lower probability, but few higher.

This could be consistent with the theorized bias. For matches with a greater probability, bookies realize that people are not willing to bet at the low odds this would entail. Hence the bookies set the odds higher 'folding' the right tail into the distribution. However, if this were the case our modelled probabilities should have thicker right tails than the implied probability. This is not what we observe. It may instead be that the true probabilities simply look like this, and there is no significant bookie bias. Consider that most of the theoretical features suggested that draws become less likely (many goals, mismatched teams), which could lead to a thicker left tail. Then, if most matches are well-matched, so that the probability is quite close to the average, $\mathrm{E}[y_i|\boldsymbol{x}] \approx \mathrm{E}[y_i] = 0.25$, we could observe a true distribution like this. However, do note that Logit (II) does show a slightly thicker right tail (which would entail matches the algorithm would bet on), and that this model performs best profit-wise. This could indicate that a better model might see positive profits. We explore one such possibility in the next section.

Table 4.2: *Profits from betting based on each algorithm (Premier League)*

	All	Logit (I)	Logit (II)	RF (I)	RF (II)	NN (I)	NN (II)
In-sample	-0.016	0.04	0.107	-0.115	0.072	0.186	0.304
Cross-validated	.	0.022	0.051	-0.111	-0.059	0.036	-0.015
Out-of-sample	-0.024	-0.151	-0.022	-0.1	-0.094	-0.061	-0.13

Notes: Values indicate expected earnings from betting one euro spread across all bets suggested by the strategy. 'All' bets that all matches are draws. Algorithms bet if they set probability higher than implied by odds. In-sample bets on train data. Cross-validation is 10-fold on train data. Out-of-sample is on pristine test set. Data is restricted to Premier League.Imputation, standardization, grid search was re-run, and models refitted.

4.2. But what about England?

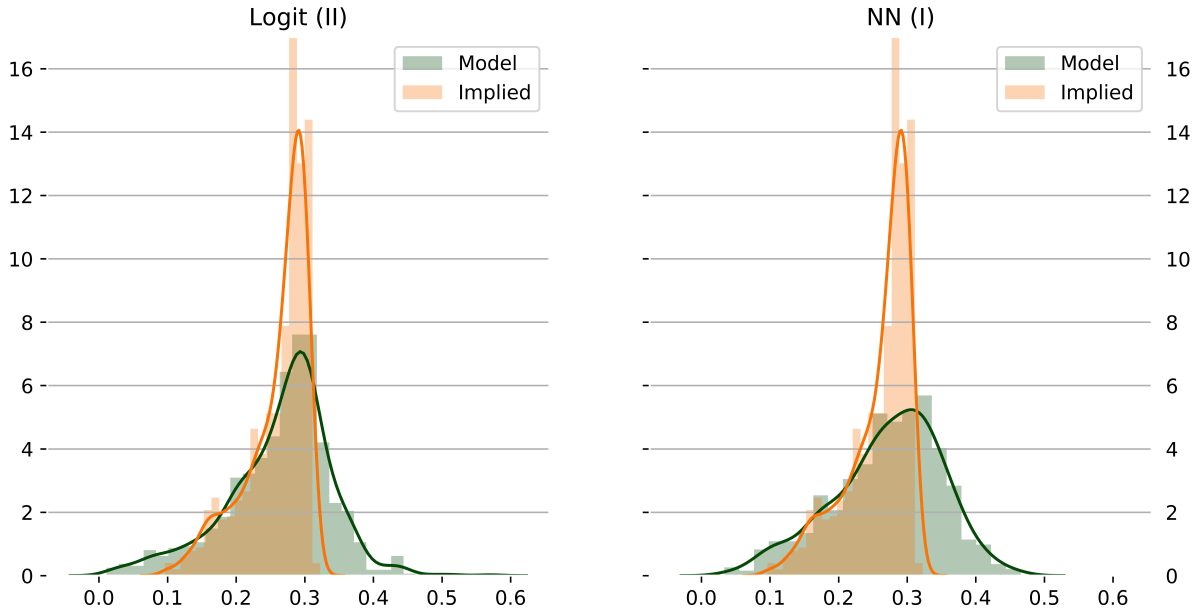
The findings of the previous section were somewhat disappointing. No model was able to earn a positive profit when evaluated out-of-sample. Again, this might indicate that draws simply are not undervalued by the bookmakers, and that there is no point in trying to predict the probabilities for draws more accurately than the bookmakers, as there is no indication of biased implied probabilities. However, instead of dealing with matches from a handful of European leagues, the mathematician writing for The Economist (2016) is only concerned with matches in the English Premier League. Indeed, the author points out that the results are not directly transferable to other leagues, more precisely, lower English leagues, as no evidence of a probability bias was found in the lower ranks of English football. To put our algorithms, and the article's hypothesis, to a final test, we therefore retrain and test the model on a data set containing only observations from the Premier League. The new results are shown in Table 4.2.

While the models seem to perform slightly better compared to the former case, we again end up losing money when evaluating the models' out-of-sample performance. As before, this might indicate that there is no point in trying to exploit a possible bias by predicting the draw probabilities more accurately than the bookmakers, as the implied probabilities already give a fair estimate of the likelihood. In the article, the author argues that especially the media coverage prior to football matches builds up a hype that results in betters opting for one side or the other instead of backing a draw. This might in particular be the case prior to "big" matches. Hence, a possible extension to the outlined modeling approach is to focus on "big" matches between well-matched teams.

4.2.1. Do we observe a bias?

The distribution of probabilities may offer some more direct evidence on whether English bookies tend to overreward draws. If this is the case, we should see fewer higher probabilities (corresponding to low odds), because bookies supposedly increase the odds. Implied probabilities can be seen in 4.2. Modelled probabilities does indeed have much thicker right tails than those implied by odds, which is consistent with the bias. However, the algorithms also have thicker left tails, which is not really consistent with the bias. An alternative explanation is simply that the algorithms are less certain, possibly due to low sample size when we only look at England (2,144 matches). The distributions are quite similar to normal distributions around the mean - It may simply be that given the small sample, the algorithms aren't certain enough that they

Figure 4.2: *Distribution of draw probabilities (Premier League)*



should tuck in the tails from their starting points.

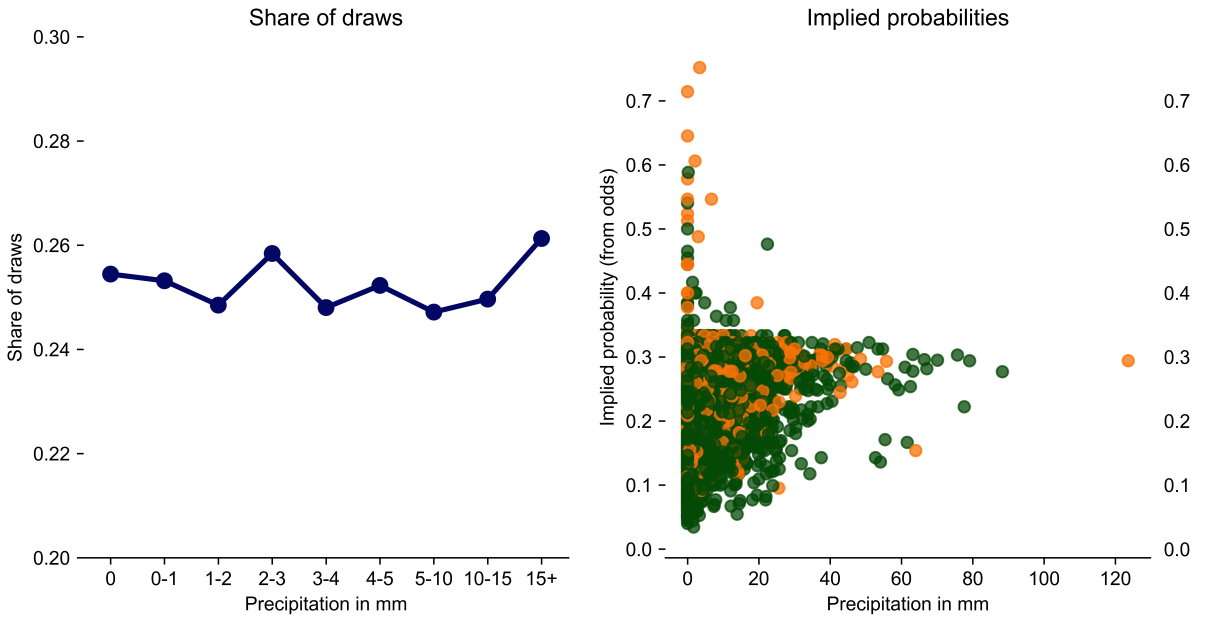
If the tails originate from bookie bias, then betting based on algorithms with thicker tails should make positive profits. Cross-validation does suggest this is the case for some algorithms, but the out-of-sample test does not. The discrepancy between cross-validation and the test set makes it hard to conclude on which case is true. The small sample size may play tricks with both estimates. However, the test case does present the best test of performance, which indicates that draws are simply improbable, rather than undervalued.

We conclude that the probability distribution of draws most likely have the form implied by odds, and that few matches are likely to end in draws, but that some matches are highly unlikely to do so. There appears to be no significant bias in odd-setting for bookies - certainly none that we have been able to take advantage of. This makes it hard to beat the bookies, because few matches differ in probabilities. Gathering more data would likely only allow us to match the baseline model, losing money corresponding the bookies mark-up.

4.3. Which features have predictive power?

Given that our model does seem to capture some facets of the probability distribution of draws, we may investigate which features drive this probability. Data scientists rarely focus on the formal hypothesis tests dear to the hearts of econometricians, so these are less well developed for the models we used. We could perform significance tests for the logit models. However, we instead take a less formal approach, and explore the results descriptively, and using a measure for feature importance for random forests.

Figure 4.3: *Match weather & outcome*



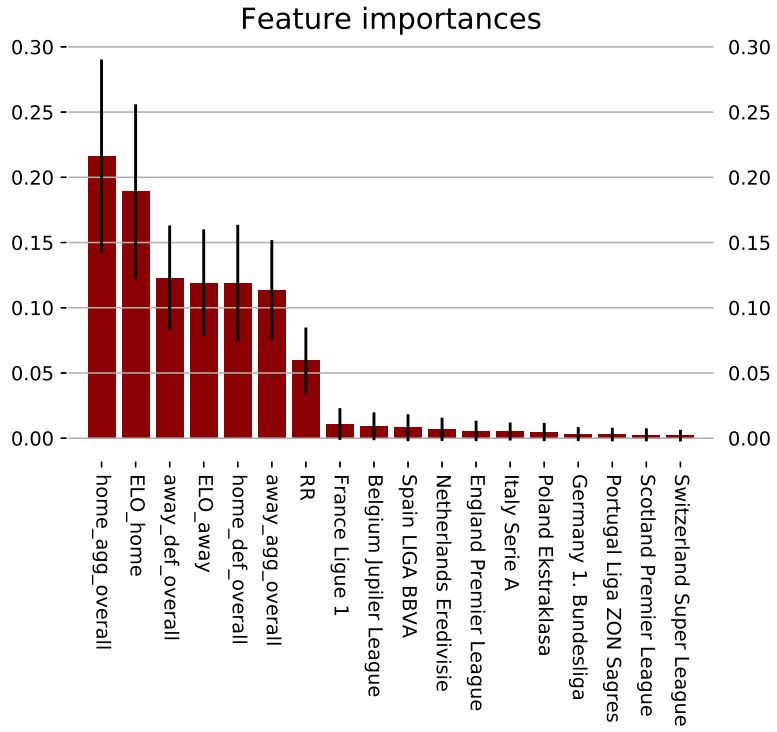
Notes: The left figure shows the share of draws among different subsets of precipitation. The right figure shows implied probabilities of matches across precipitation. Orange matches indicate draws.

4.3.1. Does rain increase the chance of draws?

A key novelty of our prediction model was to include the amount of rain in the area on the day of a match. This was incorporated with the notion that rain makes it more challenging to score goals, and hence increases the likelihood of draws. We can now attempt to conclude on whether this is the case. The left part of figure 4.3 presents an overall descriptive measure by showing the share of draws which occurred among matches with precipitation within certain levels. The share of draws is more or less stable irrespectively of the amount of rain which occurred - although there might appear a small increasing probability for the matches with more than 15 mm of rain, this might also be uncertainty as few matches see that much rain. Presented with only this graph there is no indication of rain having any predictive power on the outcome of a match.

However, we may investigate further. The analysis above found that implied probabilities actually appeared to be quite decent estimates of the true probability. The right hand side of figure 4.3 shows the distribution of such probabilities across levels of precipitation. There is no clear trend towards greater probabilities (or more draws) as levels of precipitation rise. While one could see a small trend at the beginning, it is likely only due to the larger share of matches with low precipitation. Thus far there is no evidence that the hypothesis hold true. However, we could take a more formal approach. Machine learning models are rarely oriented towards formal hypothesis tests, but they do still have various measures of importance, one of which we explore in the next section.

Figure 4.4: *Feature Importance in Random Forest*



Note: The black bars shows standard error of the feature importance between the trees in forest.

4.3.2. Measuring feature importance

We have chosen to focus on one such measure within our algorithms, and now broaden our search to look at all features, not merely weather. The random forest algorithm provides a way of measuring the improvement each input variable has on the algorithm's performance: Feature importance. Simply speaking, feature importance measures the number of times a given variable is used for dividing the observations into draws and non-draws. It should be noted that the measure tends to prefer continuous variables over dummy variables, as the former provide more options for partitioning the observations. Unsurprisingly, figure 4.4 shows league dummies as being less important.

Overall, it seems that metrics for the home team's aggressiveness, `home_agg_overall`, and its advantage over the away team, `ELO_home`, are the variables that exhibit the highest degree of degree of informativeness when predicting draws. Moreover, we note that the performance measures but that these are only significantly different from the rain variable and the league dummies. Furthermore, we note that the different performance measures contribute significantly more than the precipitation variable, `RR`. That being said, the amount of precipitation still comes with some degree of predictive power. However, the reader should keep in mind that the results section showed a simple betting strategy led to a better outcome than our models, so the actual importance of each variable should be viewed in this light.

5. Conclusion: No clear evidence of bias in bookie odds

We set out to beat bookies at their own game but were beaten to the draw. We had hoped to make money by exploiting the hypothesis that since regular people prefer to see a win in one direction or the other, bookies increase the odds of draws to draw betters and minimize their own risk (Economist 2016). We used a machine learning approach to model probabilities, and placed fictional bets on matches if the probability of a draw was modeled as larger than those implied by odds. We looked at the profits of fictional bets across about 25,000 matches from 2008-2016 in 11 European leagues. We also considered results specifically for about 2,000 matches in Premier League, because the article suggested the bias might be stronger there. We found that all approaches were unable to earn positive profits in expectation.

We investigated the probability distributions implied both by odds and our models to check whether they were consistent with biased odds. They show a peculiar probability distribution: Probability mass is highly centered in a peak, with a thick left tail and no right tail. We initially believed this was due to the right tail being 'folded' into the middle, because people were unwilling to place bets at high probabilities (i.e. low odds). However, it may instead be that the probability of draws hover around 25 pct. in most matches, except a few where mismatched teams or high goal propensities means that teams rarely end up at the same number of goals. Both the implied odds and our models using all data clearly exhibit such a pattern. If this was due to a bias, we would expect the models to show a thicker right tail, which does not occur.

We believe that the paper estimates profits well given the parsimonious statistical model defined, and the quality of data obtained. However, it is likely that that a more complex statistical model combined with more and/or better data might yield more precise probability estimates. We cannot conclusively state whether such model would exhibit thicker right tails, or be able to earn positive profits. However, one can note that our model is able to replicate other key features of the implied probability distributions (thick left tail, steep peak). Since the odds of bookies can generally be expected to be quite decent except for any biases, this indicates that our model does capture the underlying relationship. This does not suggest odds are biased.

Despite this, we cannot conclusively state whether a bias exists in the odds for draws. What we can state is that if it exists, it is not as easy to exploit as suggested in the article. We certainly cannot recommend starting to bet at draws indiscriminately. Indeed, it is our belief that using more complex models to match bookie precision would move results closer to the baseline, but not to make positive profits. In order to do so, it would be necessary to make models which actually beats the bookies. However, given the uncertainly inherent in any statistical analysis of this type, it might be relevant to check. Our exploration of feature importance suggest that such models should include measures of goal propensity and team skill, but should probably not extend a lot of effort gathering weather data.

6. References

Abu-Mostafa, Yaser S, Malik Magdon-Ismail, and Hsuan-Tien Lin (2012a). *Learning from data*. Vol. 4. AMLBook New York, NY, USA:

— (2012b). 'Learning from data'. In: vol. 4. AMLBook New York, NY, USA: chap. e-Chapter 7: 'Neural networks'.

Angrist, Joshua D and Jörn-Steffen Pischke (2008). *Mostly harmless econometrics: An empiricist's companion*. Princeton university press.

Cameron, A.C. and P.K. Trivedi (2005). *Microeconometrics: Methods and Applications*. Cambridge University Press. ISBN: 978-0-521-84805-3.

Cybenko, George (1989). 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Economist, The (2016). 'How I used maths to beat the bookies'. In: *1843 Magazine*. URL: `https://www.1843magazine.com/features/the-daily/how-i-used-maths-to-beat-the-bookies`.

Funahashi, Ken-Ichi (1989). 'On the approximate realization of continuous mappings by neural networks'. In: *Neural networks* 2.3, pp. 183–192.

Goodfellow, I., I Bengio, and A. Courville (2016). *Deep Learning*. MIT Press.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2001). *The Elements of Statistical Learning*. Springer.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). 'Multilayer feedforward networks are universal approximators'. In: *Neural networks* 2.5, pp. 359–366.

James, Gareth et al. (2014). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated. ISBN: 1461471370, 9781461471370.

Pedregosa, F. et al. (2011). 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

A. Code & data files (available at GitHub)

Main source files are available at the GitHub repository https://github.com/rbjoern/BookieDraws. The code work for the paper was done in Python, and is available as Jupyter Notebooks. Top level datafiles are available in as csv-files.

Data

The following raw data files were used to construct the main dataset. They are not available on GitHub due to size restrictions, but can be obtained by contacting the authors on the site.

Kaggle European Soccer Database. SQlite database downloaded from *https://www.kaggle.com/hugomathien/soccer* and extracted to csv-files.

Elo ratings Elo-ratings scraped from *clubelo.com* using *ScrapeEloRatings.ipynb*.

European Climate Assesment weather grids. 0.25 degree grids for Europe from 1995-2010 and 2010-2017. Downloaded from *https://www.ecad.eu/* as NetCDF files.

Stadium coordinates. Loaded directly from *WikiData.org* in *GenerateMainData.ipynb*.

The data is combined to form the main dataset *main_df.csv*, and split into a test and train dataset (*data_train.csv, data_test.csv*). The data work is done using the following code files:

GenerateMainData.ipynb. Loads and combines the files above to form the main dataset. It also performs pre-modelling data processing (e.g. computes average goal variables).

SplitMainData.ipynb. Splits the dataset into a test and train dataset.

ScrapeEloRatings.ipynb. Scraper to extract ratings from *clubelo.com*

Modelling

The modelling pipeline from figure 3.1 is run through these files. All files are available for the entire dataset, and only for the Premier league (denoted by *_premier*).

Grid_Search Files are available for all three algorithms. Each of them performs the pre-processing data steps, and then searches for optimal hyper-parameters over a relevant grid.

Modelling_Pipeline Performs data pre-processing, and then estimates optimal models based on grid search. It then evaluates these both in-sample and out-of-sample.

The main result tables (table 4.1 and 4.2) are produced in the pipeline file.

Figures

The figures are produced as follows (remaining figures are TikZ from LaTeX):

Figure 2.2 (weather & stadium maps) is produced in *WeatherMap.ipynb*.

Figure 3.2 (grid search heat map) is produced in *Grid_search_RF*.

Figure 4.1 (probability distribution) is produced in *Modelling_Pipeline.ipynb*.

Figure 4.2 (probability distribution) is produced in *Modelling_Pipeline_Premier.ipynb*.

Figure 4.4 (feature importance) is produced in *Modelling_Pipeline.ipynb*.

Figure 4.3 (weather & outcome) is produced in *Modelling_Pipeline.ipynb*.

Figure C.1 (probability distribution for all) is produced in *Modelling_Pipeline.ipynb*.

Figure C.2 (probability distribution for all) is produced in *Modelling_Pipeline_Premier.ipynb*.

B. Algorithm descriptions

B.1. Logistic regression

Here we consider the logistic regression model. Logistic regression models the probability that the target y_i belongs to a particular (binary) category. In our case, the target has value one, if match i ended in a draw, and zero otherwise. If we let $p_i(x_i)$ denote the conditional probability $P(y_i = 1|x_i)$, the probability implied by the logistic regression model is given by

$$p_i(x_i) = \frac{\exp\{c + x_i\beta\}}{1 + \exp\{c + x_i\beta\}}, \tag{B.1}$$

where c is an intercept, x_i is a $1 \times d$ row vector containing the independent variables and β is a $d \times 1$ column vector containing regression parameters to be estimated (James et al. 2014).

B.1.1. Estimation of the model

In order to fit the logistic regression model to the observed data, the implementation underlying scikit-learn solves the following optimization problem

$$\min_{\beta, c} P(\beta, c) - C \log L(y, X, \beta, c), \tag{B.2}$$

where $\log L(y, X, \beta, c)$ is the log-likelihood function given by

$$\log L(y, X, \beta, c) = -\sum_{i=1}^{n} \log\left(1 + \exp\{-y_i'(c + x_i\beta)\}\right), \tag{B.3}$$

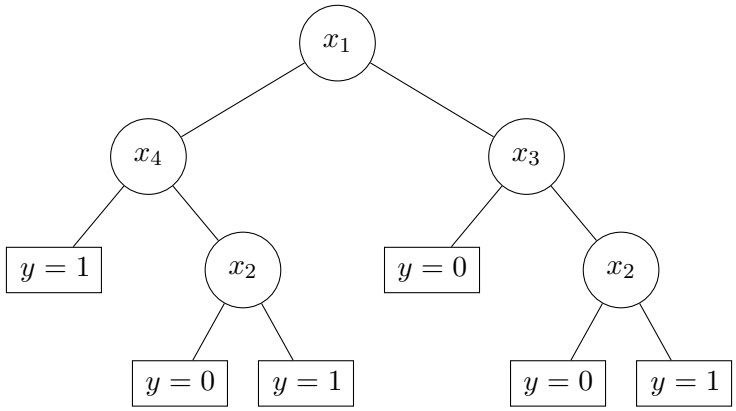
with $y_i' = 1$ if $y_i = 1$, and $y_i' = -1$ otherwise. $P(\beta, c)$ is a penalty function, or regularization term. In scikit-learn, the estimation can be performed using the two classical regularization approaches described in the main test. That is, the regularization comes either as $L1$ or $L2$,

$$(L1) \qquad P(\beta, c) = ||\beta|| = \sum_{i=1}^{d} |\beta_i|, \qquad (L2) \qquad P(\beta, c) = \beta^T\beta = \sum_{i=1}^{d} \beta_i^2. \tag{B.4}$$

In either case, the regularization term penalizes the complexity of the model by imposes a penalty on large parameter values to prevent the model from overfitting. The parameter C is an inverse measure of the regularization strength. The lower the value of C, the more weight is attributed to the regularization term, while less weight is imposed on maximizing the likelihood-function, i.e. fitting the data. A low value of C will hence yield models with relatively low complexity.

We estimated two logistic models. One using the metrics and dummy variables described in the main text, and one model using where we also included polynomial features, i.e. quadratic terms and interaction terms. The former resulted in 18 independent variables, wheres the latter resulted in a total of 190 variables. Both models were estimated using grid search with a 10-fold cross-validation. The grid search was performed over the two penalty functions, $L1$ and $L2$, and the regularization strength C. For the parameter C, the grid search was performed over a set of 12 exponentially increasing values given by $\exp\{-2 + i6/11\}$, for $i = 0, 1, \ldots, 11$. All in all 24 different combinations.

Figure B.1: *A simple Classification tree*



B.2. Random forests

One of the most commonly used machine learning techniques is the Random forest. The algorithm builds on the idea of classification/regression trees. In the context of our objective, the goal of each classification tree is to classify draws, i.e. y_i is discrete. We can describe the data generating process of a match ending in a draw with a general formulation $y_i = g_0(x_i) + \varepsilon_i$. The goal of a classification tree is then to estimate $g_0(x_i)$ without imposing a structure. A tree works as a piece-wise constant function, where the data points are grouped in kinks at each node, until there is little classification improvement left or a specific stopping criterion is met, see figure B.1.

A random forest applies aggregated bootstraping, also known as bagging, to an abundance of trees, then obtains a vote form each tree, and classifies the outcome by majority vote. That is, it take a random sample with replacement B times (the same idea as K-fold cross-validation), and predict the outcome with informations from all of the B trees. The procedure is described in algorithm 1.

Random forest works well if the goal is a prediction problem and is in a high dimensional environment, e.g. if we want to use clubs as an identifier in our model. We say that the random forest is a greedy algorithm because it always takes the seemingly best split, and ignores potential other interactions.

B.2.1. Hyperparameters in Random forest

Algorithm 1: Random forest

Input: $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, number of trees B, number of variables m

Output: Trees $T_1, \ldots, T_B$

for *b=1 to B* **do**

- Draw bootstrap sample S' of size N with replacement from S
- **if** *node size* $> n_{min}$ **then**
  - Select p variables at random from the m variables.
  - Pick the best split-point among the p variables.
  - Split the node into two daughter nodes.
- **Return:** T_b

In general, Random forest has gained popularity because it requires little tuning, but the algorithm still benefits from tuning hyperparameters. To ensure our model isn't overfitting the data, we consider whether the include the following hyperparameters in our grid search.

Maximum depth: The default value of *None* is chosen, since one should rarely restrict the forest unnecessarily. This means each tree is allowed as many partitions as it finds optimal on a given branch. But, as described below, the total number of features is still restricted.

Criterion function: A downside of random forest is that the predictions tend to underestimate the probability of extreme values, i.e. probabilities close to zero and one. One way to combat this problem is to use entropy as a minimization measure instead of the Gini impurity, which is more focused on minimizing misclassification. Though it should be noted, that there is little difference between the two measures.

Bootstrap: Is either True or False, indicating whether we want to draw our random samples with or without replacement. Since one of the benefits of bagging is sampling with replacement, this parameters is set to True.

Minimum sample splits: A general recommendation is to set this parameter to two in a classification problem (Hastie, Tibshirani, and Friedman 2001, p. 590), and we follow this procedure. This leaves us with three parameters to tune in our grid search: maximum number of features, number of estimators, and minimum sample leaves, see figure 3.2. As a general rule, these should always be tuned[15].

Maximum number of features: Refers to the maximum number of splits each tree can use as features. Increasing the number of features will in general increase the performance of the algorithm since it can choose more freely between the input variables, on the other hand this also lowers the individuality of each tree, which is the strength of random forest. Using grid search we find that this is $\sqrt{n\_features}$

Number of estimators: Describes the number of trees in our forest and performance is increasing with the number of trees. It should be noted that the algorithm seems to stabilize around 200 (ibid., p. 589), so including between 200 to 2000 trees should be enough to get a good performance. Using grid search, the random forest consists of $B = 1600$ trees

Minimum sample leaves: Indicates how few end notes the tree must have before stopping. A small number of leaves increasing the chance of *fitting the noise*, increasing the number of leaves reduces overfitting. The grid search sets $min\_sample\_leaf = 100$.

The random forests were estimated using the Python module *scikit-learn* (Pedregosa et al. 2011).

[15] https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

B.3. Artificial neural networks

This section provides a short summary of what artificial neural networks are, how we implemented one in the paper. For a pedagogic introduction to networks see Abu-Mostafa, Magdon-Ismail, and Lin (2012b). For a more advanced treatment see Goodfellow, Bengio, and Courville (2016).

B.3.1. Neural networks are universal approximators

Neural networks are often introduced with reference to their initial inspiration from the neurons of the brain. However, modern neural networks have little to do with neuroscience. Instead, 'it is best to think of feedforward networks as function approximation machines that are designed to achieve statistical generalization' (ibid., p. 164). Basically, neural networks are a complex learning algorithm, which can approximate any target function[16].

A basic neural network with one hidden layer with J nodes is shown in figure B.2. The first layer inputs K features to each of the nodes in the hidden layers. These then compute a linear signal $s_j = \sum_{i=0}^{K} \beta_{j,i}^{(1)} x_i$ (note that the weights are different for each node), and applies an *activation function* $h_j = \theta(s_j)$. The output layer then calculates a linear signal from these an applies an output function $\sigma(\cdot)$ to calculate the algorithm output $f(\boldsymbol{x}) = \sigma\left(sum_{j=0}^{J} \beta_j^{(2)} h_i\right)$. Note that if we remove the hidden layer and let σ be the sigmoid function, this is exactly logistic regression. The easiest way to think of neural networks are therefore as combinations of units which work (sort of) like regressions. Combining them is what increases representational capacity. Formally, we can combine the entire process as

$$\begin{aligned} \hat{f}(\boldsymbol{x}) &= \sigma\left(\beta_0^{(2)} + \sum_{j=1}^{J^{(1)}} \beta_j^{(2)} \eta\left(\beta_0^{(1)} + \sum_{k=1}^{K} \beta_{k,j}^{(1)} x_k\right)\right) \\ &= \sigma\left(\tilde{\eta}\left(\tilde{\boldsymbol{x}}\beta^{(1)}\right)\beta^{(2)}\right) \end{aligned} \tag{B.5}$$

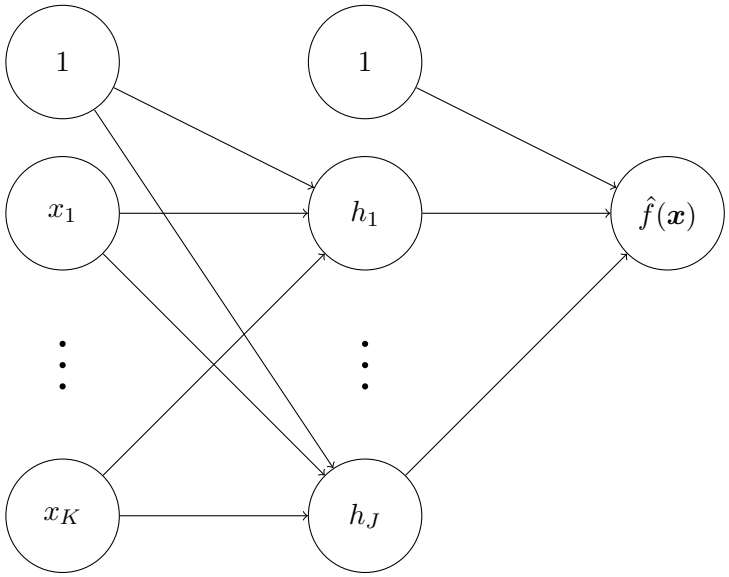
One-layer networks are theoretically sufficient to approximate any function if enough nodes are provided. Adding more layers may allow one to approximate it with much smaller networks in practice though(Abu-Mostafa, Magdon-Ismail, and Lin 2012b). We therefore implement both a one and two layer network. Adding multiple layers do not alter the basic functionality - we just combine more regression-like objects. It can be written as (where L denotes the number of layers).

$$\begin{aligned} f(x) &= \sigma\left(\theta\left(\ldots\theta\left(\theta\left(x\beta^{(1)}\right)\beta^{(2)}\right)\ldots\right)\beta^{(L)}\right) \\ &= \sigma\left(\beta_0^{(L)} + \sum_{h=1}^{J^{(L-1)}} \beta_h^{(L)} \theta\left(\ldots\theta\left(\beta_0^{(2)} + \sum_{j=1}^{J^{(1)}} \beta_j^{(2)} \theta\left(\beta_0^{(1)} + \sum_{k=1}^{K} \beta_{j,k}^{(1)} x_k\right)\right)\right)\right) \end{aligned} \tag{B.6}$$

The notation largely follows that of (ibid.). However, the formalities are not that relevant. The main thing to remember is that neural networks are sort of like regressions, except they can provide arbitrarily complex function, and are much harder to find the optimal solution for. Fortunately, there is a large community concerned with doing just that.

[16]The universal approximation capabilities were first proved by (Hornik, Stinchcombe, and White 1989), (Cybenko 1989) and (Funahashi 1989), and later extended to modern versions of neural networks (Goodfellow, Bengio, and Courville 2016, p. 192).

Figure B.2: *A basic feedforward neural network with one hidden layer.*



B.3.2. Specification of the network

There are several types of neural networks. We use the most basic one: Feed-forward networks (or multilayer perceptrons). Even within this, there are several choices to make which define the network. Each is noted here, with a short reasoning.

Output function: Sigmoid. We use the sigmoid (logistic) function to squash the output to $[0, 1]$ which provide a probabilistic interpretation.

Activation function: ReLU. Rectified linear units imply that $\theta(z) = \max\{0, z\}$. They are easy to optimize, still allow non-linear approximation, and work very well in practice (Goodfellow, Bengio, and Courville 2016, p. 187).

Cost function: Entropy (maximum likelihood). This is the default way to train neural networks, and provides probabilistic modelling.

No early stopping. Early stopping implies watching validation error, and stopping when this begins to increase rather than after loss convergence (Abu-Mostafa, Magdon-Ismail, and Lin 2012b, pp. 24-27). However, as noted, the score is not meaningful for draws, so we focus on probabilistic convergence instead.

Optimization: Adam. Neural networks are optimized using *backpropagation* (Goodfellow, Bengio, and Courville 2016, pp. 197-214). Particularly, we use the Adam (adaptive moments) algorithm, which is robust to choices of hyperparameters, and works well in practice with medium-size datasets (ibid., p. 301).

Architecture. The number of hidden units was chosen using cross-validated grid search, as described in the paper. An algorithm was suggested with one and two layers respectively. The grid search suggested that both were kept relatively small, with 29 nodes in the single hidden layer, and (13,20) nodes in the two-layer version.

Regularization. The networks used $L2$-regularization. The weights were set using grid search, an arrived at $\alpha = 10^-4$ and $\alpha = 10^-3$ for one and two layers respectively.

The networks were estimated using the Python module *scikit-learn* (Pedregosa et al. 2011).

C. Supplementary figures

Figure C.1: *Distribution of draw probabilities from algorithms and implied by odds*

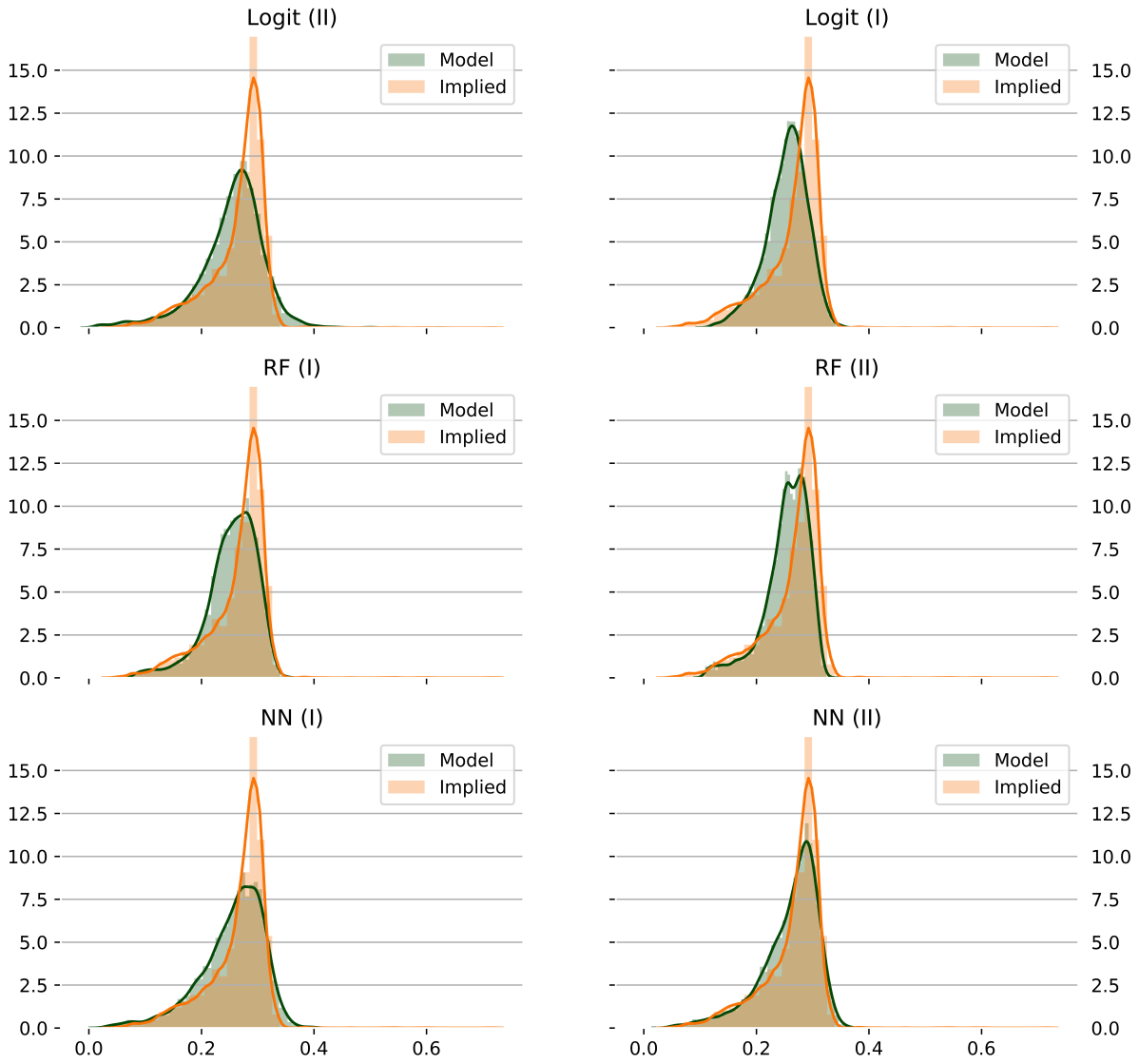


Figure C.2: *Distribution of draw probabilities from algorithms and implied by odds (Premier League)*