DEPARTMENT OF INFORMATICS

MSc in Data Science

Thesis

PRODUCT PLACEMENT RECOMMENDATION:
Exploring Machine Learning approaches to decide how to position slot games on a website to maximize performance

Author: Andreas Stavrou

Supervisor: Dr. Theodoros Lappas

Athens, November 2022

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μεταπτυχιακό στην Επιστήμη των Δεδομένων

Διπλωματική Εργασία

ΠΡΟΤΑΣΗ ΤΟΠΟΘΕΤΗΣΗΣ ΠΡΟΙΟΝΤΩΝ:
Ερευνώντας μεθόδους Μηχανικής Μάθησης στην απόφαση τοποθέτησης παιχνιδιών σε μία ιστοσελίδα με σκοπό την μεγιστοποίηση απόδοσης

Συγγραφέας: Ανδρέας Σταύρου

Επιβλέπων: Δρ. Θεόδωρος Λάππας

Αθήνα, Νοέμβριος 2022

Abstract

Product placement is becoming an increasingly important way for brands to reach their target audience in subtle ways. Businesses are using product placement to increase their sales, brand awareness, and draw in customers – all without "traditional" ads. It's the art of determining where your products appear within a store or website through planning, negotiation, and design. Behavioral science studies show that product placement does have a net positive impact on product performance and brand awareness. But can we find the best position a product should be placed at to attract more customers?

In this thesis, various machine learning approaches were explored that identify the best position of a slot game on a given website, in order to maximize turnover, number of bets and player loyalty.

To accomplish the above, a three-step approach was implemented. As a rough introduction, the first step considered is a regression task that predicts the daily turnover of a game based on its unique characteristics and given location. Following that, the second step is to identify key features that influence the algorithm’s decision in order to implement an interpretability pipeline that gives a detailed output of feature importance. In the last step, counterfactual explanations were used combined with our algorithm to implement a system that recommends the ideal position of each game given a specific goal.

This report is separated into three parts, each of which assesses the steps providing the description, development and concluding remarks of each step.

Περίληψη

Η τοποθέτηση προϊόντων έχει καταστεί ένας ολοένα και αυξανόμενος σημαντικός τρόπος για τις εταιρίες με σκοπό να προσεγγίσουν το κοινό τους με διακριτικούς τρόπους. Οργανισμοί χρησιμοποιούν την τοποθέτηση προϊόντων για να αυξήσουν τις πωλήσεις, την αναγνωρισιμότητα, και να προσελκύσουν πελάτες – όλα αυτά χωρίς «παραδοσιακές» διαφημίσεις. Είναι η τέχνη του να καθορίζεις που θα εμφανιστούν τα προϊόντα σε ένα φυσικό κατάστημα ή σε μία ιστοσελίδα μέσω της οργάνωσης, διαπραγμάτευσης και σχεδιασμού.

Μελέτες της συμπεριφορικής επιστήμης δείχνουν ότι η σωστή τοποθέτηση προϊόντων μπορεί να έχει θετική επίδραση στην επίδοση προϊόντων και στην αναγνωρισιμότητα των εταιριών. Το ερώτημα είναι, έχουμε τη δυνατότητα να ανακαλύψουμε τη βέλτιστη θέση που πρέπει να τοποθετηθεί ένα προϊόν ώστε να προσελκύσει περισσότερους από το αναμενόμενο πελάτες;

Στην παρούσα διπλωματική, ερευνήθηκαν αρκετές προσεγγίσεις μηχανικής μάθησης για να αναγνωρίσουν την καλύτερη θέση ενός παιχνιδιού σε κάποια ιστοσελίδα, με σκοπό την μεγιστοποίηση του τζίρου, του αριθμού των στοιχημάτων και την αφοσίωση του πελάτη.

Για να επιτευχθούν τα παραπάνω, εφαρμόστηκε μία μέθοδο τριών σημείων. Σαν μία στοιχειώδη εισαγωγή, το πρώτο σημείο αφορά την χρήση διάφορων αλγορίθμων με σκοπό εκτίμηση του τζίρου ενός παιχνιδιού βασιζόμενη στα μοναδικά χαρακτηριστικά του και στη θέση του. Ακολούθως, το δεύτερο σημείο είναι η αναγνώριση των βασικών χαρακτηριστικών που επηρεάζουν την απόφαση του αλγόριθμου με σκοπό την δημιουργία ενός καναλιού που θα εξάγει αναλυτική εξήγηση της σπουδαιότητας των συντελεστών. Στο τελευταίο σημείο, χρησιμοποιήθηκαν επεξηγήσεις *counterfactuals* οι οποίες συνδυαστικά με τον αλγόριθμο του πρώτου σημείου αποτελούν το σύστημα που θα προτείνει την κατάλληλη θέση κάθε παιχνιδιού δοσμένου ενός συγκεκριμένου στόχου.

Αυτή η διπλωματική εργασία χωρίζεται σε τρία μέρη, το καθένα του οποίου αναλύει καθένα από τα τρία σημεία που αναφέρθηκαν παραπάνω.

Acknowledgements

I would like to thank Dr. Theodoros Lappas for his significant contribution and guidance throughout the course of this thesis. His advice and technical recommendations were instrumental to its completion, and despite the density of the material dealt with, Dr. Lappas's patience and guidance allowed me to dissect the material and present it in a way that advanced my research.

Additionally, I would like to express my deepest gratitude to Ms. Marion Nikoloudaki, Mr. Nick Fournogerakis, Mr. Konstantinos Giorgas and Mr. Konstantinos Kanaris from Light & Wonder. Their immense knowledge base and experience in the area of Machine Learning have encouraged me throughout my academic research, and have provided me with insightful guidance on how the theory is applied in a professional environment. Our collaboration in their unique field exposed me to the nexus between online gambling market and machine learning resulting in the creation of the strategies discussed in this thesis.

Thanks to their invaluable support, I have been able to produce a thesis that I am confident will serve as a powerful foundation for my future research, studies, and work in the field.

List of Tables

List of Figures

Table of Contents

1. **Problem and Approach**

1.1 Introduction

In general, recommending the ideal position of a game in an online betting website is considered very challenging. There exist multiple different parameters that have impact when considering the best place for a game on a given day and on a given website. First one must prove that the position itself is of great meaning to the performance of the game. Following that, we must factor that the "better" positions are very limited, and we cannot just follow the greedy approach and place every game on top positions. Combined with extra constraints, such as operators' agreements with the website, game popularity and betting characteristics, ideal positioning becomes an even more challenging task.

To frame the problem, the task is to find the best possible position of each game on any given day that will maximize its performance, the turnover, the number of bets and the brand awareness. To accomplish the above, a three-step approach was implemented.

1.2 Process

The first step was to deploy an algorithm to predict the turnover of a game on a given day. This process involved data wrangling and feature engineering in order to find the best combination of features to better predict the outcome. After the feature engineering, a combination of algorithms was used in an ensemble technique, in which Extreme Gradient Boosting and Deep Neural Networks combined to form one predictor. Iterating through feature engineering and modeling with parameters experimentation yielded the best possible results.

The next step of this approach was to be able to develop a model explainability pipeline which would output the feature importance in each decision that the algorithm made. This step was vital to guide us through data issues and to better understand what our predictor was learning and then with data engineering we managed to make our predictor learn differently.

The final step was to recommend a games' position and it was made possible through our algorithm from the first step combined with counterfactuals explanations.

1.3 Methodology

The following graph illustrates the methodology followed:

Figure 1.1: Workflow of the Game Positioning Strategy

1.4 Remarks

As a general remark, the strategy developed provides proof that position is indeed an important factor in a games' performance and our predictor combined with counterfactuals explanations can successfully find the ideal position to maximize a games' turnover. This underlines the importance that machine learning models can have in commerce and their ability to provide robust signals in marketing strategies.

2. Data Extraction

The entire strategy can be divided into 5 main parts, as illustrated in Figure 1 with the first part being the data extraction. The data provided by the company was not ready to be consumed by an algorithm and data wrangling methods were involved to finally make a complete dataframe with the data given. Two different sets of data were given, the first had data regarding the positioning of each game on any given day, information about each game such as unique characteristics, information about the themes of the games, data about the location grid which hosted the games and finally information about the pages that were hosting the games. The second set had performance information and metrics for each game on a given date. To ensure that confidentiality was preserved, the performance data was aggregated on a game - day level to summarize performance to secure player's identity and information.

2.1 Data challenges

Given those two sets, the next task was to properly bind those records together. However, those two sets originated from two different databases and IDs between the records did not match, therefore a more sophisticated way to bind those records was required, other than a join on one feature. To overcome this issue, a join with two filters was applied to each record, one for the exact date and one on the games' name. The first one was straightforward as we can match exact dates, however the matching on the games' name was not, as those two different databases had small differences in onomatology of each game. To correctly match the names of those games, a function was created that used the Levenshtein distance and matched all records with 90% similarity ratio or more. The Levenshtein distance is a text similarity measure that compares two words and returns a numeric value representing the distance between them. The distance reflects the total number of single-character edits required to transform one word into another.

Following that, data was cleaned, and duplicated records were removed. Data quality issues were faced during this initial step, as we had records that had two different positions on a single date performing the same turnover. This data quality issue could be detrimental for our approach because we want to investigate the correlation of the position to the games' performance. The dataframe, which had records with good and bad positions, having the same performance, could point the algorithm into entirely different results. Since we did not have

the option to impute those values and did not have the knowledge of which values to keep, a decision was made to keep all records with the highest position in one dataframe and all records with the lowest position into another, preprocess and model and then compare these results with each other. Those findings hinted to us that we can safely keep the dataframe with the highest positions; however these quality issues may affect our algorithms performance due to the small number of records given.

Turnover and performance in general are heavily influenced by players' "favorite" games. A player could have a game as their favorite, or search for it right away and play. Therefore, a significant portion of the total turnover could have come out of other factors rather than the games' position. If a game is famous and peoples' favorite, it will have higher turnover despite a lower position. If we wanted to be robust, we should factor in only Turnover from people first coming in, without any favorites to play. Then we could see how the position would play a role in turnover, because right now we have a lot of variability out of all those other factors. To remedy the above, a control feature was created to ingest occurrence controls into each game and mute "strong" and favorite games. However, the dataframe we had, during the exploratory data analysis, hinted that the position did indeed play a significant role in the performance of each game despite the above.

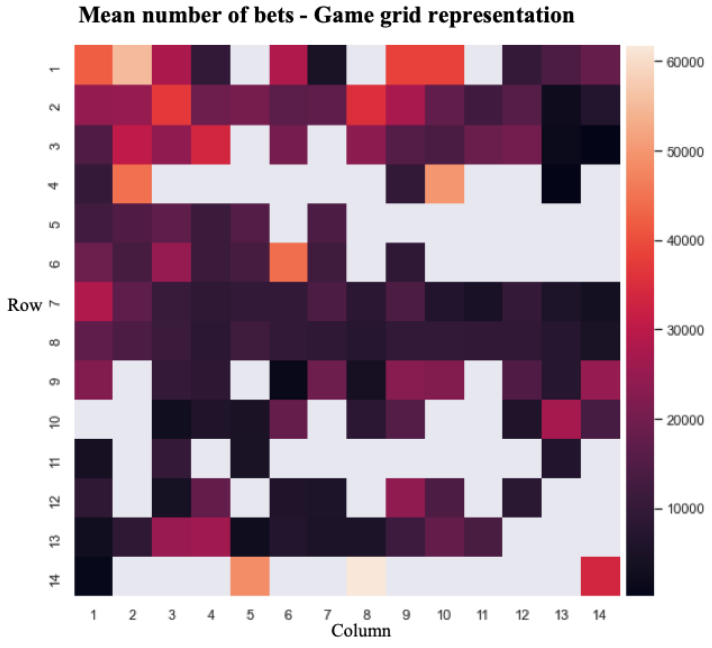

Figure 2.1: Mean number of bets – Game grid representation: Each tile represents a unique position. This figure represents the mean number of bets for all records for each position. The $14x14$ grid represents the actual grid of games displayed on a website.

Other than that, the data which was offered, had an abundance of information regarding the position of the game, the games' characteristics, the performance of the game and betting metrics specific to each game such as RTP (Return to Player), volatility and betting size range.

2.2 Data dictionary

For completeness, this report incorporates a data dictionary for better understanding of the experimentations. See Appendix – The data.

3. Preprocessing

Data preprocessing is important in any machine learning approach. Data preprocessing, which is a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure or for injection into an algorithm.

Given the nature of our task, the correct preprocessing of the data was vital. The data that was given was heavily imbalanced regarding the games' itself, containing many performance records for one game and nearly none for another. During experimentation, when that data was given to the algorithm, the feature importance showed that performance was influenced by the game itself and that it had no correlation with the positioning whatsoever. However, this was not in line with the findings from the exploratory data analysis and the above highlighted the data quality issues which had to be addressed before moving forward. Missing values were also addressed during this phase of the process and filled with either an iterative process or with the mean depending on the percentage of missing values.

3.1 Experiments

Before presenting the feature engineering that was implemented, it is important to showcase the procedure and the experimentation that took place in order to understand and justify the need for the specific engineering.

During the initial prototyping of the algorithm, the feature importance of the predictor showed that the estimation of the target variable was influenced a lot by game specific factors and unique characteristics. However, this was only the case because of imbalanced game records which favored the game itself and pointed the algorithm into that direction. The predictor had witnessed many good performances from "strong" or "famous" games that learned to estimate performance only with the game in mind. We also had to factor in new games that were presented to the game grid. A new game, even if it is presented in top positions of the grid, will not achieve great performance and that will weigh down the importance of good position. All the above guided us through a direction of feature engineering in which game characteristics and occurrences were muted in order to reduce some of the problems variability that we cannot control or that we should not factor in. The task is not to make the algorithm recognize "strong" games, but rather recognize strong positions.

3.2 Feature engineering

The question that needs to be answered is whether the position of each game influences its performance and to what degree. Therefore, the main concern regarding feature engineering was to highlight the features that corresponded to position data, create new features of position data that was in line with the business knowledge of the position grid, and discard position features that added bias to the algorithm.

To address positioning of each game, column and row features were kept and based on those features, a new attribute was created called field. The field is coming out of the business knowledge and the representation of each grid showing on the website that hosts the games. The field attribute is a categorical attribute that separates the grid into specific fields, 1st, 2nd, 3rd, etc. This attribute, after it showed significant importance when calculating turnover, paved the way for more specific tile importance exploration inside of each field.

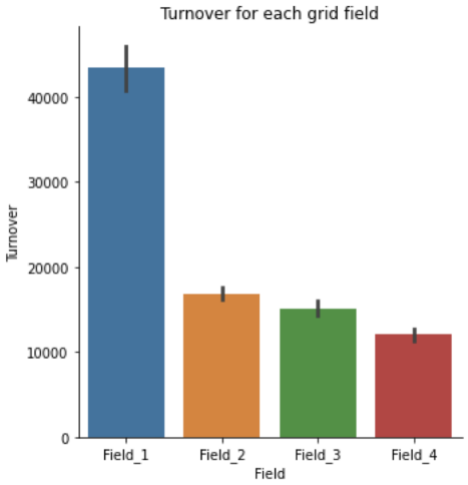

Figure 3.1 Overall turnover in each field

To address the issue with strong games, an occurrence control feature was injected into each record which calculated the number of occurrences and penalized the records with more performances. This feature paired with a similar categorical feature that consisted of the names of the top 25 strongest games managed to mute the importance of the game itself and achieved better performance of the algorithm.

To preserve the importance of seasonality, the date column was further separated and created categorical features of the years, months and days to help the predictor grasp the importance of seasonality.

3.3 Additional preprocessing

The last layer of preprocessing addressed both the categorical features and the numerical features. Regarding the categorical features, encoding was implemented for the variables *“Brand”, “Gametype”, “Page”, “Page_type”, “Provider_Party”, “Theme”*. Regarding the numerical features, scaling was performed to the variables *“Location_Weighting”, “Weighted_Days”, “RTP”, “Lines”, “Reels*”.

4. Modeling

The modeling part of this approach is the first stepping stone for the complete system. The goal is to create a robust algorithm to predict turnover given a dataframe of games' characteristics and position on any given day. This process involved iterating through data engineering and experimentation to pinpoint the best possible combination of feature engineering, algorithm selection, and parameter tuning. In total, the different algorithms that were used were:

Number	Model
1	Ridge Regressor
2	Huber Regressor
3	CatBoost Regressor
4	Support Vector Regression
5	XGBoost Regressor
6	TabNet

Table 4.1: Regression Models

Finally, the best results came with XGBoost (Extreme Gradient Boosting) paired with TabNet which is to be expected as those approaches are among the best approaches one can use with tabular data representation. The other algorithms were combined together with a Stacked Generalization approach which is a form of ensemble, however this approach fell short when compared to the first two.

4.1 Extreme Gradient Boosting

XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression. It's vital to an understanding of XGBoost to first grasp the machine learning concepts and algorithms that XGBoost builds upon supervised machine learning, decision trees, ensemble learning, and gradient boosting.
Supervised machine learning uses algorithms to train a model to find patterns in a dataset with labels and features and then uses the trained model to predict the labels on a new dataset's features. Among the machine learning methods used in practice, gradient tree boosting (J.

Friedman 2001) is one technique that shines in many applications. Tree boosting has been shown to give state-of-the-art results on many standard benchmarks (P. Li. Robust 2010).

Decision trees create a model that predicts the label by evaluating a tree of if-then-else true/false feature questions and estimating the minimum number of questions needed to assess the probability of making a correct decision. Decision trees can be used for classification to predict a category, or in our case regression to predict a continuous numeric value.

A Gradient Boosting Decision Trees (GBDT) is a decision tree ensemble learning algorithm similar to random forest, for classification and regression. Ensemble learning algorithms combine multiple machine learning algorithms to obtain a better model. Both random forest and GBDT build a model consisting of multiple decision trees. The difference is in how the trees are built and combined.

Random forest uses a technique called bagging to build full decision trees in parallel from random bootstrap samples of the data set. The final prediction is an average of all of the decision tree predictions.

The term “gradient boosting” comes from the idea of “boosting” or improving a single weak model by combining it with several other weak models in order to generate a collectively strong model. Gradient boosting is an extension of boosting where the process of additively generating weak models is formalized as a gradient descent algorithm over an objective function. Gradient boosting sets targeted outcomes for the next model in an effort to minimize errors. Targeted outcomes for each case are based on the gradient of the error (hence the name gradient boosting) with respect to the prediction.

GBDTs iteratively train an ensemble of shallow decision trees, with each iteration using the error residuals of the previous model to fit the next model (Tianqi Chen, Carlos Guestrin 2016). The final prediction is a weighted sum of all of the tree predictions. Random forest “bagging” minimizes the variance and overfitting, while GBDT “boosting” minimizes the bias and underfitting.

XGBoost is a scalable and highly accurate implementation of gradient boosting that pushes the limits of computing power for boosted tree algorithms, being built largely for energizing

machine learning model performance and computational speed. With XGBoost, trees are built in parallel, instead of sequentially like GBDT. It follows a level-wise strategy, scanning across gradient values and using these partial sums to evaluate the quality of splits at every possible split in the training set.

4.2 TabNet

TabNet is a deep tabular data learning architecture that uses sequential attention to choose which features to reason from at each decision step (Sercan O. Arik, Tomas Pfister 2019). The TabNet encoder is composed of a feature transformer, an attentive transformer and feature masking. A split block divides the processed representation to be used by the attentive transformer of the subsequent step as well as for the overall output. For each step, the feature selection mask provides interpretable information about the model's functionality, and the masks can be aggregated to obtain global feature important attribution. The TabNet decoder is composed of a feature transformer block at each step.

In the feature transformer block, a 4-layer network is used, where 2 are shared across all decision steps and 2 are decision step-dependent. Each layer is composed of a fully connected (FC) layer, BN and GLU nonlinearity. An attentive transformer block example – a single layer mapping is modulated with a prior scale information which aggregates how much each feature has been used before the current decision step. sparsemax is used for normalization of the coefficients, resulting in sparse selection of the salient features.

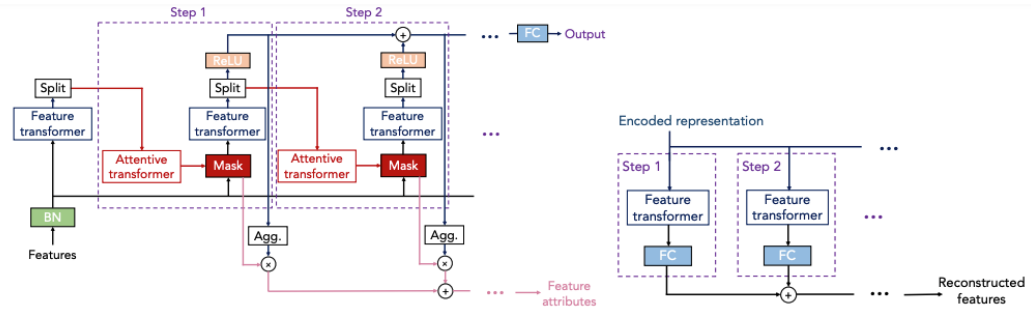

Figure 4.1: (a) TabNet encoder, composed of a feature transformer, an attentive transformer and feature masking. A split block divided the processed representation to be used by the attentive transformer of the subsequent step as well as for the overall output. For each step, the feature selection mask provides interpretable information about the models' functionality, and the masks can be aggregated to obtain global feature important attribution. (b) TabNet decoder, composed of a feature transformer block at each step. (c) A feature transformer block example – 4 – layer network is shown, where 2 are hared across all decision steps and 2 are decision step-dependent. Each layer is composed of a fully connected layer, BN and GLU nonlinearity. (d) An attentive transformer block example – a simple layer mapping is modulated with a prior scale information which aggregates how much each feature has been used before the current decision step. sparsemax (Martings and Astudillo 2016) is used for normalization of the coefficients, resulting in sparse selection of the salient features.

4.3 Stacked Generalization

Stacked generalization or stacking is an ensemble machine learning algorithm. It uses a meta-learning algorithm to learn how to best combine the predictions from two or more base machine learning algorithms. The benefit of stacking is that it can harness the capabilities of a range of well-performing models on a classification or regression task and make predictions that have better performance than any single model in the ensemble. Stacking addresses the question: Given multiple machine learning models that are skillful on a problem, but in different ways, how do you choose which model to use?
The architecture of a stacking model involves two or more base models, often referred to as level-0 models, and a meta-model that combines the predictions of the base models, referred to as a level-1 model.

- Level-0 Models (Base-Models): Models fit on the training data and whose predictions are compiled.
- Level-1 Model (Meta-Model): Model that learns how to best combine the predictions of the base models.

The meta-model is trained on the predictions made by base models on out-of-sample data. That is, data not used to train the base models is fed to the base models, predictions are made, and these predictions, along with the expected outputs, provide the input and output pairs of the training dataset used to fit the meta-model.

In this thesis Ridge, Huber, Support Vector and CatBoost regressors were used as the base models and built upon those, the Stacking Regressor combined their predictions to form a final estimation.

Ridge regression is a model tuning method that is used to analyze any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

Huber regression (Huber 1964) is a regression technique that is robust to outliers. The idea is to use a different loss function rather than the traditional least-squares; we solve:

$$\underset{\beta}{\text{minimize}} \quad \sum_{i=1}^{m} \phi(y_i - x_i^T \beta)$$

for variable $\beta \in R^n$, where the loss ϕ is the Huber function with threshold $M > 0$,

$$\phi(u) = \begin{cases} u^2 & \text{if } |u| \leq M \\ 2Mu - M^2 & \text{if } |u| > M. \end{cases}$$

This function is identical to the least squares penalty for small residuals, but on large residuals, its penalty is lower and increases linearly rather than quadratically. It is thus more forgiving of outliers.

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression used the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points.

CatBoost builds upon the theory of decision trees and gradient boosting. The main idea of boosting is to sequentially combine many weak models (a model performing slightly better than random chance) and thus through greedy search create a strong competitive predictive model. Because gradient boosting fits the decision trees sequentially, the fitted trees will learn from the mistakes of former trees and hence reduce the errors. This process of adding a new function to existing ones is continued until the selected loss function is no longer minimized.

The best results came out of experimentation using a portion of each and a combination of the stacking regressor regarding the predictions however XGBoost algorithm gave better results in all of the experimentations and therefore was chosen for the next steps of the implementation as well.

4.3 Train – Test Split

The data are split into a training and a test set. The training set will be used to fit the model, while the testing set will be used to examine the predictive ability of the model by comparing the predictions made with the actual true values. The reason for doing so is that the objective of any machine learning model is to predict a feature on unseen, new data. Thus, by evaluating the model on the unseen test data, an actual examination of its predictive ability can be made.

Additionally, in order to more accurately evaluate the predictive ability of the model, K-fold cross validation is used. As explained by James, the intuition behind this is that K-fold cross validation partitions the training dataset into K complementary subsets and in each iteration, one subset is used as a validation subset whereas the rest are used as the training sets (James, et al., 2013). Furthermore, once all the iterations have been completed, the results are averaged and one estimate of the model's predictive ability is given.

Figure 4.2: 5-Fold Cross Validation

4.4 Hyperparameter Tuning

One of the most important aspects of any machine learning model is the tuning of its hyperparameters, which are the parameters that define the architecture of the model. Essentially, this has to do with finding the best design of a model's architecture so as to achieve optimal performance for a given problem. As is often the case for any new machine learning task, the optimal architecture of a model is not known a priori. For this reason, an exploratory procedure must be initialized which will look at a range of possible model architectures and establish the optimal one. This procedure is called hyperparameter tuning and involves fitting

multiple models and choosing the hyperparameters from the model that achieved the best result.

An important difference that should be noted is the distinction between the parameters of the model and its hyperparameters. A model's parameters are computed through training, when the model is optimized and fitted. As a result, the parameters are learned from the training data. On the other hand, the hyperparameters define the architecture of the model and cannot be learned during training.

In this thesis, the Grid Search technique has been used to optimize the hyperparameters of all the regression models. In essence, this method creates a model for every possible combination of all the hyperparameters that were given, evaluates each single model, and then chooses the architecture that produced the best scores.

In the following table, the hyperparameters that were used in each classification model will be explained:

Models	Hyperparameters
Ridge Regressor	**Alpha**: the penalty term that denotes the amount of shrinkage (or constraint) that will be implemented in the equation
Support Vector Regressor	**C**: This parameter controls the strength of the penalty for each misclassified data point. A small value of C assigns a low penalty for misclassified points and thus a decision boundary with a larger margin is chosen. On the contrary, a large value of C assigns a high penalty on misclassified points and thus a decision boundary with a smaller margin is chosen. **Epsilon**: It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. **Gamma**: This hyperparameter is used only with non-linear kernels. It controls the distance of influence of each individual training observation. A high value of gamma gives more importance to nearby points, whereas a low value for gamma indicates that

	far points are also important in adjusting the decision boundary.
Huber	**Alpha**: the penalty term that denotes the amount of shrinkage (or constraint) that will be implemented in the equation. **Epsilon**: It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. **Max Iterations**: Number of iterations that scipy.optimize.minimize(method="L-BFGS-B") has run for.
CatBoost Regressor	**Learning rate:** Is the step size shrinkage used when updating the feature weights at each boosting round. This is used to prevent overfitting and thus make the model more robust. **Max depth**: Similar to the random forest max depth hyperparameter. It controls the maximum depth that each tree will grow on any boosting round. **No of estimators:** Is the number of decision trees that are sequentially added to the model. **Colsample by level:** Random subspace method. The percentage of features to use at each split selection, when features are selected over again at random.
XGBoost Regressor	**Learning rate**: Is the step size shrinkage used when updating the feature weights at each boosting round. This is used to prevent overfitting and thus make the model more robust. **No of estimators**: Is the number of decision trees that are sequentially added to the model.

Table 4.2: Hyperparameters of each regression model

4.5 Evaluation

For the purpose of this implementation, which is aligned with the business model of the company, we primarily care about the monetary value that has been generated by the particular algorithm. For each record on a given day we want to know how well it did in terms of turnover and therefore our main evaluation metric is going to be Mean Absolute Error (MAE). MAE is conceptually simpler and also easier to interpret than RMSE: it is simply the average absolute vertical or horizontal distance between each point in a scatter plot and the $Y = X$ line. In other words, MAE is the average absolute difference between X and Y. Furthermore, each error contributes to MAE in proportion to the absolute value of the error. RMSE was also evaluated in the first steps of the modeling to compare algorithms with each other. The first task was to find the more robust algorithm between the abundance of models that we tried and proceed with only one in the steps of explanation and recommendation. The RMSE scores on train and test set for all the algorithms are as shown:

Algorithm	Train RMSE	Test RMSE
Baseline Dummy Mean Regressor	0.4786	0.4786
HuberRegressor	0.2625	0.2587
CatBoostRegressor	0.1551	0.1451
SVR	0.2376	0.2307
XGBRegressor	**0.1203**	**0.0942**
StackingRegressor	0.1533	0.1361
TabNet	0.1604	0.1544
Blend	0.1395	0.1419

Table 4.3: Train and test set RMSE for all algorithms being examined

The test set in which the evaluation took place, was split in buckets depending on the records' turnover. Splitting into buckets and then evaluating each one as a different set was necessary as one Mean Absolute Error number could draw wrong conclusions. A MAE of 3.000$ in the test set for example would be great for games with very high turnover, but alarming for games with low turnover. This is why we need to evaluate our algorithm on each bucket and examine whether we can trust its predictions on all buckets. For completeness, the buckets that were split are the ones below and the splitting was decided on the game's distribution of the turnover.

Bucket	Turnover Range
Bucket 1	[0$ – 5.000$]
Bucket 2	[5.000$ – 10.000$]
Bucket 3	[10.000$ – 15.000$]
Bucket 4	[15.000$ – 25.000$]
Bucket 5	[25.000$ – 35.000$]
Bucket 6	[35.000$ – 300.000$]

Table 4.4: Test buckets split for separate MAE evaluation

The best scoring in each bucket resulted from the deep neural network with 4-layer dense blocks and had the scores below:

Bucket	MAE
Bucket 1	988.50$
Bucket 2	2538.08$
Bucket 3	3869.17$
Bucket 4	5939.24$
Bucket 5	10784.78$
Bucket 6	21488.17$
Overall Test Set	5854.69$
Train Set	5583.74$

Table 4.5: Best MAE scores for each bucket

4.6 Error analysis

Error analysis is the process to isolate, observe and diagnose erroneous ML predictions thereby helping understand pockets of high and low performance of the model. For this analysis, we wanted to explore the errors done by the predictor, the distribution of them, and touch upon some records that the predictor was wrong by a high margin. The comprehension of the model behavior will not only assist in fixing the algorithm and make it more robust but also it will help us have a better understanding of the algorithms' feature importance.

The first thing to consider was the distribution of the errors done by the algorithm. We want an algorithm that is closer to the true values most of the time and we want the errors to be in a

good ratio to the true value. For example, a MAE of 10000$ with a percentage ratio of 10% is significantly better than a MAE of 1000 with a percentage ratio of 1000%.

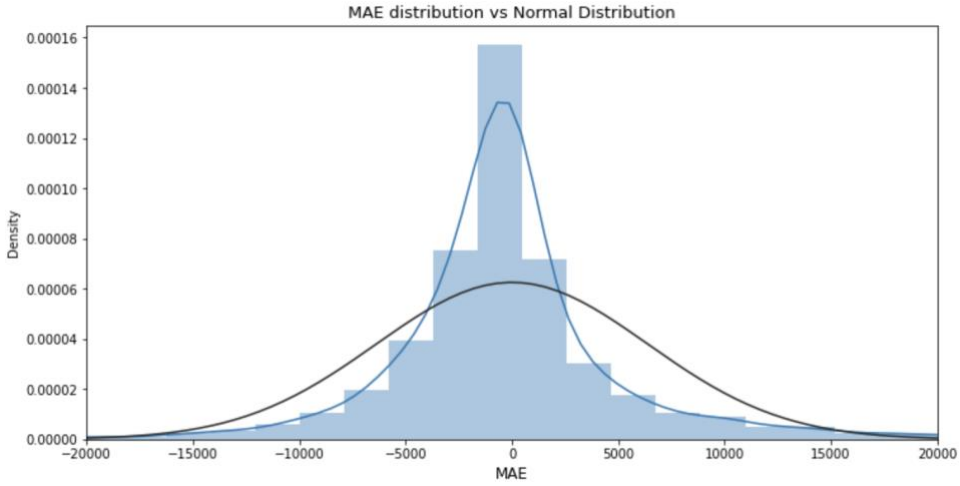

Figure 4.3: MAE distribution when compared to normal distribution

The second thing to consider was the records with high percentage ratio fault. We wanted to know why the predictor had such poor estimations for some records. With this approach, we could potentially find some features that when they do exist, it drives the predictor to make a specific estimation. If we explore and find those features, then we could effectively use them for the third step of this approach, to propose the best possible position for a given game. To implement the above, a high-precision model-agnostic explanation technique was used, called Anchors.

4.7 Error analysis using Anchors

The main focus of this implementation is to feed those "bad" records to this model-agnostic system and explore the features that contribute the most to the wrong estimation of our algorithm. This system explains the behavior of complex models with high-precision rules called anchors, representing local, "sufficient" conditions for predictions (Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin 2018). In their paper, they have introduced novel model-agnostic explanations based on if-then rules, which they call anchors, with an anchor explanation which is a rule that sufficiently "anchors" the prediction locally – such that changes to the rest of the feature values of the instance do not matter. For example, given a game record from our data, the location of a game in $Column < 2$ (good position) and $Row < 5$ (good position), virtually guarantee a prediction of *good turnover*. Anchors are intuitive, easy to

comprehend, and have extremely clear coverage – they only apply when all the conditions in the rule are met, and if they apply the precision is high (by design).

Given our problem, which is a regression task, we first had to alter the task, and turn it to a classification task because that is how anchors can work. Considering that we want to explore errors, we will define two kinds of errors, "Overrated" and "Underrated". The separation line between those two has the true values of each record. For those records that are off by a lot, we choose two of them – one that is overrated and one that is underrated. We implement the model – agnostic system in order to find out why the algorithm is stating that a given record is *Overrated* or *Underrated* and what are the features that contribute the most to such wrong estimation.

Prediction	Anchor	Precision	Coverage
Underestimated	f3 > 6.00 AND f1 <= 0.13 AND f2 > 7.00	0.994604	0.080326

Figure 4.4: Anchors Example 1 – The algorithm will almost always underestimate the turnover of this record because it has $Column > 7, Row > 6$ and $Position\ Weighting <= 0.13$

In the case above, the anchor is very clear, it states that given the record is in "mediocre" position, it will almost certainly classify it as underestimated, in practice, the algorithm will almost always predict less than the actual value.

Prediction	Anchor	Precision	Coverage
Overestimated	f7 <= -0.40 AND f3 <= 3.00 AND f8 <= -0.38 AND...	0.954241	0.101447

Figure 4.5: Anchors Example 2 – The algorithm will almost always overestimate the turnover of this record because it has $Row <= 3$ and a small number of Lines and Reels which is a game characteristic.

In the case above, the anchor stated the opposite, that given the record is in good position, it will almost certainly classify it as overestimated, in practice, the algorithm will almost always predict more than the actual value.

After running the anchors for all the records in the test set, we have concluded that for the records that position is important, then with only the position features, the algorithm will "anchor" the record to one side of the classification line.

5. Model Explainability

The second phase of this approach is to be able to develop a model explainability pipeline which would output the feature importance in each decision that the algorithm makes. This step was vital to guide us through data issues and to better understand what our predictor was learning and then with data engineering we managed to make our predictor learn differently. The techniques that were used for model interpretability were Shapley values, TabNet explanations through the feature selection masks explained in the previous chapter and XGBoost built-in feature importance. Out of those three different techniques, Shapley values offered unique insights into the decision of the algorithm.

5.1 Shapley Explanations

SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions.

Predictive machine learning models like XGBoost become even more powerful when paired with interpretability tools like SHAP. These tools identify the most informative relationships between the input features and the predicted outcome, which is useful for explaining what the model is doing, getting stakeholder buy-in, and diagnosing potential problems. It is tempting to take this analysis one step further and assume that interpretation tools can also identify what features decision makers should manipulate if they want to change outcomes in the future. However, using predictive models to guide this kind of policy choice can often be misleading. The reason relates to the fundamental difference between correlation and causation. SHAP makes transparent the correlations picked up by predictive ML models. But making correlations transparent does not make them causal! All predictive models implicitly assume that everyone will keep behaving the same way in the future, and therefore correlation patterns will stay constant. To understand what happens if someone starts behaving differently, we need to build causal models, which requires making assumptions and using the tools of causal analysis.

Flexible predictive models like XGBoost or LightGBM are powerful tools for solving prediction problems. However, they are not inherently causal models, so interpreting them with SHAP will fail to accurately answer causal questions in many common situations. Unless

features in a model are the result of experimental variation, applying SHAP to predictive models without considering confounding is generally not an appropriate tool to measure causal impacts used to inform policy. SHAP and other interpretability tools can be useful for causal inference, and SHAP is integrated into many causal inference packages, but those use cases are explicitly causal in nature. To that end, using the same data we would collect for prediction problems and using causal inference methods like double ML that are particularly designed to return causal effects is often a good approach for informing policy.

During this project, the additive nature of Shapley values was utilized to analyze and then plot important features and how those features influenced the target variable "Turnover". One of the fundamental properties of Shapley values is that they always sum up to the difference between the game outcome when all players are present and the game outcome when no players are present. For machine learning models this means that SHAP values of all the input features will always sum up to the difference between baseline (expected) model output and the current model output for the prediction being explained. The easiest way to see this is through a waterfall plot that starts at our background prior expectation for a "Turnover" price $E[F(X)]$, and then adds features one at a time until we reach the current model output $F(X)$:

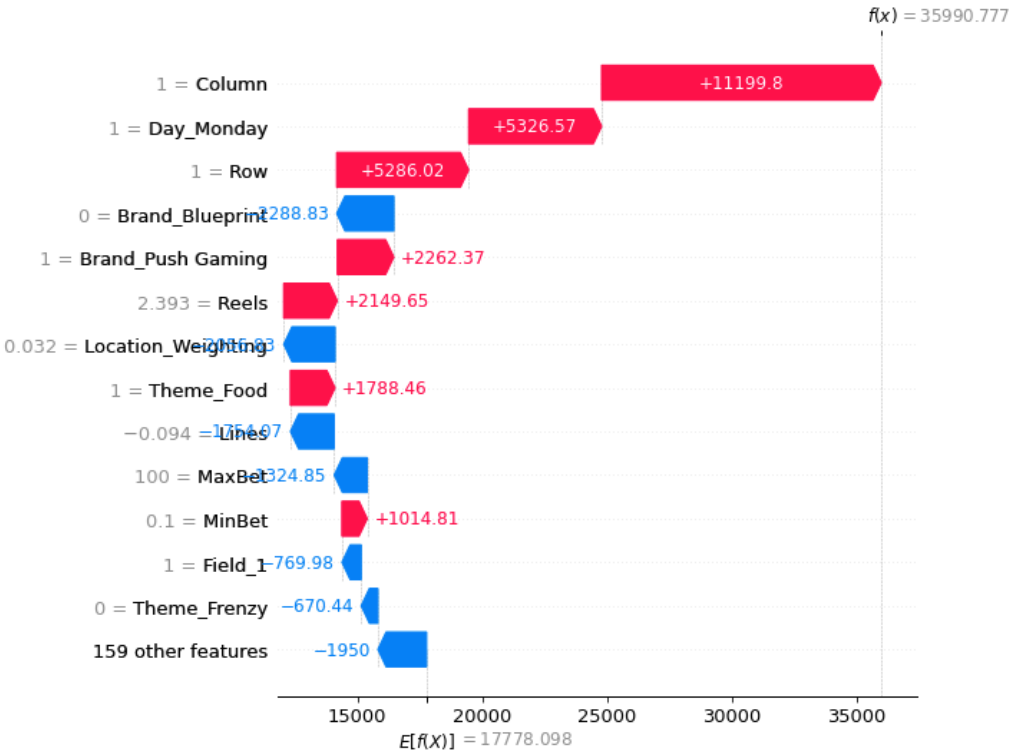

Figure 5.1: This waterfall plot shows how we get from base values to predicted values for a game located on a good position

In the above plot we see how the Column feature with value one, influenced the predictor to add 11.199$ to the turnover and an extra 5.285$ for being in the first Row.

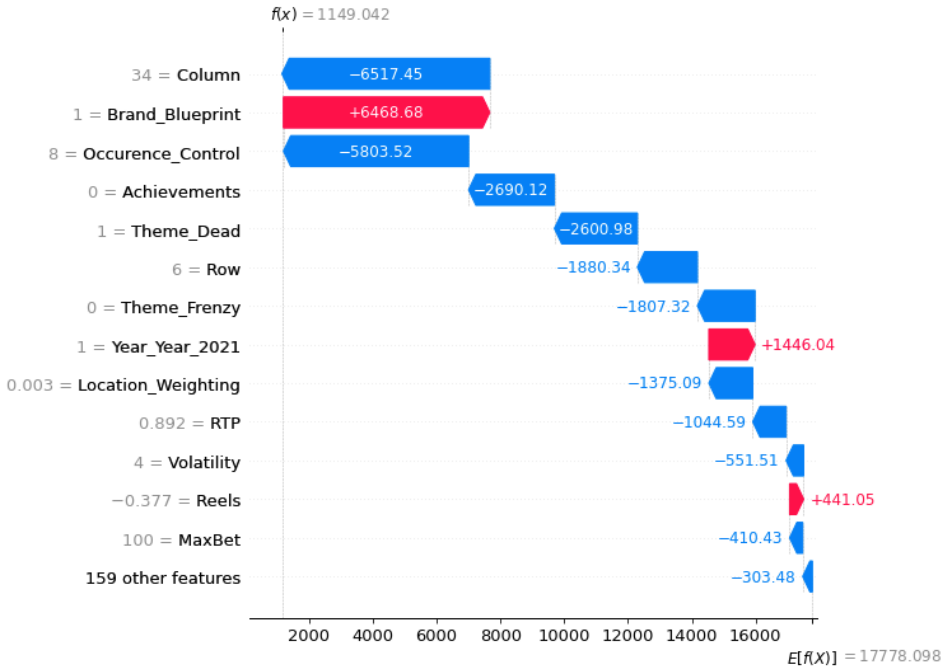

Figure 5.2: This waterfall plot shows how we get from base values to predicted values for a game located on a bad position

In this example however, we see that the bad positioning of the game, influenced the predictor to subtract from the expected “Turnover” value, and to be exact, the predictor estimated that this game will do 6.517$ less for being located on “Column” 64, and it will also do 1.880$ less for being located on “Row” 6.

In general, when utilizing the additive nature of Shapley values, it is clear from our data, that the predictor learns that the better the positioning of any given game, the more the expected turnover.

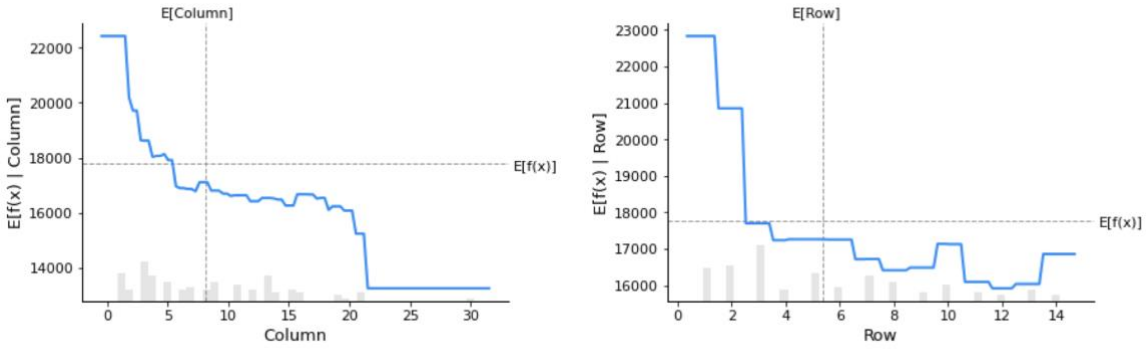

Figure 5.3: Standard partial dependence plot with a single SHAP value overlaid

5.2 TabNet Explanations

TabNet uses sequential attention to choose which features to reason from at each decision step, enabling interpretability and more efficient learning as the learning capacity is used for the most salient features.

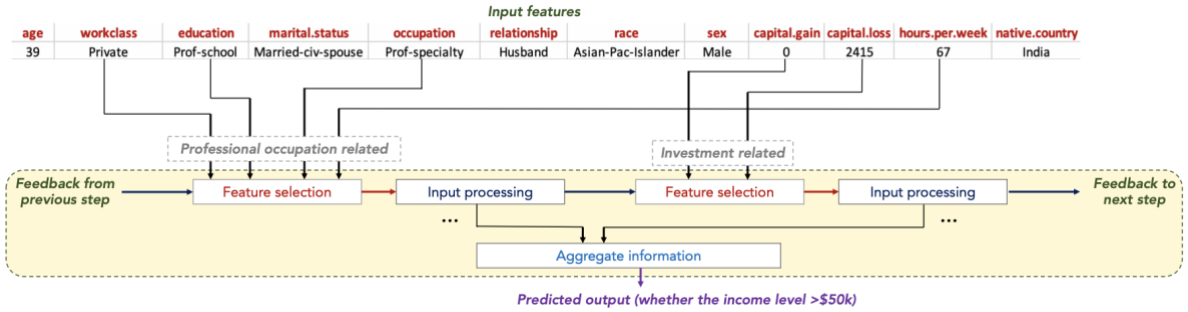

Figure 5.4 TabNet's sparse feature selection exemplified for Adult Census Income prediction (Dua and Graff 2017). Sparse feature selection enables interpretability and better learning as the capacity is used for the most salient features. TabNet employs multiple decision blocks that focus on processing a subset of input features for reasoning.

Regarding interpretability, experiments have shown that TabNet manages to correctly identify the most salient features in real-world datasets. Sercan O. Arık and Tomas Pfister on their research and implementation on TabNet, state that when considering the simple task of mushroom edibility prediction (Dua and Graff 2017), TabNet achieves 100% test accuracy. It is known that "Odor" is the most discriminative feature – with "Odor" only, a model can get > 98.5% test accuracy. This, a high feature importance is expected for it. TabNet assigns an importance score ratio of 43% for it, while other methods like LIME (Ribeiro, Singh, and Guestrin 2016), Integrated Gradients (Sundararajan, Taly, and Yan 2017) and DeepLift (Shrikumar, Greenside, and Kundaje 2017) assign less than 30% (Ibrahim et al. 2019).

Regarding the explanation of TabNet in this thesis, it did not identify "Column" or "Row" as the most salient features in the dataset. However, it did identify position in the sense of "Field" as a strong indicator of the estimation for "Turnover". To be exact, when "Field 4" feature, aka the worst field a game could be positioned, is a very important feature for TabNet (3rd in importance) which is to be expected, as these positions tend to have significantly less "Turnover".

5.3 XGBoost Explanations

A benefit of using gradient boosting is that after the boosted trees are constructed, it is relatively straightforward to retrieve importance scores for each attribute. Generally, importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance. Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function. The feature importances are then averaged across all of the decision trees within the model. Pedro Carmona, Aladdin Dwekat and Zeena Mardawi have completed a study that opens the "black boxes" of machine learning algorithms and fills the literature gap by showing how it is possible to fit a very precise Machine Learning model that is highly interpretable, by using Extreme Gradient Boosting (XGBoost), and applying new model interpretability improvements.

In this thesis, XGBoost was used as a guide for feature engineering with its Global interpretability capabilities. Global measures help understand inputs and their overall modelled relationship with the prediction target - in some cases global interpretations can be highly approximate. lobal interpretability shows the machine-learned relationships between the output or response variable and the input variables or features across the whole data. Global explanations are about understanding how a model makes predictions based on a complete view of its features and how they impact the underlying model structure. They answer questions regarding which features are more important, how they affect the response variable, and what types of possible interactions arise (Boehmke and Greenwell, 2020).

Feature importance in XGBoost was very different before and after the feature engineering. As mentioned in the beginning of this thesis, the dataset was unbalanced with many occurrences of specific games and very few occurrences of others. That resulted in discrimination regarding the theme of the game which influenced the algorithm to favor specific games and thus considering "Themes" as the important feature for making a decision on estimating "Turnover". However, with the approach of muting these game characteristics we managed to discover the importance of position. XGBoost explanations verify those findings with the graph below.

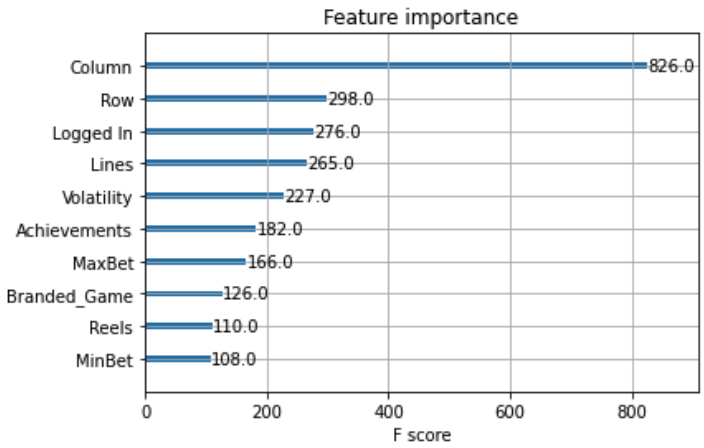

Figure 5.5 XGBoost feature importance

As we can see, features “Column” and “Row” which are the deciding features on game location, where first and second most important features for XGBoost.

5.4 Findings

Being able to interpret the outcome of an algorithm is key for this approach, as interpretable machine learning means that the decision that this algorithm makes can be translated into a domain that is understandable. This was of great importance to our implementation, as we did not need just a good estimator, but we wanted a good estimator that valued position as much as this one did. Shapley values and feature importance metrics validated that our algorithm was indeed making decisions heavily influenced by position.

XGBoost had “Column” feature as the most important factor when estimating “Turnover”, followed by “Row”. This is exactly in line with the business model as well, as the column of the game has the most important role in good positioning.

Because of that, this implementation can move to step three of the approach with counterfactual explanations and propose ideal positioning for maximizing turnover. The above could not properly work if the algorithm did not rely on position as much as it did.

6. Recommendation System

The final step of this system is to recommend a position for a game to maximize its performance. However, in this thesis there was not a recommendation system implemented. This thesis uses the predictor from the first step and counterfactual explanations to recommend the ideal position given a performance goal. To set the ground, first we need to better understand counterfactuals.

6.1 Counterfactual Explanations

Counterfactual explanations (CFEs) are an emerging technique under the umbrella of interpretability of machine learning models (Sahil Verma, John P. Dickerson, Keegan Hines, 2021). They provide "what if" feedback of the form "if an input datapoint were x' instead of x, then an ML model's output would be y' instead of y.". Counterfactual explanations are not widely adopted as a technique due to their challenges in operationalizing. In order for a system to operate well with counterfactual explanations it has to oblige to some general rules. The model that is being interpreted has to be a structural causal model, it has to be static and the data that is being examined needs to be interventional. Elaborating on the above terms will help in understanding our task and whether counterfactual explanations can wisely be deployed.
A structural causal model (SCM) gives constraints on one feature (unary) or between multiple features (n-ary) of the dataset in the form of equations and is therefore dataset specific. Most datasets do not have a readily available, making is difficult to generate CFEs that are actionable by individuals. A complete SCM is almost infeasible to have, but even partial SCMs are sufficient for generating meaningful CFEs. The dataset in this thesis enables us to generate meaningful CFEs as partial SCM.

Most CFE generation approaches assume that the underlying ML model is stationary, and this must be considered when providing actionable CFE. The nature of the system that is being built for this thesis, has a static ML model and therefore also enables us to generate CFEs.
To elaborate more on CFE we consider the following scenario: Score p was returned because variables V had values (v1, v2, ...) associated with them. If V instead had values (v1', v2', ...), and all other variables had remained constant, score p' would have been returned. While many such explanations are possible, an ideal counterfactual explanation would alter values as little as possible and represent a closest world under which score p' is returned instead of p. The

notion of a “closest possible world” is implicit as stated by Sandra Wachter, Brent Mittelstadt and Chris Russell and this thesis has implemented exactly that.
The techniques used to generate counterfactual explanations on deep networks are already widely studied in the machine learning literature under the name “Adversarial Perturbations.” (Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy, 2015). Algorithms capable of computing counterfactuals are used to confuse existing classifiers and regressors by generating a synthetic data point close to an existing one such that the new synthetic data point is either classified differently or has a different outcome. The above shows a tremendous strength of counterfactuals and that is that they can be efficiently and effectively computed by applying standard techniques, even to cutting-edge architectures.

Principally, counterfactuals bypass the substantial challenge of explaining the internal workings of complex machine learning systems. (Jenna Burrell, 2016) Even if technically feasible, such explanations may be of little practical value to data subjects. In contrast, counterfactuals provide information to the data subject that is both easily digestible and practically useful for understanding the reasons for a decision, challenging them, and altering future behavior for a better result.

Counterfactual explanations could be implemented in several ways. The transience of decision-making models suggests that counterfactuals either need to be computed automatically at the time a decision is made or computed at a later time based on an archived copy of the model. As multiple outcomes based on changes to multiple variables may be possible, a diverse set of counterfactual explanations should be provided, corresponding to different choices of nearby possible worlds for which the counterfactual holds.

6.2 CF Implementation - DiCE

For this third step of the approach, a python library called DiCE was utilized. DiCE implements counterfactual (CF) explanations that provide information by showing feature-perturbed versions of the same game in different positions that would have gained more “Turnover”, e.g., The game “F.F.” would have gained 30% more turnover if its position was $Column = 3$, and $Row = 4$. In other words, it provides “what-if” explanations for model output and can be a useful complement to other explanation methods, both for end-users and model developers who want to further utilize this data to suggest the ideal positioning.

The core idea of this setup is finding such explanations as an optimization problem, similar to finding adversarial examples. The critical difference is that for explanations, we need perturbations that change the output of a machine learning model but are also diverse and feasible to change. Therefore, we are generating a set of counterfactual explanations with tunable parameters for diversity and proximity of the explanations to the original input. For this case, we cannot change physical characteristics or occurrence traits of a game, but we can change column and row of a game. With counterfactual explanations and DiCE we implemented simple constraints on features to ensure feasibility of the generated counterfactual examples (only feature “Column” and “Row” can be altered).

In order to execute counterfactuals generation, we feed the data alongside with the model from step one to DiCE and set the parameters for counterfactuals. Those parameters are:

- Total different counterfactuals to be created
- Desired Class
- Proximity Weight
- Diversity Weight
- Feature Weights
- Features to vary
- Permitted feature range

6.3 Findings

With all those parameters above we manage to create as many counterfactual explanations as we need with all the constraints that were explained above. In this report you will find some CFs to showcase the implementation.

Query instance (original outcome : 21859)

	Location_Weighting	Weighted_Days	Column	Row	Active
0	0.0023	0.1668	14.0	14.0	1.0

1 rows × 173 columns

Diverse Counterfactual set (new outcome: [25000.0, 30000.0])

	Location_Weighting	Weighted_Days	Column	Row	Active
0	0.0023	0.16680001	1.0	10.0	-
0	0.0023	0.16680001	2.0	9.0	-

2 rows × 173 columns

Figure 6.1: Generation of counterfactual explanations scenario 1

In this situation, a game located at $Column = 14\ and\ Row = 14$ with a turnover gain of $21859\$$ saw a $50\% - 100\%$ turnover gain with those two scenarios. Scenario one suggested $Column = 1, Row = 10$ while scenario two suggested $Column = 2, Row = 9$.

Query instance (original outcome : 47750)

	Location_Weighting	Weighted_Days	Column	Row	Active
0	0.0024	0.1695	8.0	14.0	1.0

1 rows × 173 columns

Diverse Counterfactual set (new outcome: [70000.0, 90000.0])

	Location_Weighting	Weighted_Days	Column	Row	Active
0	0.0012	0.42289999	1.0	2.0	-
0	0.0012	0.42289999	12.0	5.0	0.0

2 rows × 173 columns

Figure 6.2: Generation of counterfactual explanations scenario 2

In this situation, a game located at $Column = 8\ and\ Row = 14$ with a turnover gain of $47750\$$ saw a $20\% - 40\%$ turnover gain with those two scenarios. Scenario one suggested $Column = 1, Row = 2$ while scenario two suggested $Column = 12, Row = 5$.

Query instance (original outcome : 13565)

	Location_Weighting	Weighted_Days	Column	Row	Active
0	0.0021	0.152	1.0	6.0	1.0

1 rows × 173 columns

Diverse Counterfactual set (new outcome: [15000.0, 25000.0]

	Location_Weighting	Weighted_Days	Column	Row	Active
0	0.0021	0.152	7.0	2.0	1.0

1 rows × 173 columns

Figure 6.3: Generation of counterfactual explanations scenario 3

In this situation, a game located at a good position at Column=1 and Row=6 with a turnover gain of 13565 saw an increase of 80% by changing position, but this time, not necessarily to a better position.

The algorithm estimated that this game does not need a good position for better turnover and will do sufficiently good on a worse position. That will allow us to propose ideal positioning for every game since the algorithm does not propose Column and Row 1 as the greedy choice and can in fact create a grid by matching each game's needs with the right position. Counterfactual explanations could propose Column=1 and Row=1 and stop right there. But that would not be an optimization technique or even a feasible one as we have many games that we equally want to maximize their performance. Therefore, the current state of implementation is suitable for the business need of this task and with the combination of more than a few different counterfactual explanations we can have countless combinations to match the maximization of "Turnover".

7. Conclusions

To conclude with, this thesis has considered and developed a strategy that utilizes machine learning models and explanation techniques to recommend the ideal positioning of a game in order to maximize its performance. Overall, the results presented from this approach demonstrate the superior ability that machine learning models have in predicting an outcome given features or alter features that would predict a given outcome.

The predictor from step one of this system managed to estimate the turnover of each game with satisfactory results given the complexity of the problem and the number of records. Shapley values on step two worked as expected and gave valuable insight on the algorithms' decision making. That kind of insight was instrumental for this approach and allowed the system to proceed to step three of this implementation and propose ideal positioning with counterfactual explanations, a technique that even though it is unconventional, it has proven to be effective and computationally efficient.

The fact that given very little data and by utilizing a simple algorithm combined with counterfactual explanations to deploy a recommendation system is very satisfactory and exhibits the value that data-driven methods and machine learning models can provide.

Appendix – The data

Tracking data

Daily Tracking

Column name	Description
DatePageLink	Date and ID combination
ID	Unique identifier of the record
Date	Datetime of the record
LocationID	Unique identifier of the location
SearchStringID	Unique identifier of the search name
LocationWeighting	Variable regarding location variability
WeightedDays	Variable regarding date variability
PageTypeID	Unique identifier of the page type
Name	Name of the game
GameID	Unique identifier of the game
LocationID	Unique identifier of the location
Location	Name of the location from first to last position
Column	Number of the column of the grid
Row	Number of the row of the grid
Active	Boolean stating active or inactive game
PageID	Unique identifier of the page
WeightingTile	Variable stating the size of the tile on the grid

Games

Column name	Description
SearchNameID	Unique identifier of the search name
GameID	Unique identifier of the game
Game	Name of the game
SearchName	Search name of the game
BrandID	Unique identifier of the brand
Brand	Name of the brand
SupplierID	Unique identifier of the supplier

Supplier	Name of the supplier
Gametype	Name of the different game types
SearchStringID	Unique identifier of the search name
BrandedGame	Boolean stating branded or not game
RTPHigh	Upper range of Return to player number
RTPLow	Lower range of Return to player number
Lines	Physical characteristic of the game, number of vertical lines
Reels	Physical characteristic of the game, number of horizontal lines
Freegame	Boolean stating if there is a free game offered
Achievements	Boolean stating if there are achievements
MinBet	Number of the minimum bet of the game
MaxBet	Number of the maximum bet of the game
HitRate	Physical characteristic of the game
Volatility	Characteristic of the game, higher volatility equals more returns and risk
GametypeID	Unique identifier of the game type.

Themes

Column name	Description
Theme ID	Unique identifier of the theme
Theme	Name of the theme
GameID	Unique identifier of the game

Location Grid

Column name	Description
LocationID	Unique identifier of the location
Location	Number of the location – 1 is the first position at tile (1,1)
Column	Number of the column
Row	Number of the row
Active	Boolean stating active or inactive position
PageID	Unique identifier of the page
Weighting_Tile	Variable stating the size of the tile on the grid

Pages

Column name	Description
Page	Name of the page
Site	Name of the site
Country	Name of the country
Continent	Name of the continent
Page type	Name of the page type
Page ID	Unique identifier of the page
URL	URL of the page
Mobile / Desktop	String stating the origin of the user in the page
Country ID	Unique identifier of the country
Manufacturer ID	Unique identifier of the manufacturer
Site group ID	Unique identifier of the site group
Active	Boolean stating active or inactive page
Regulated	Boolean stating regulated or not page

Performance

Column name	Description
BeginTime	Timestamp of each record
GameGroupID	Unique identifier of the game group
NumBets	Number of bets on a given day for a given game
Turnover	Sum of the turnover of a given game on a given day
GGR	Difference between losses and winnings for the game on a given day
Distinct players	Number of unique players that gambled on a game on a given day
GameGroupName	Name of the game
Provider party	1st, 2nd or 3rd party games
GameProviderName	Name of the provider of the game

Bibliography

Boehmke, B. and Greenwell, B., 2019. Hands-on machine learning with R. Chapman and Hall/CRC.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository. URL http://archive.ics.uci.edu/ml. Accessed: 2019-11- 10.

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy, Explaining and Harnessing Adversarial Examples, ArXiv, Mar. 20, 2015, at 1, https://arxiv.org/pdf/1412. 6572.pdf [https://perma.cc/64BR-WVE7]

Ibrahim, M.; Louie, M.; Modarres, C.; and Paisley, J. W. 2019. Global Explanations of Neural Networks: Mapping the Landscape of Predictions. arxiv:1902.02384 .

James, G., Witten, D., Hastie, T. & Tibshirani, R., 2013. Resampling Methods. In: An Introduction to Statistical Learning with Applications in R. New York: Springer, pp. 176- 184.

J. Friedman 2001. Greedy function approximation: a gradient boosting machine. Annals of Statistics, 29(5):1189–1232

Jenna Burrell, How the Machine 'Thinks': Understanding Opacity in Machine Learning Algorithms, 3 BIG DATA & SOC., Jan. 5, 2016, at 5; Kroll et al., supra note 4, at 638.

Martins, A. F. T.; and Astudillo, R. F. 2016. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. arXiv:1602.02068

P. Li. Robust 2010. Logitboost and adaptive base class (ABC) Logitboost. In Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI'10), pages 302–311

Pedro Carmona, Aladdin Dwekat, Zeena Mardawi, 2020. No more black boxes! Explaining the predictions of a machine learning XGBoost classifier algorithm in business failure. https://doi.org/10.1016/j.ribaf.2022.101649

Ribeiro, M.; Singh, S.; and Guestrin, C. 2016. Why Should I Trust You? Explaining the Predictions of Any Classifier. In KDD.

Ribeiro, M., Singh, S., & Guestrin, C. (2018). Anchors: High-Precision Model-Agnostic Explanations. AAAI.

Sahil Verma, John P. Dickerson, Keegan Hines, 2021. Counterfactual Explanations for Machine Learning arXiv: 2106.07756

Sandra Wachter, Brent Mittelstadt, & Chris Russell 2018, Counterfactual Explanations Without Opening the black box: Automated decisions and the GDPR, Harvard Journal of Law & Technology Volume 31, Number 2

Sercan O. Arik, Tomas Pfister 2019. TabNet: Attentive Interpretable Tabular Learning. arXiv: 1908.07442

Shrikumar, A.; Greenside, P.; and Kundaje, A. 2017. Learning Important Features Through Propagating Activation Differences. arXiv:1704.02685.

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic Attribution for Deep Networks. arXiv:1703.01365.

Tianqi Chen, Carlos Guestrin 2016. XGBoost: A Scalable Tree Boosting System. arXiv: 1603.02754