Machine Learning II - Final Project Report

Omer Madmon, Omer Nahum, Eilam Shapira

Overview

The English Premier League is the top level of the English football league system. Contested by 20 clubs, it operates on a system of promotion and relegation with the English Football League (the second level of the English football league system). Seasons run from August to May with each team playing 38 matches (playing all 19 other teams both home and away). Each team earns 3 points for a win, 1 point for a draw and 0 points for a loss. The final season league table is induced by the cumulative number of points for each team.

The Premier League has the highest revenue of any association football league in the world, and in this project we will tackle the task of predicting the final league table. This task is highly important for various organizations, including football clubs, media channels, gambling agencies, etc.

Task Definition

Our task is to predict the final English Premier League table based on data regarding the contested teams squads and season matches (as will be described in the Datasets section below). To deal with this task we will introduce two high level approaches:

First Approach

In this approach we will directly predict the number of points each team earns by the end of the season based on the squad data, and then sort the teams by the predicted number of points. This reduces the ranking problem into a regression problem. Formally:

Denote $x_{i,s} \in R^d$ as feature vector representing the squads of team i in season s.
Denote $X_s \in R^{20 \times d}$ as matrix of $x_{1,s}, \ldots x_{20,s}$ (all squad vectors of teams in season s).
Denote $Y_s \in R^{20}$ as the vector containing the number of points acquired by each teams in season s.
Assuming the train set contains seasons $s_1, \ldots s_k$, denote $X \in R^{(20k) \times d}$, Y^{20k} as the vertical concatenation of $X_{s_1}, \ldots X_{s_k}$ and $Y_{s_1}, \ldots Y_{s_k}$ respectively.

Then our first approach solves the Premier League ranking problem as a regression problem on the explained variable Y based on the explaining variables X.

The features contained in the vector representation of each team squad are explained in more detail in the 'Basic Part/Datasets' section.

Second Approach

In this approach we will break down the ranking task into smaller tasks of predicting the probabilities of win/draw/lose for every match in the season. That is, for every ordered pair of teams (home_team, away_team) we will predict the probability of each match outcome.

After probabilities prediction for each match of the season, we can aggregate the match outcome into a final season table by either computing the expected number of points for each team (will be referred to as 'Expectation') or simulating all season matches (will be referred to as 'Simulation').

Probabilities Learning Subtask

Denote $x_{(i,j),s} \in R^{2d}$ as the vector representing teams i and j in season s. This is basically a horizontal concatenation of x_i and x_j as defined in the first approach.
Denote $y_{(i,j),s} \in \{HomeWins,\ Draw,\ HomeLoses\}$ as the outcome of the match between team i (as home team) and team j (as away team) in season s.
Denote $X_s \in R^{38 \cdot 2d}$ and $Y_s \in \{HomeWins,\ Draw,\ HomeLoses\}^{38}$ as the matrix and vector containing all the data regarding season s.

Given a train set containing observations from multiple seasons $s_1, \ldots s_k$, in the prediction learnings subtask is to use a learning algorithm that predicts for each (ordered) pair of teams (i, j) the probabilities of the possible outcomes of the match between teams i (as home team) and team j (as away team). Then, given a test set containing observations from a new season, the learned parameters are being used to predict the probabilities for the new matches outcomes.

In the basic part we will use logistic regression and a neural network to solve the probabilities learning subtask. Then, in the advanced part we will introduce dependencies between consecutive matches' outcomes to solve the probabilities learning subtask.

Ranking Task Given Learned Probabilities

As described above, we will present two ranking methods for solving the Premier League ranking problem given the learned probabilities.

Expectation: for each team i, the expected number of points PTS_i is given by

$$PTS_i = \Sigma_{(j \neq i)} [3P(home\ wins \mid (i,j)) + 1P(draw \mid (i,j)) + 0P(home\ loses \mid (i,j)) + \\ + 3P(home\ loses \mid (j,i)) + 1P(draw \mid (j,i)) + 0P(home\ wins \mid (j,i))]$$

Simulation: for each season match sample an outcome from the predicted probability distribution, and then for each team sum all points to create the final league table. This process of simulating the season matches outcomes can also be repeated multiple times, and the predicted number of points for each team would be the average over all simulation. Notice that as the number of simulations increases, the predicted table converges to the predicted table of the expectation ranking method.

In the basic part, each method of the second approach will be tested with both expectation and simulation.

Basic Part

In the basic part we will introduce a single algorithm for the first approach and two algorithms for the second approach (each one will be using both expectation and simulation to induce ranking). We will treat the single algorithm from the first approach and one algorithm from the second approach as 'baseline' methods, and then design a neural network based algorithm for the second approach.

Datasets

We will be using data from multiple sources:

1. FIFA Complete Dataset (2015-2021)
    - The FIFA Dataset contains players data from multiple leagues, teams and seasons.
    - Each player is represented by a set of numerical features (sprint speed, tackling, finishing, etc.) and categorical features (team, nationality, right/left footed, special skills, etc.)
    - The FIFA dataset was used to represent team squads as will be described in the 'Basic Part/Data Manipulation and Exploration' section.
    - This dataset will be used for both approaches.
2. Results of English Premier League Matches (2015-2021)
    - Data for 2018/2019-2020/2021 was collected manually.
    - Each match is represented by date, home team, away team, match result.
    - This dataset will be used only for the second approach.
3. Final Tables of Previous Seasons (2015-2021)
    - Data for 2017/2018-2020/2021 was collected manually.
    - Each table contains the team name, final table position and number of points.
    - This dataset will be used for both approaches.

Data Manipulation and Exploration

Player Representation

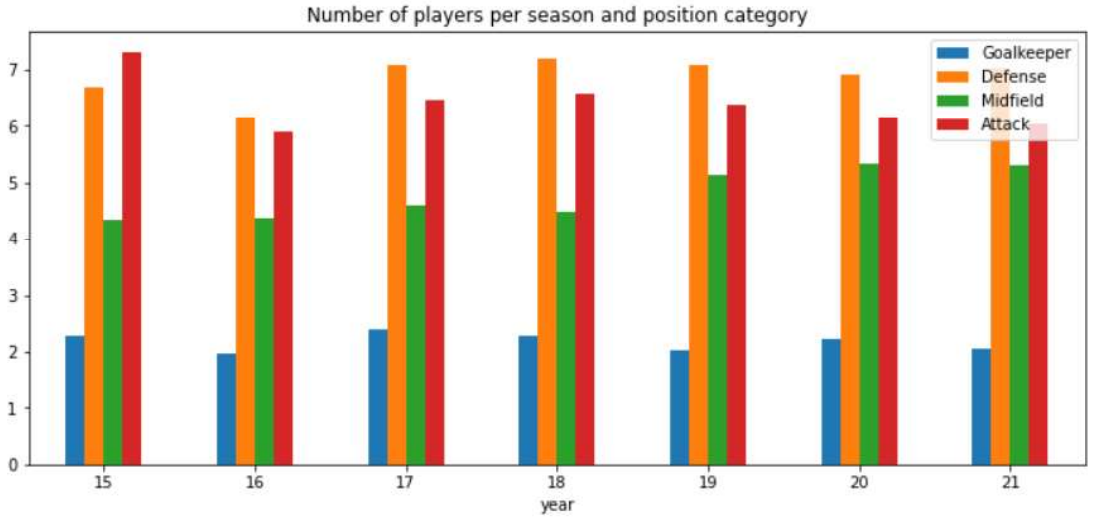
In the FIFA dataset, each player is represented by over 100 features, most are numerical features in the range 0-99, representing skills score and position score, and others categorical features such as nationality, strong foot, and features representing special skills of the players (of instance: leadership, free kicks taker, etc).

In order to represent a player in a numerical vector space we first transformed each categorical variable to multiple dummy binary variables. Then, after exploring the data, we found out that some goalkeeper features are irrelevant or missing for non-goalkeepers players (and vice-versa). Therefore we have decided on the relevant features for goalkeepers and non-goalkeepers players. For instance, 'reflexing' is a goalkeeping feature which isn't relevant for non-goalkeepers. That way we end up representing these two types of players in different dimensions.

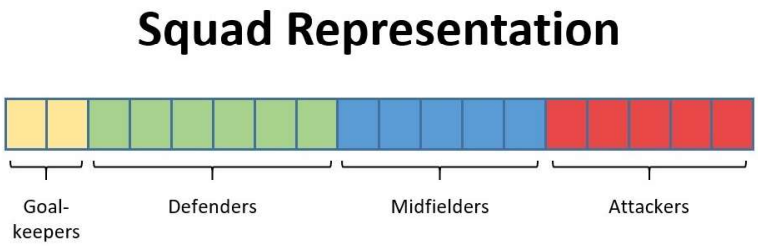
Squad Representation

We have decided to represent each team by the top K players in its squad. In particular, each player is represented by a high dimensional vector of features, including skills scores, positions scores, dummy variables representing additional features (such as: personality and special skills). Then, a team squad is represented as a concatenation of players (top k_0 goalkeepers, top k_1 defenders, top k_2 midfielders, top k_3 attackers, where $K = k_0 + k_1 + k_2 + k_3$). Within each category, all players are sorted from best to least (by players' overall score).

In order to determine the values of each k_i, we have first labeled each player into one of the following labels: goalkeeper, defender, midfielder and attacker, based on their scores for the more specific positions in the field. Then, we have visualized the distribution of the players labels in the data:



As we can see from the figure, most field players are labeled as either defense or attack and only few players are considered as midfielders. We have decided to fix $k_0 = 2,\ k_1 = 6,\ k_2 = 5,\ k_3 = 5$ and whenever a squad has less than 5 midfielders fill in from defenders or attackers.



Note that in this schematic figure, each block is a player representation vector itself.

Methods

First approach baseline - Linear Regression

We solved the regression problem using linear regression.

Second approach baseline - Logistic Regression

Match outcome probabilities learned by multiclass logistic regression model. Then, ranking is determined by either expectation or simulation.

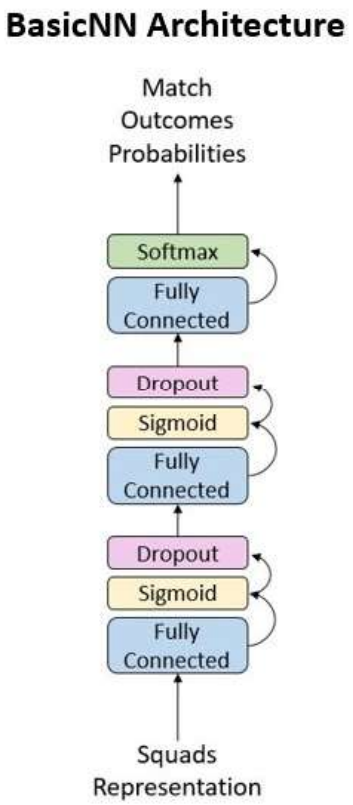
Second approach - Deep Model

Match outcome probabilities learned by a deep neural network. Then, ranking is determined by either expectation or simulation.

Architecture

In the basic part we will be using a fully connected architecture consisting of three layers as described in the 'BasicNN Architecture' figure.

Activation functions between all layers are sigmoid, and the last layer activation function is softmax (in order to induce a valid distribution measure over match outcomes). We have used dropout layers between each two layers to create regularization. The neural network is trained with respect to the categorical cross entropy loss function.



Hyper Parameters Tuning

We chose to focus on tuning the following hyper parameters: learning rate, batch size, number of epochs, number of layers, dropout probability, activation functions and dimension of hidden layers. We have performed a grid search as follows:

1. For each combination of hyper parameter values, perform leave-one-out cross validation (each season data is test set once).
2. Choose the top-3 models achieving the highest ordinal accuracy score on average (see 'Evaluation Metrics/Additional Metrics' section for the definition of ordinal accuracy).
3. Finally we have chosen the model achieving the highest score with respect to other ranking metrics on average (see 'Evaluation Metrics/Ranking Evaluation Metrics' section for the metrics definitions).

Evaluation Metrics

To evaluate the methods described above we have defined a few metrics that fit the specific task of predicting the Premier League final table. First, we have defined the following ranking groups:

1. First place wins the championship.
2. Places 2-4 qualify for the European Champions League.
3. Places 5-6 qualify to the UEFA European League.
4. Places 7-17 do not qualify to any competition nor are relegated from the premier league.
5. Places 18-20 relegated to the second league.

It is obvious that flipping the true 7th and 8th place is not as bad as flipping the true first and second place when predicting the final league table. Some mistakes have crucial consequences while others are less important, and we expect our evaluation metrics to reflect this property of the ranking problem. Therefore, our ranking evaluation metrics will punish a miss of the ranking group more heavily compared to a miss of only the exact position within the correct ranking group.

Adjusted Ranks

To reflect the idea described above we have first defined a new rank for each position in the predicted table. The idea is that the difference between consecutive positions within the same rank group is smaller than the difference between consecutive positions belonging to different ranking groups. The adjusted ranks have been determined by the following rules:

1. The difference between each two consecutive positions from different ranking groups is always 1.
2. The differences between each two consecutive positions within the same ranking group is always uniform.

Then, the adjusted ranks created are transformed into the range 1-20 with the same proportions.
Our ranking evaluation metrics are variations of known metrics performed on the adjusted ranks instead of the standard ranks.

Ranking Evaluation Metrics

These metrics evaluate the difference between the predicted table and the true table, based on the adjusted ranking only (i.e. these metrics are independent of the predicted and true number of points for each team).

We denote the true adjusted rank of team i as r_i^* and the predicted adjusted rank of team i as r_i.

We have used the following evaluation metrics:

1. Hamming - the normalized sum of adjusted ranks differences (in absolute value) between the two tables. We have reported the complement value (1 minus the normalized distance), i.e. similar tables obtain larger values.
   $1 - minmax\_norm(\Sigma_{i \in teams} |r_i - r_i^*|)$
2. Spearman - spearman correlation of the two tables using the adjusted ranks instead of the standard ranks (spearman correlation is defined as the pearson correlation of the ranks vectors).
   $pearson\_correlation((r_1, \ldots, r_{20}), (r^*_1, \ldots, r^*_{20}))$

3. Adjusted MAP - a variation of the MAP metric, when P@K is defined as the intersection of the set of the first k teams in the **predicted** table and the set of the first k teams in the **true** table (instead of using the intersection between the first k teams and the relevant set in the original MAP).
   $\frac{1}{20}\Sigma_{k=1,\ldots 20} P@k$, where
   $P@k = minmax\_norm(|\{first\ k\ teams\ in\ the\ predicted\ table\} \cap \{first\ k\ teams\ in\ the\ true\ table\}|)$

Additional Metrics

For each method we will also report the average points error between the two tables: $\frac{1}{20}\Sigma_{i\in teams}|p_i - p^*{}_i|$, where p_i^*, p_i are the true and predicted number of points of team i respectivelly.

In addition, for the second approach models (Logistic Regression and Neural Network) we will also report classification metrics for the probabilities learning sub task:

1. Accuracy - ratio between the number of correct predictions and total number of observations.
2. Ordinal Accuracy - same as accuracy, but a misclassification involving a draw is counted as half as a misclassification involving a win and a draw. The motivation for this approach is that predicting a draw when the true outcome is a win of one of the teams is not as bad as predicting one team wins when it loses.

Note that all metrics (except the points error) have been normalized to the range between 0 and 1.

Basic Part Results

The results of the basic part are given in the following table (mean and stdev by leave-one-season-out-cross-validation):

<table>
<tr><th></th><th>Hamming</th><th>Adjusted MAP</th><th>Spearman</th><th>Points Error (L1)</th><th>Accuracy</th><th>Ordinal Accuracy</th></tr>
<tr><td>LinReg</td><td>0.698
stdv: 0.066</td><td>0.490
stdv: 0.087</td><td>0.527
stdv: 0.180</td><td>12.821
stdv: 1.832</td><td></td><td></td></tr>
<tr><td>LogReg + E</td><td>0.764
stdv: 0.034</td><td>0.587
stdv: 0.064</td><td>0.723
stdv: 0.103</td><td>8.934
stdv: 1.259</td><td rowspan="2">0.499
stdv: 0.037</td><td rowspan="2">0.576
stdv: 0.038</td></tr>
<tr><td>LogReg + S</td><td>0.751
stdv: 0.036</td><td>0.561
stdv: 0.065</td><td>0.722
stdv: 0.096</td><td>9.111
stdv: 1.352</td></tr>
<tr><td>BasicNN + E</td><td>0.783
stdv: 0.053</td><td>0.607
stdv: 0.074</td><td>0.770
stdv: 0.109</td><td>8.517
stdv: 1.513</td><td rowspan="2">0.527
stdv: 0.049</td><td rowspan="2">0.603
stdv: 0.051</td></tr>
<tr><td>BasicNN + S</td><td>0.780
stdv: 0.053</td><td>0.606
stdv: 0.075</td><td>0.751
stdv: 0.154</td><td>8.634
stdv: 1.617</td></tr>
</table>

LinReg - Linear Regression | LogReg+E - Logistic Regression + Expectation | LogReg+S - Logistic Regression + Simulation |BasicNN+E -Basic Neural Network + Expectation | BasicNN+S - Basic Neural Network + Simulation

As we can see from the table above, the second approach outperforms the first approach and the deep model method performs better than the logistic regression method. In the simulation ranking method we have used 10 iterations, as a greater number of simulations will start to converge to the expectation.
From the table above it can be seen that there is no significant difference in the second approach models between the two ranking methods expectation and simulation. In addition, we can see that the BasicNN model performed better than the Logistic Regression and Linear Regression models, which implies that the linearity assumption in the problem we are trying to solve does not hold.

Advanced Part

In the advanced part we will try to improve the second approach by also addressing the order of matches played by each team, and add dependency between the outcome of a match and the outcomes of the teams' previous matches.
This will be done by modifying the network architecture used in the basic part. That is, we will also use recurrent architectures to implement the second approach.
In addition to the data used in the basic part we will also use additional features representing the previous matches outcomes for each team.
We will use the same evaluation metrics defined in the basic part.

Data Manipulation

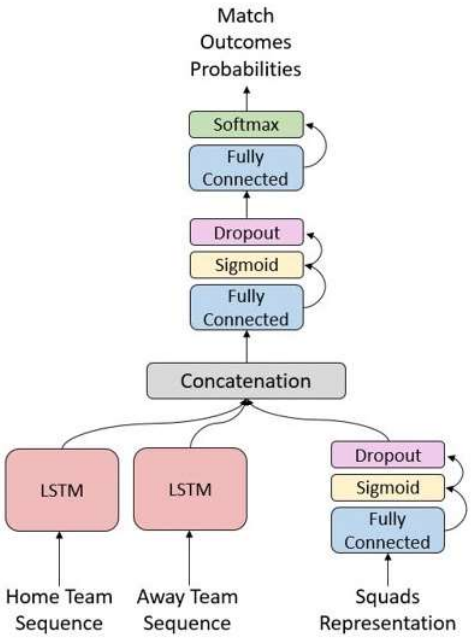
In addition to the input vector $x_{(i,j),s}$ defined in the basic part, we have defined the history input sequence $h_{i,f,s}$ - which represents the matches history of team i in fixture f of season s. The history sequence contains up to 5 encoded previous match results. The aggregated input vector of the advanced architecture is now consists of $x_{(i,j),s}$, $h_{i,f,s}, h_{j,f,s}$.

Architecture

The squad representation $x_{(i,j),s}$ is inserted into a FC layer (as in the basic part), while simultaneously the two history sequences are inserted into two separate LSTM layers. The three outputs are then concatenated and inserted into a sequence of FC layers and finally transformed by a softmax activation layer to get match outcome probabilities (similar architecture as in the basic part). The full architecture of the advanced part is visualized in the 'AdvancedNN Architecture' figure.

Note that the length of the history sequences varys as function of f (for instance, in the first fixture there is no history). To tackle this challenge we have used a batch size of 1 so we can slice the encoded history without losing information. The neural network is

AdvancedNN Architecture



trained with respect to the multiclass cross entropy loss function (same loss as in the basic part).

Inference

In order to predict the final table of season s given the trained model, matches of the test season are simulated one by one, sorted by date. For each match (and it's teams' histories) the model predicts probabilities for match outcome, and the result of the match is sampled from the predicted distribution.

Then, the history is updated based on the last fixture simulation results, and the next fixture is predicted based on the updated history.

In order to achieve robustness, the simulation is repeated 100 times and the predicted number of points for each team is the average over all simulations.

Results

The results of the advanced part are given in the following table, in comparison to the neural network model presented in the basic part:

	Hamming	Adjusted MAP	Spearman	Points Error (L1)	Accuracy	Ordinal Accuracy
BasicNN + E	0.783 stdv: 0.053	0.607 stdv: 0.074	0.770 stdv: 0.109	8.517 stdv: 1.513	0.527 stdv: 0.049	0.603 stdv: 0.051
BasicNN + S	0.780 stdv: 0.053	0.606 stdv: 0.075	0.751 stdv: 0.154	8.634 stdv: 1.617		
AdvNN	0.797 stdv: 0.057	0.625 stdv: 0.089	0.759 stdv: 0.159	8.296 stdv: 1.115	0.412 stdv: 0.023	0.536 stdv: 0.021

BasicNN+E -Basic Neural Network + Expectation | BasicNN+S - Basic Neural Network + Simulation | AdvNN - Advanced Neural Network

The advanced model achieved better results for most of the final-table evaluation metrics, but none of the improvements was significant. In the probabilities subtask metrics there was a significant drop in the advanced model. This can be explained by the fact that these metrics are calculated during the simulations, and the history-sequences component of the input is obtained by simulations and may be different from the actual true sequence for the same match.

Obviously the true sequences are not available during test-time (otherwise the final table is actually known), but in order to correctly evaluate the performance of the classification model alone, one can insert the true history sequences into the model and obtains the results: Accuracy 0.522, Ordinal Accuracy 0.602.

Accuracy	Ordinal Accuracy
0.522 stdv: 0.044	0.602 stdv: 0.049

These results are similar to the results obtained by the BasicNN model. This implies that simulating the match outcomes generates a trailing error, which hurts the model's performance.

Creative Part

Notice that the match outcome probabilities subtask is an ordinal classification task - the classes of {Home Wins, Draw, Home Loses} are ordered, meaning: Home Wins < Draw < Home Loses (as reflected in the ordinal accuracy measure). In the basic and advanced parts the order relation of the classes was not utilized during the classification subtask. Therefore, in the creative part we will try to improve the match outcome probabilities subtask (and the predicted tables induced by it) by treating the classification task as an ordinal classification task.

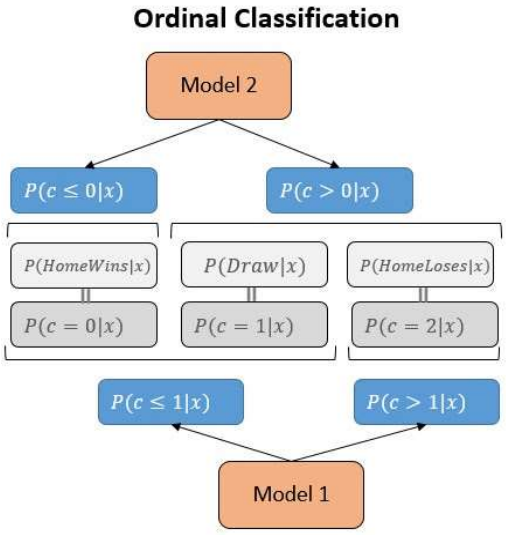
We will present two approaches for ordinal classification, based on two papers - one approach will be implemented to improve both the logistic regression and the basic NN methods, and the other approach is a new architecture of a deep model that utilizes the classes order.

A Simple Approach to Ordinal Classification (Eibe Frank and Mark Hall, 2001)

Link to paper: https://link.springer.com/chapter/10.1007/3-540-44795-4_13

The paper presents a simple high level approach for ordinal classification that reduces the multiclass classification task into multiple binary classification tasks, and aggregates their results.
In our case, instead of using a single multiclass model to directly predict the probabilities $P(HomeWins), P(Draw), P(HomeLoses)$, we train two separate binary models in order to predict the probabilities:
$P_1(HomeLoses), P_1(\neg HomeLoses)$ by the first model. and
$P_2(HomeWins), P_2(\neg HomeWins)$ by the second model.
Then, the aggregated match outcome distribution is given by:

$$P_{agg}(HomeWins) = P_2(HomeWins)$$
$$P_{agg}(Draw) = P_2(\neg HomeWins) - P_1(HomeLoses)$$
$$P_{agg}(HomeLoses) = P_1(HomeLoses)$$



We have implemented this approach to improve both the logistic regression method and the basic neural network method. For instance, instead of a single multiclass logistic regression model, we have used two binary logistic regression models as explained above. The results are given in the following table, comparing each method separately:

Logistic Regression Comparison:

	Hamming	Adjusted MAP	Spearman	Points Error (L1)	Accuracy	Ordinal Accuracy
LogReg	0.764 stdv: 0.034	0.587 stdv: 0.064	0.723 stdv: 0.103	8.934 stdv: 1.259	0.499 stdv: 0.037	0.576 stdv: 0.038
OrdLogReg	0.765 stdv: 0.044	0.585 stdv: 0.071	0.730 stdv: 0.109	8.914 stdv: 1.263	0.493 stdv: 0.023	0.580 stdv: 0.029

LogReg - Logistic Regression | OrdLogReg - Ordinal Logistic Regression. All methods use the expectation ranking method.

Deep Models Comparison:

	Hamming	Adjusted MAP	Spearman	Points Error (L1)	Accuracy	Ordinal Accuracy
BasicNN	0.783 stdv: 0.053	0.607 stdv: 0.074	0.770 stdv: 0.109	8.517 stdv: 1.513	0.527 stdv: 0.049	0.603 stdv: 0.051
OrdNN	0.785 stdv: 0.056	0.611 stdv: 0.089	0.752 stdv: 0.129	8.370 stdv: 1.587	0.509 stdv: 0.045	0.590 stdv: 0.046

BasicNN -Basic Neural Network | OrdNN -Ordinal Neural Network. All methods use the expectation ranking method.

The results in the above table are only represented only by the ranking method of expectation, since in the basic part we saw that expectation and simulation produces similar results.

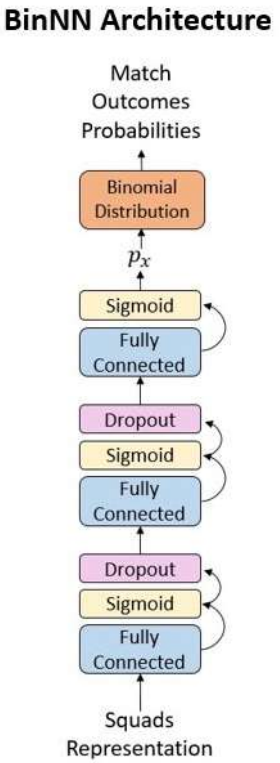
In both cases the ordinal approach did not change the results significantly. This might be related to the fact that this ordinal classification approach is more effective as the number of classes grows, and might suit better to problems with a larger number of classes (in our problem there are only three classes). The paper also investigates the same hypothesis by running several simulations over different discretizations of the target variable to classes.

Classification of Ordinal Data Using Neural Networks (Joaquim Pinto da Costa, Jaime S. Cardoso, 2005)

Link to paper: https://link.springer.com/chapter/10.1007/11564096_70

The paper presents a high level approach for ordinal classification using neural networks. The general idea is to force the output class distribution to give the most significant probability mass to the predicted class, and then gradually reduce the probability mass as the classes move away from the predicted class. This is achieved by forcing the distribution to be a binomial distribution.

The full architecture we have implemented is visualized in the 'BinNN Architecture' figure. The input vector representing the squads is inserted into FC layers (similarly to the BasicNN architecture), while in the last layer, the number of units is not 3, but 1. This one unit is transformed by a Sigmoid activation layer, and performs as a probability p_x. Then, we induce a binomial distribution $Bin(K-1, p_x)$, where K is the number of classes (3 in our case), and by that we obtain a distribution over {HomeWins, Draw, HomeLoses}.



In the paper, the loss function that was used was Mean Squared Error (MSE) between the predicted probabilities vector and the true one-hot probability vector. In our implementation we tried to use both MSE and Categorical Cross Entropy (as we used in previous methods).

The results are given in the following table, compared to the best models and approaches achieved so far:

	Hamming	Adjusted MAP	Spearman	Points Error (L1)	Accuracy	Ordinal Accuracy
BasicNN	0.783 stdv: 0.053	0.607 stdv: 0.074	0.770 stdv: 0.109	8.517 stdv: 1.513	0.527 stdv: 0.049	0.603 stdv: 0.051
BinNN + CE	0.782 stdv: 0.053	0.597 stdv: 0.079	0.772 stdv: 0.107	8.656 stdv: 1.208	0.412 stdv: 0.037	0.633 stdv: 0.031
BinNN + MSE	0.782 stdv: 0.050	0.602 stdv: 0.077	0.763 stdv: 0.099	9.826 stdv: 2.516	0.464 stdv: 0.035	0.627 stdv: 0.038

BasicNN - Basic Neural Network | BinNN + CE - Binomial Neural Network + Cross Entropy | BinNN + MSE - Binomial Neural Network + Mean Square Error Loss. All methods use the expectation ranking method.

The most noticeable effect of the binomial neural network is a significant improvement of the ordinal accuracy metric, and a significant drop in the standard accuracy. When comparing the two loss functions used with the binomial neural network, we notice that the cross-entropy loss archives the most extreme results. Despite these significant changes, the final-table evaluation metrics did not change significantly. This might imply that the subtask probabilities metrics are not indicative enough to evaluate the classification subtask in the context of the table pediction class.

GitHub Repository

Source code and extended version of the project report are available at the GitHub repository: FinalProjectML2.