Predicting the success of a hockey shot

October 11, 2023

Section 1: Introduction

The goal of this report is to use machine learning to predict the success or failure of a hockey shot based on information available before the puck's release. We will not be using box score information such as the final score, because we want to make predictions that could be useful in forming a game plan before or during a game.

Section 2 defines the problem and details the dataset and feature selection. Section 3 details preprocessing and modeling, while section 4 displays results.

Section 2: Problem formulation

Dataset and feature selection

Classifier models will be trained on official play-by-play data recorded by the National Hockey League since the 2000-2001 season (gathered by Martin Ellis using the NHL stats API [1]). The classification of shots will be binary: goal or no-goal. For simplicity, we will assume a regular, non-penalty game situation.

The dataset contains multiple tables of NHL data, containing information on games, game events, players and teams. However, we are only interested in individual shots, therefore we will mostly work with data in the table *game_plays*. The table consists of hundreds of thousands of events recorded in NHL games, such as shots, face-offs and penalties. Its features are detailed in table 1. We will also use the features detailed in table 2 from tables *game* and *game_plays_players*. We will filter the data so that all rows represent shot-type events.

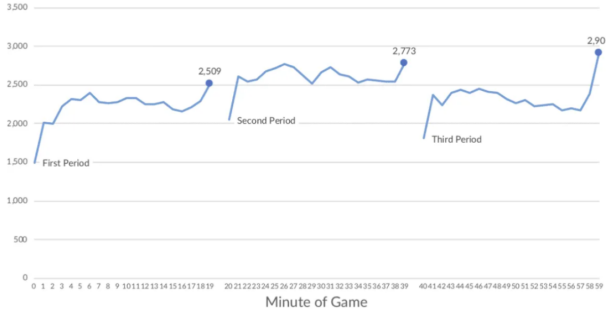
Of these features, only the following are relevant for representing shot-type events: *x, y, period, periodType, periodTime, goals_away, goals_home, playerType* and *type*. The coordinates of the shot and goals scored so far are crucial, but the other features offer valuable information on the context of the shot, too. For example, goals are more common in the final minutes of a game when it is common for the losing team to switch their goalie for a sixth attacker (figure 1.), and a playoff game sees, on average, 4.4% fewer goals than a regular season game[4]. The features *st_x, st_y* and *periodTimeRemaining* provide no extra

Table 1.

Feature	Explanation	Type
play_id	Unique ID for a play	Categorical
game_id	Unique ID for a game	Categorical
team_id_for	Unique ID for the team making the play	Categorical
event	The type of event (e.g. faceoff, shot, tackle)	Categorical
secondary_type	Subtypes for shot-type events (e.g. wrist shot)	Categorical
x	The x coordinate of an event (relative to center court)	Continuous
y	The y coordinate of an event (relative to center court)	Continuous
period	The period during which an event took place	Categorical
periodType	The type of period (regular or overtime)	Categorical
periodTime	Amount of time elapsed in the period	Continuous
periodTimeRemaining	Amount of time left in the period	Continuous
dateTime	Date and time of the event	Continuous
goals_away	Goals scored so far in the game by the away team	Continuous
goals_home	Goals scored so far in the game by the home team	Continuous
description	A textual description of the event	Categorical
st_x	X coordinate multiplied by -1	Continuous
st_y	Y coordinate multiplied by -1	Continuous

Table 2.

Feature	Explanation	Type
type	Type of game (regular season or playoff)	Categorical
home_rink_side_start	Starting side of the home team (left or right)	Categorical
playerType	Type of player (shooter or scorer)	Categorical



information, so they will be ignored. We will use the following features in preprocessing only: *home_rink_side_start, team_id_for, game_id* and *play_id*. Date, player and team data is ignored in the training process to prevent overfitting.

Each shot is labeled as either a goal or a missed shot by the feature *playerType*: our task is an example of supervised learning.

Section 3. Methods

Data preprocessing

First, we must filter the table *game_plays_players* to only include rows (which correspond to plays) with "shooter" or "scorer" as value for *playerType* - this way we narrow the data down to just shot attempts. We then merge *game_plays_players* and the two other tables (on *play_id* and *game_id*).

	team_id_for	team_id_against	x	y	period	periodType	periodTime	goals_away	goals_home	player_id	playerType	type	home_rink_side_start	home_team_id
0	16	4	-71	9	1	REGULAR	54	0	0	8473573	Shooter	R	right	16
1	16	4	-88	5	1	REGULAR	56	0	1	8474141	Scorer	R	right	16
2	4	16	56	-7	1	REGULAR	69	0	1	8474668	Shooter	R	right	16
3	16	4	-37	-24	1	REGULAR	133	0	1	8470281	Shooter	R	right	16
4	4	16	-71	11	1	REGULAR	144	0	1	8477353	Shooter	R	right	16

Especially in the playoffs, the home team has an advantage over the visiting team [2]. We will generate a feature *team_for_is_home*, which indicates whether the shooting player plays for the home team (by comparing *team_id_for* and *home_team_id*. We can now drop *home_team_id, team_id_for, team_id_against* and *player_id* to prevent overfitting. Instead of home goals and away goals, the essential scoring information lies within the amount of goals scored for and against the shooter's team. Therefore, we must replace *goals_away* and *goals_home* with new features *goals_against* and *goals_for*.

	x	y	period	periodType	periodTime	goals_against	goals_for	playerType	type	home_rink_side_start	team_for_is_home
0	-71	9	1	REGULAR	54	0	0	Shooter	R	right	-1
1	-88	5	1	REGULAR	56	1	0	Scorer	R	right	-1
2	56	-7	1	REGULAR	69	0	1	Shooter	R	right	1
3	-37	-24	1	REGULAR	133	1	0	Shooter	R	right	-1
4	-71	11	1	REGULAR	144	0	1	Shooter	R	right	1

The source data expresses x and y coordinates as absolute distance from the center of the rink, this is to say the data does not take into account the changing of sides between periods. This is detrimental to the learning process, because the critical piece of information is the relative distance from the puck to the net. In addition to the ongoing period, we also need to take into account which side the shooter's team starts the game on, which can be deduced from the features *home_rink_side_start* and *team_for_is_home*. We will replace "left" and "right" in *home_rink_side_start* with -1 and 1, and calculate the adjusted coordinates x_1 and y_1 for all rows with the following formula:

$$p = period \,\%\, 2 \cdot 2 - 1$$

$$(x_1, y_1)^T = team\_for\_is\_home \cdot home\_rink\_side\_start \cdot p * (x, y)^T$$

where p is -1 for even and 1 for odd periods. Now the direction of the shot is adjusted to always be positive along the horizontal axis. We can now drop *home_rink_side_start* and rename *playerType* as *scored*. Roughly 2% of the data is missing values for *x* and *y*, so we will filter such rows out. We will also encode all categorical features as unique integers and standardize continuous features, following best practices [3].

	x	y	period	period_type	period_time	goals_against	goals_for	scored	game_type	team_for_is_home
0	0.787699	-0.508773	0	1	-1.54428	-0.985208	-0.97476	0	2	0
1	1.0774	-0.282857	0	1	-1.53857	-0.263839	-0.97476	1	2	0
2	0.532083	-0.395815	0	1	-1.50143	-0.985208	-0.253742	0	2	1
3	0.208302	1.35504	0	1	-1.31858	-0.263839	-0.97476	0	2	0
4	-1.63214	0.620808	0	1	-1.28716	-0.985208	-0.253742	0	2	1

Modeling

Considering that the chosen features contain multiple categorical features, and the size of our training dataset (490000 rows), random forest classification could be a good training method. The method is

also easily provided by the Scikit Learn library.

The loss function used is log-loss, which measures the difference of the predicted probability and the actual target value (0 or 1 in our case). Log-loss is a binary loss function, and our task is binary in nature. Furthermore, offensive strategy in hockey revolves around the success probabilities of shots taken in different situations, making log-loss a natural function to use in our binary classification task.

Another model that utilizes log-loss is the multi-layer perceptron classifier. It is suitable for large, non-linearly separable datasets. The lack of linear separability made another model, logistic regression, useless for our task. The multi-layer perceptron classifier is a powerful model for binary classification tasks, and it, too, is conveniently provided by the Scikit Learn library.

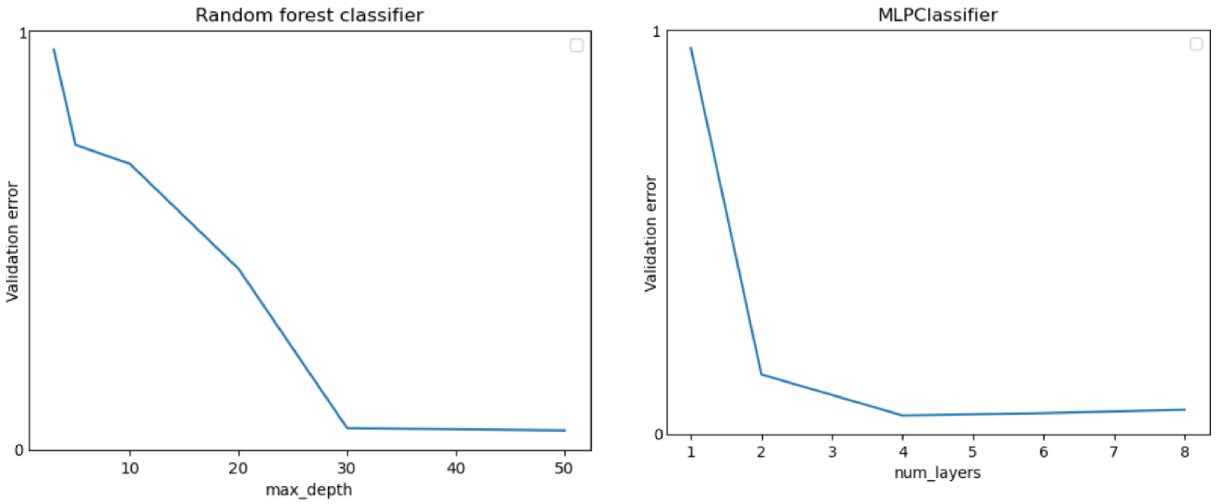
To ensure somewhat practical training times, we will not use the entire dataset. Instead, we will use 700000 rows, which will be divided into a training set, a validation set and a test set using a randomized 70/20/10-split. The split ensures a large enough volume of data to train an accurate model, whilst leaving enough data for precise validation and testing. Both models will be trained and validated with the same training and validation set, respectively. Season averages in scoring fluctuate[5], therefore randomization is essential due to the time-dependent nature of the source data.

4. Results

After training both models with the training data at different max depths and hidden layer counts, these are the validation results. All validation and test errors are measured using the mean squared error function.

The optimal maximum tree depth for the random forest classifier seems to be around 30 based on the “elbow” of the curve. (figure 2). At depth 30, we reach a validation error of 0,0120 and an F1 score of 0,9054.

Figure 2

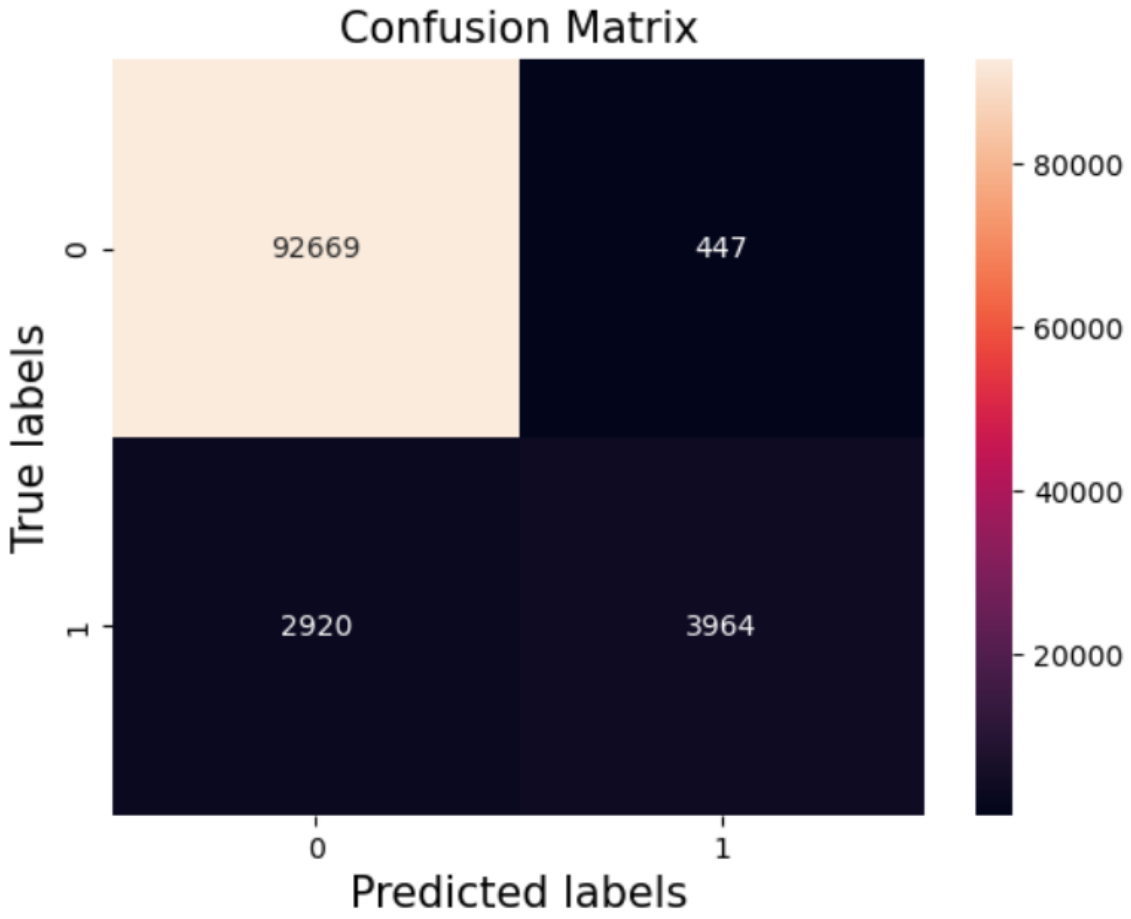


The optimal amount of hidden layers for the multi-layer perceptron seems to be 2, based also on the point of steepest decline (figure 2). At 2 hidden layers, we reach a validation error of 0.05015, which is comparable to that of the random forest classifier, but its F1 rating - which is very important with a highly imbalanced dataset like ours, because it takes precision into account - is significantly lower than that of the random forest classifier, only 0,4721.

Based on these results the random forest classifier at a maximum depth of 30 is the superior model

for our task. Here are some performance metrics taken using the test set.

Figure 4.




```
F1: 0.7019034971226207
Precision: 0.8986624348220358
Accuracy: 0.96633
```

The model recorded a final test error of 0,0294.

We reach a high accuracy, and most importantly, precision. Only around 9 percent of hockey shots end in a goal, and our dataset is therefore highly unbalanced. A naive model predicting always zero will achieve an accuracy of over 90 percent, but terrible precision. Our model has achieved both, showcased by the F1 score.

6. References

[1] Kaggle: "NHL Game Data", Martin Ellis via NHL stats API 2020

[2] "How Significant is Home Advantage in the NHL?", Zachary S Bischof 2021

[3] "What is Feature Scaling & Why is it Important in Machine Learning?", Jason Chong, Towards Data Science 30.12.2020

[4] "NHL Scoring Trends: Regular Season vs. Playoffs", Jeff Donchess, Dratings 4.5.2022

[5] "NHL League Averages", Hockey Reference 2023

[Figure 1] "Last-Minute Hockey Goals", Jonathan Schwabish, PolicyViz 12.9.2019