Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Master's Thesis

Using machine learning in Football

Bc. Thomas Stvarnik

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Bc. Thomas Stvarnik

Informatics

Thesis title

Using machine learning in Football

Objectives of thesis

The main goal of this thesis, is to present and explore the benefits of using machine learning on match event data, to analyze players and their actions on the football field Using XGBoost in python. Supporting goals are to train and explain a XGBoost model that can evaluate the chance of scoring a goal when a player shoots a shot.

Methodology

The methodology of this thesis is based on the analysis of technical and scientific sources focusing on the use of machine learning on real world problems.

Based on the synthesis of the knowledge gained, a XGBoost model will be trained, and a Python script will be created. When given event data to process, the script will evaluate the data using the trained model, and visualize the results in a user friendly way.

The performance of the model will be assessed by comparing the estimated number of goals in a wider range of data, and the actual number of goals scored.

The proposed extent of the thesis

60-80 pages

Keywords

Machine learning, football, data, xgboost, python

Recommended information sources

RAABER, Dominik a MEMMERT, Daniel. (2018). Data Analytics in Football – Positional Data Collection, Modelling and Analysis. DOI: 10.4324/9781351210164.

RUSSELL, Stuart J., Peter NORVIG a Ernest DAVIS. Artificial intelligence: a modern approach. 4th ed. Upper Saddle River: Prentice Hall, 2020. ISBN 1292401133

Expected date of thesis defence

2022/23 SS – FEM

The Diploma Thesis Supervisor

Ing. Petr Hanzlík, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 26. 11. 2022

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 31. 03. 2023

Declaration

I declare that I have worked on my master's thesis titled "Using machine learning in Football" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 31.3.2023 _____

Acknowledgement

I would like to thank my supervisor Ing. Petr Hanzlík Ph.D. for his guidance throughout the whole process of writing and doing research.

Using machine learning in Football

Abstract

This thesis explores the application of machine learning techniques to football data analysis, focusing on the history, use cases, and implementation of these methods in the sport. The study begins with a literature review, tracing the development of data analysis in football from early beginnings with Charles Reep to the modern age, and discussing the role of event data in the sport. The thesis then delves into the history, types, and workflow of machine learning, with emphasis on supervised, unsupervised, semi-supervised, and reinforcement learning. The practical part of the study is dedicated to the implementation of a machine learning model, specifically the training process and utilization of the final selected model to create a shot map. This implementation relies on Python, XGBoost, Pandas, and Matplotlib technologies. The thesis outlines the intents, workspace configuration, and training data features, before describing the model training algorithm and methods for finding the best results. The study offers valuable insights into the potential of machine learning applications in football data analysis, and enhancing understanding of the game.

Keywords: Machine Learning, Data analysis, Event data, Python, xGboost, Football, xG

Using machine learning in Football

Abstrakt

Tato diplomová práce zkoumá aplikaci strojového učení v analýze fotbalových dat se zaměřením na historii, případové studie a implementaci těchto metod ve sportu. Studie začíná rešerší literatury, která sleduje vývoj analýzy dat ve fotbale od raných začátků s Charlesem Reepem až po moderní dobu a diskutuje o roli eventových dat ve sportu. Práce se poté zabývá historií, typy a pracovním postupem strojového učení.
Praktická část studie je věnována implementaci modelu strojového učení, konkrétně procesu trénování a využití vybraného modelu pro vytvoření mapy střel. Tato implementace spoléhá na technologie Python, XGBoost, Pandas a Matplotlib.
Diplomová práce popisuje záměry, konfiguraci pracovního prostoru a vlastnosti tréninkových dat předtím, než popíše algoritmus tréninku modelu a metody pro nalezení nejlepších výsledků. Studie nabízí cenné náhledy do potenciálu aplikací strojového učení v analýze fotbalových dat a zlepšení porozumění hře.

Klíčová slova: Machine Learning, Datová analýza, Eventová data, Python, xGboost, Fotbal, xG

Table of content

1 Introduction

Machine learning is increasingly being used in football to examine and comprehend data related to the sport. The most useful type of data for such purposes is what we call “event data”, which is a collection of tracking information gathered during a match, such as player and ball movement and the actions of the match officials, often gathered using specialized tracking systems installed in the stadium, combined with some human input.

This type of data can then be used to establish players or teams’ performances in certain situations, overall gameplay, or style. For the analysis of this type of data, it is typical to use visualizations, mathematical models, or machine learning.

One specific use case of machine learning in football is predicting the result of shots taken, with the goal being to develop a model that can accurately determine the probability of a shot resulting in a goal. Such information can be very useful in determining the actual performance of a team or player, especially in football, as there is a small number of goals scored over a single match, so it can be hard to judge players or teams solely on their actual goals scored, because they might just have been lucky. This model can tell us which team has managed to create better chances for themselves to win, or which player is getting himself into the best positions to score.

In this thesis we will explore the technologies behind, and the process of creating such a model, and explore some use cases for it.

Objectives and Methodology

1.1 Objectives

The main goal of this thesis, is to present and explore the benefits of using machine learning on match event data, to analyze players and their actions on the football field Using XGBoost in python. Supporting goals are to train and explain a XGBoost model that can evaluate the chance of scoring a goal when a player shoots a shot.

1.2 Methodology

The methodology of this thesis is based on the analysis of technical and scientific sources focusing on the use of machine learning on real world problems.
Based on the synthesis of the knowledge gained, a XGBoost model will be trained, and a Python script will be created. When given event data to process, the script will evaluate the data using the trained model, and visualize the results in a user friendly way.
The performance of the model will be assessed by comparing the estimated number of goals in a wider range of data, and the actual number of goals scored.

2 Literature Review

2.1 History of data analysis in football

2.1.1 Early beginnings - Charles Reep

Charles Reep was an accountant who got inspired after a conversation with the captain of Arsenal about an analysis of wing play being used by the club, which emphasized the objective of wing players to quickly move the ball forward.
This conversation inspired Reep and was the beginning of his passion for attacking football, and the widespread adoption of the play style.

Later, Reep became frustrated with a game he was attending. The game was slow, and the scoring attempts were inefficient.
This was where he decided to write down the actions on the pitch. He recorded pitch positions and passing sequences with outcomes using a system that mixed symbols and notes to obtain a complete record of a game. Basically, the first or one of the first event data recordings.
The idea was to understand the patterns of the team and suggest possible improvements.
With the data collected, he found that an average football game at the time would consist of 280 attacking moves, and an average of 2 goals scored. Using simple extrapolation, he claimed that a small improvement, could get teams to the average of 3 goals scored per match.
Reep established himself as the first performance analyst in professional football, and would continue gathering match data, and developing on his strategies derived from the analysis of the data he collected.
When he first published his work, some of his analysis suggested that teams scored on average every nine shots, and that half the goals scored came from recovered possessions in the final third. He also suggested that teams should try to move the ball as quickly forward as possible, instead of passing in a more passive manner. He was promoting a quick, direct and long-pass playing style.
However not everybody was convinced about his various findings. He has been criticized for his simplistic methods, where his conclusions were often more the result of correlation, not

causation, not taking into consideration the quality of the team, as well as having strong preconceived notions that would prevent him from exploring alternative hypotheses.
(Pollard 2002)

2.1.2 Next generation - Opta

In 1996, Opta consulting started to collect match data for the Premier League, with the goal to rank teams and players. It was a major step forward for the industry, as they had signed a deal with a sponsor of the Premier League, to provide their data to each team in the league, as well as to the media.
The clubs received Excel sheets with reports on some basic statistics and information.
Teams now got an overview about how many kilometers each player ran, and how many tackles and passes they made.
But what got people thinking about the use of data in football the most, was arguably their index, called “Opta Index”, which rated players across 70 categories to determine the best players for each position.
(*The history of football analytics - Part 2.* [no date])

2.1.3 Modern age

In recent times, the realm of professional football has experienced a considerable shift due to the emergence of data analytics. This has resulted in an increasing number of teams integrating data-driven decision-making processes into their operations. Nowadays, it's uncommon to come across a professional soccer team that doesn't have a dedicated data analytics team or collaborate with an external company for such services.

The application of data analytics in football has become a crucial component in attaining success. By harnessing the extensive data gathered from matches, teams can extract valuable insights about their performance, pinpoint areas that need enhancement, and make well-informed choices regarding their tactics, training, and recruitment strategies.

Looking at the industry nowadays, the most notable companies are Opta, Wyscout, InStat, and StatsBomb.

- **Opta** - Opta, now owned by StatsPerform, has expanded beyond football, and is now involved in a variety of sports, including but not limited to Hockey, Rugby, Baseball, Basketball, and American football. Opta is tracking event data for the Premier league, EFL, and Scottish leagues.
- **Wyscout** - A company under the Hudl umbrella, is tracking and providing event data and videos for over 600 leagues worldwide, and 400 000 players, as well as providing their own platform where users can see videos, and simple statistics from matches.
- **InStat** - Recently acquired by Hudl, is providing data and videos for Hockey, Football and Basketball, and is tracking over 150 000 players when it comes to football itself.
- **StatsBomb** - Well known for advanced statistics, and a web platform with a lot of advanced features and metrics. The company tracks over 90 leagues worldwide.

Event data can be very useful, and it keeps getting better. One of the more recent improvements in this field is tracking data.
Regular event data usually provides us with the information about the action that is happening with the ball. This means, all the other players on the pitch are not accounted for. This is where tracking data comes in. Tracking data is keeping track of other players on the pitch. Some tracking data is tracking every player on the pitch, some tracking data only includes a certain area around the ball. Anyhow this can be very important information, as having the ball in the penalty area, and having only the goalkeeper separate our player from the goal, can be very different from having a couple enemy players right in front of him.
Expected Goals model (xG)
The xG model, or Expected Goals model, is a widely used statistical tool in the realm of football analytics, and in recent years became popularized to the wider public by the media outlets covering football events, as they slowly begin to integrate it into their match statistics.

The model's primary purpose is to quantify the probability of a shot resulting in a goal, based on various factors such as shot location, angle, body part used, and the game situation. By providing a numerical value for each shot's likelihood of becoming a goal, the xG model offers a deeper understanding of a team's performance and the quality of their goal-scoring opportunities.

Many different variations of it exist across the industry.

Many companies and even some enthusiasts have developed their own xG model.

Here are a couple of examples from two of the bigger companies in the industry:

Wyscout:

According to Wyscout, their xG model uses the following features.

- Location of the shot
- Location of the assist
- Foot or head
- Assist type
- Was there a dribble of a field player or a goalkeeper immediately before the shot?
- Is it coming from a set piece?
- Was the shot a counterattack or did it happen in a transition?
- Tagger's assessment of the danger of the shot

Plus, apparently some “technical parameters”, which we have no way of identifying.(*Wyscout xG* [no date])

We can see that the general idea here is to assess where the shot took place, and what happened immediately before the shot.

StatsBomb:

StatsBomb takes this a step further, with their ability to include different kinds of tracking data. Just to name a few interesting upgrades to their model when compared to most other models, they have been able to include:

- Positioning of the goalkeeper
- Positioning of the surrounding defenders and attackers
- Height of the ball at the moment the shot is struck
- Shot velocity

(*Upgrading Expected Goals - StatsBomb | Data Champions* [no date])

Here is a visual representation of what the model gets fed, when it comes to positioning of the players around the shooter.

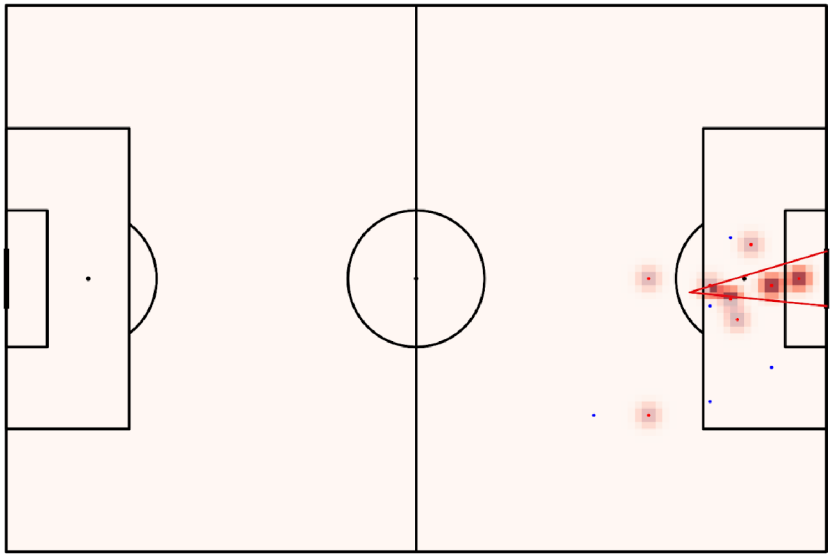

Figure 1: Tracking data Visualization (StatsBomb)

Use cases

One of the main use cases of the xG model is evaluating a team's offensive and defensive capabilities. By analyzing the total xG value for shots taken and conceded, we can get a clearer picture of a team's effectiveness in creating goal-scoring chances and preventing the opposition from doing the same. This insight can help coaching staff in adjusting tactics and addressing areas of concern.

Another notable application of the xG model is in player analysis. By examining a player's xG metrics, analysts can assess their performance in terms of goal-scoring potential, shot selection, and positioning. This data-driven approach can assist in identifying talent,

evaluating transfer targets, and making informed decisions on player development and recruitment.

Finally, the xG model has found its place in the world of sports betting, as bettors utilize xG data to make more informed wagers. By understanding the underlying performance metrics of teams and players, bettors can potentially identify value bets and make data-driven decisions.

In conclusion, the xG model has become an indispensable tool in football analytics, offering valuable insights into team and player performance. Its diverse range of use cases, from tactical analysis to player scouting and betting, highlights the importance of data-driven decision-making in modern football.

2.2 Use cases of event data in football

Delving into the realm of football analytics, the application of event data has become increasingly prevalent, enabling teams and analysts to draw meaningful insights and inform strategic decisions. Event data, which encompasses information related to various on-pitch occurrences, such as passes, shots, and tackles, serves as a valuable tool for dissecting the intricacies of the game.

With the growing availability of granular event data, analysts are now able to dissect individual and team performances with remarkable precision. By utilizing cutting-edge visualization techniques, often powered by libraries like matplotlib, these professionals can transform raw data into insightful visual representations. These graphics offer an accessible means for identifying patterns, understanding tactics, and evaluating player contributions within specific contexts.

One popular use of event data is to examine a team's passing network, which can reveal the underlying structure of a team's tactical approach. By plotting the average positions of players and mapping the frequency of passes between them, analysts can identify key playmakers, assess the balance of the formation, and uncover potential weaknesses to exploit.

Another notable application of event data is the creation of shot maps, which provide a spatial representation of all the shots taken during a match or over a specific period. These visualizations can help analysts pinpoint areas from which a team is most dangerous, as well as identify defensive vulnerabilities that may warrant attention.
Moreover, the versatility of event data lends itself to the analysis of various defensive metrics, such as pressing intensity and tackle success rates. By visualizing these aspects, analysts can gain a deeper understanding of a team's defensive structure and efficacy, informing potential adjustments to tactics or player selection.

In essence, the utilization of event data in football has become an essential component of modern analytics. By harnessing the power of visualization tools like matplotlib, analysts can unlock valuable insights into team dynamics, tactics, and player performance, ultimately influencing critical decisions both on and off the pitch.

2.3 Types of machine learning

Depending on the methodology and learning approach, Machine Learning can be divided into four categories:

- Supervised Machine Learning
- Unsupervised Machine Learning
- Semi-Supervised Machine Learning
- Reinforcement Machine Learning

(*Types of Machine Learning - Javatpoint* [no date])

2.3.1 Supervised Machine Learning

As the name says, Supervised Machine Learning depends on supervision. It is implemented with the use of labeled data which contains tons of examples of Target and Features. Supervised Machine Learning uses algorithms to detect relationships and patterns of the input and output data. The process of finding relationships between Features and Target from the dataset is called Fitting or Training.

Supervised Machine Learning solves two types of problems:

- Classification - prediction of categories present in the dataset.
- Regression - prediction of continuous output variables.

Advantages of Supervised Machine Learning:

- Since it works with a labeled dataset, we have an idea about the classes predicted.
- These algorithms are good at predictions based on previous experience.

Disadvantages:

- Not fitted for complex problems.
- Takes a long time to train.
- Predictions can be wrong if the testing data differ from the training data.

Applications of Supervised Learning:

- Medical Diagnosis
- Image Segmentation
- Spam Detection
- Fraud Detection
- Speech Recognition

2.3.2 Unsupervised Machine Learning

Unlike Supervised Learning, Unsupervised needs no supervision. The idea is that the model predicts output on unlabeled data.
The main goal is to group data by similarities, patterns, and differences. Though it still requires some level of human involvement, analyzing that output makes sense.

Unsupervised Learning can be of two types:

- Clustering: groups objects into "clusters" based on similarities
- Association: associates data by dependency on one another

Advantages of Unsupervised Learning:

- Because of the unlabeled dataset, it can work with more complex tasks.
- It is easier to obtain unlabeled datasets.

Disadvantages:

- Since no exact outputs are given, outputs can be less accurate.

Applications of Unsupervised Learning:

- Recommendation engines
- Network Analysis
- Anomaly Detection

2.3.3 Semi-Supervised Machine Learning

Semi-Supervised Learning is a golden mean between Supervised and Unsupervised. While training, it uses both labeled and unlabeled data. It blends a small size of labeled data with a big size of unlabeled.

Semi-Supervised Learning can be of two types:

- Transductive Learning: focuses on improving the accuracy of prediction for specific instances in data.
- Inductive Learning: focuses on generalizing the mapping between input and output data to new, unseen data.

Advantages of Semi-Supervised Learning:

- Efficiency
- Easy to understand
- Solves the issues of Supervised and Unsupervised Learning

Disadvantages:

- Iterations are unstable.
- Low accuracy.

Applications:

- Image Classification
- Natural Language Processing
- Computer Vision
- Anomaly Detection

2.3.4 Reinforcement Machine Learning

Reinforcement Machine Learning is based on the feedback process, where an agent learns from its own experience and gets rewarded for good actions and punished for bad. The feedback is either a reward or a punishment, and the goal is to maximize the reward. It is similar in a way to a human who learns from their experience.

Types of Reinforcement Learning:

- Positive Reinforcement Learning: when an agent is rewarded for taking a particular action, which encourages it to take similar action in the future.
- Negative Reinforcement Learning: when an agent is punished for taking a particular action, which makes it change an approach to avoid punishment in the future.

Advantages of Reinforcement Learning:

- It is good at solving complex problems, which other techniques could not do.
- Because of its similarity to a human, it tends to be more accurate.

Disadvantages:

- Not good to use for simple problems.
- Needs a lot of computation and data.

Applications of Reinforcement Learning:

- Robotics
- Game AI
- Healthcare
- Finance
- Recourse Management
- Personalized Recommendations
- Autonomous Vehicles.

2.4 Machine learning workflow

Machine learning techniques are used for classification work by training a model on a labeled dataset when later given unlabeled data to label it. The selection of the technique fully depends on the problem there is to solve, and the data given, if it is structured and labeled or not. The more data is given, the better the prediction will be.
It goes as follows:

- **Data Preparation Stage**: this step is first and is highly important since good data preparation will lead to more accurate and efficient algorithms. At this stage, all the data must be collected. After, the data must be labeled with class labels and cleaned so there is no inconsistency.

- **Feature Selection Stage**: this step is a process of reducing and selecting the right input variables for our predictive model. The importance of reducing it is not only to decrease the computational cost but also to, sometimes, improve the model's performance. It is done using statistical methods which evaluate the relationships between input and target variables, after choosing the ones with the strongest connection. These methods will be discussed later.

- **Model Selection Stage**: at this stage, the most fitting for the task ML model is chosen.

- **Hyperparameter Tuning Stage**: the performance of the model is highly dependent on the choice of the hyperparameters. Hyperparameters are adjustable and are always set before the actual training. It is a time-consuming and expensive process of finding optimal hyperparameters, mostly beneficial for more complex models.

- **Model Training Stage**: at this point, the right learning algorithm is fed with the prepared data, to learn the patterns and relationships. The goal is to minimize the difference between the predicted and actual output.

- **Model Evaluation Stage**: after training, evaluation is needed to estimate how well it performs. it is done by:

1. Training and evaluating on the same dataset: mostly beneficial for complex models that have issues with generalizing.
2. Split test: it is a fast, flexible, and simple way to test the model. The dataset is split into two parts so that the model is trained and tested on the different data.
3. Cross-validation techniques: using cross-validation techniques prevents having a "lucky" or "unlucky" split of the data. The results are averaged across multiple training sessions and overall performance is evaluated.

Accuracy, precision, and recall are some of the common metrics used for evaluation.

- **Model Deployment Stage**: after all the training and evaluation, the model can be deployed to predict unseen data points.

2.4.1 Data preparation

Why Prepare Data?

To be processed and analyzed, raw data must be reformatted, cleaned, and put into one file or data table.

It is usually a long process but is essential to prevent the possibility of biases from poor data quality.

Data must be formatted according to the software tool which will be used.

Real-world data is dirty.

Data can be incomplete, have inconsistencies, or even errors that need to be looked upon.

Benefits of Preparing Data

- **Fix Errors**: during the Data preparation, errors can be caught and fixed, otherwise later it is harder to do, once it's out of its source.

- **High-quality of Data**: when going through all the preparation stages you are sure that data is of high quality.
- **Better business decisions**: having high-quality data leads to a faster and more efficient analysis which equals better, high-quality business decisions.

Main Data Preparation Steps

Figure 2: Data preparation steps (DeZyre)

- **Data Discretization** - particularly important for numerical data. Is a way of transforming continuous data into finite with minimal loss. For example, data binarization transforms continuous attributes into binary.

- **Data Cleaning** - at this stage duplicates get removed, structural errors get fixed, and missing data and incorrect format are handled. Altair Monarch is a helpful technology to speed up the cleansing process with more than 80 pre-built data preparation functions.

- **Data Integration** - is the process of combining data from different sources and providing a unified overview of it.

- **Data Transformation** - is a process of transforming data into a usable format to match the destination software.

- **Data Reduction** - typically means the reduction of the volume of data. It is beneficial for storage efficiency, improves performance, and reduces cost. Common technologies used are Compression and Data Deduplication.

Data binning

Binning data helps to simplify the visualization of the data. It is usually a compression of data into smaller "bins" by reducing the number of possible levels or values. For example, grouping the age of people in the ranges by decades. It helps avoid overfitting for smaller datasets.

2.4.2 Feature selection

Why is Feature Selection important?

- **Overfitting or The Curse of Dimensionality** - this problem arises when the data we are working with has multiple features (columns), so-called high-dimension data. The more features data has - the higher are chances for overfitting.
- **Occam's razor** - simplicity is key. Ocamm's razor is a method of choosing between theories or assumptions. Normally, the simplest is the best.
- **Garbage in - Garbage out** - some features sometimes only take up space and don't provide any important information, therefore making the dataset bulkier. It is best to get rid of such.

How to select the best features?

Fortunately, nowadays, there are technologies to use for a simpler feature selection such as Scikit-learn. Most of the available feature selection methods can be divided into three categories:

- **Filter-based**: filter features are based on some metric. For example, Chi-Square or Correlation;

- **Wrapper-based**: this approach takes a set of features as a problem. Example, Recursive Feature Elimination;
- **Embedded**: This category uses algorithms that have pre-built selection features, such as Lasso.

Feature Selection Methods:

- **Pearson Correlation** - is a filter-based method. This method takes the features with the highest correlation to a target variable as these are usually more informative and do best for a predictive model. Though the method can only identify linear relationships, it may not be as good at identifying interactions between features and nonlinear relationships.

- **Chi-Squared** - is also filter-based. This method is generally used for measuring the independence of the variables, therefore it can tell us which of them are most likely to have these relationships. It is important to acknowledge that the Chi-Squared test assumes that variables are categorical, and samples are independent, so in the case of continuous variables and relations between samples - other methods are better to consider.

- **Recursive Feature Elimination** - it is a wrapper-based method. Scikit-learn (ML library for Python) describes it as: "... the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.". (*sklearn.feature_selection.RFE — scikit-learn 1.2.2 documentation* [no date])This method might be expensive but is very useful to reduce the dimensionality of the dataset and to get rid of irrelevant features.

- **Lasso (Least Absolute Shrinkage and Selection Operator)** - it is an embedded method. Lasso is a kind of regularized regression that penalizes the objective function, therefore some coefficients of features are forced to be shrunk to zero which equals being removed from the model. Lasso assumes that the relationship between the target variable and features is linear and might not be the best choice for non-linear relationships.

- **Tree-based** - also called the decision tree method, it is an embedded method. This method uses a tree-like model of decisions and their possible consequences to resolve the problem. The tree is built in a way of recursive splitting of the dataset into smaller ones by the values of the features until the needed criteria are met. The outcome is the tree where each "branch" is a decision based on the values of the features, and every "leaf" is a prediction. In comparison to a Random Forest, for example, the Tree-based method is well interpretable and provides intuitive explanations for decisions. Also, this method is not as expensive computationally as Random forest.

- **Random Forest** - is an embedded method. It is related to a tree-based method, though the approach is a bit different. Unlike the Tree-based method where only one decision tree is built, Random Forest builds multiple decision trees and combines the predictions to improve the accuracy of the model. The idea is to add some Randomness into a tree-building process by implementing the random selection of the set of features and a set of data points for each tree. This makes a model less prone to overfitting, as well as reduces the correlation between trees.

- **Gradient Boosting Machines** - it is an embedded method. This approach also uses a set of decision trees to resolve the problem. GBM produces the importance of a feature by adding up the number of times the split of the node has happened in the decision tree, weighted by improvement in the error function caused by the split. Although GBM may not find the optimal set of features, it is good for reducing dimensionality and improving accuracy.

2.4.3 Model selection

Model Selection is an important step in training a model. There are several factors to consider when choosing the best one, such as the complexity of the task, the size of the dataset, available computing resources, and desired performance.
The common approach to selection is to compare the performance of different models using the training, validation, and test datasets. The best to fit is the best to choose.
When selecting a model, it is important to consider the interpretability of the model. Some are easier and some are harder to interpret. The easiest is Decision Trees and Linear Regression while Neural Networks are usually hard to interpret.

Here are some of the common ML models:

- Linear Regression
- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVMs)
- Neural Networks

Linear regression

Linear regression is probably one of the most common and well-understood models in ML. It is mostly used for regression tasks. The LR algorithm belongs to supervised learning and works by applying relations that predict the outcome of the event based on independent variable data points. The result is a straight line that is surrounded by the given data points, as close to it as possible.

Generally, Linear Regression represents a linear relationship between dependent and independent variables. It is done by observing how the value of the dependent variable changes with the change of the independent variable.

An equation for LR goes as followed:

$y= \beta 0+ \beta 1x+ \varepsilon$

Where:

- Y= Dependent Variable
- X= Independent Variable
- β 0= intercept of the line
- $\beta 1$ = Linear regression coefficient (slope of the line)
- ε = random error

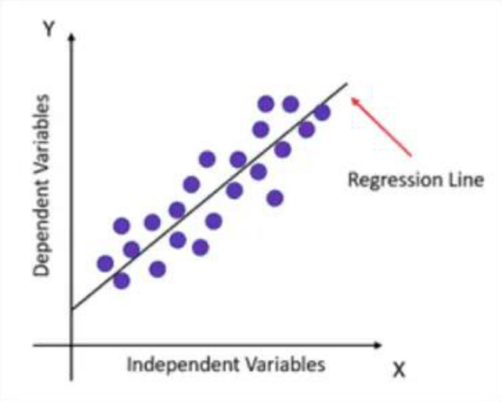

Figure 3: Linear regression graph (Knowledge Hut)

Types of Linear Regression:

- **Simple Linear Regression** - works with one independent variable
- **Multiple Linear Regression** - number of independent variables differs from 1
- **Non-Linear Regression** - when the most fitting line is not straight but curved.

Logistic regression

Logistic Regression also falls under supervised learning. It uses a set of independent variables for predicting the categorical dependent variable and is mostly used for classification problems.

Since output must be categorical it can be either 1 or 0, Yes or No, True or False, etc. But instead of giving an exact categorical result, it gives us a probabilistic value that lies between 1 and 0 which describes the likelihood of something.

Instead of fitting a regression line, in Logistic regression, we are fitting an "S" shaped function(Sigmoid function) with maximum values of 0 and 1.

The dependent variable must be naturally categorical for it to work.

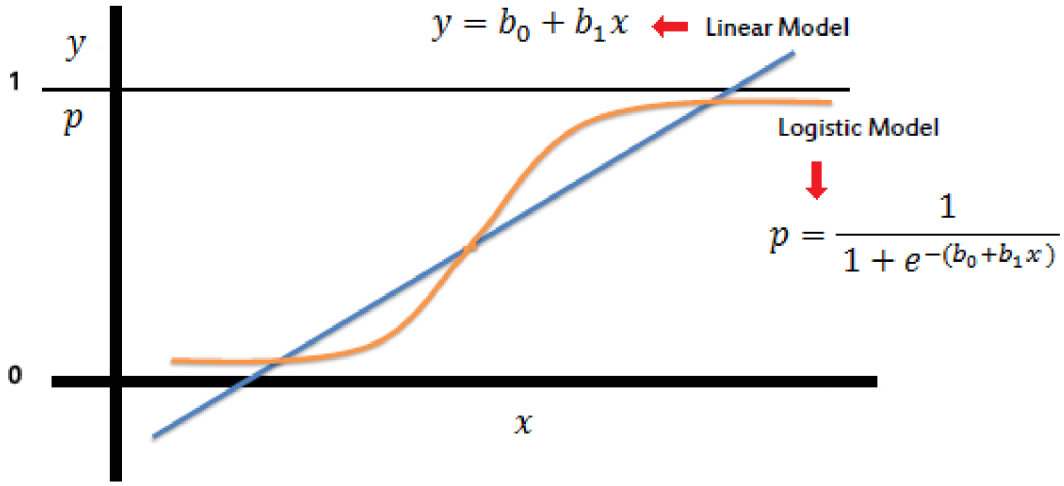

Figure 4: Logistic Regression (saedsayad)

Types of Logistic Regression:

- **Binomial**: dependent variable can have only two values such as 0 and 1, yes or no, pass or fail, etc.

- **Multinomial**: dependent variable can be of 3 or more unordered types. For example, "plastic", "paper", and "glass".

- **Ordinal**: dependent variable can be of 3 or more ordered types, like "short", "average", and "tall".

Decision trees

Decision Trees fall into the supervised learning category and can be used for both classification and regression, but if preferred - they are better suited for classification. It is called a tree because of its structure, it starts with the root node, expands with more branches, and grows into a tree. In Decision Trees, internal nodes represent the features, leaf nodes represent outcomes, and branches - decisions.

A Decision Tree is a graphical/visual representation of all the possible solutions to a given problem, considering all the conditions.

There are two types of nodes:

- Leaf Node: the outcome of the decision, the end node not followed by any other branches.
- Decision Node: used to decide, followed by multiple branches.

In order to build a tree a CART (Classification and Regression Tree) algorithm is used. It is built on the basis of Gini's impurity index to originate binary splits.
Gini's impurity index describes the probability of a variable being classified wrongly when chosen randomly.
The idea of a tree is to simply ask a question like Yes or No, then divide it into sub nodes or outcomes. Also, it accepts not only categorical data but numerical as well.

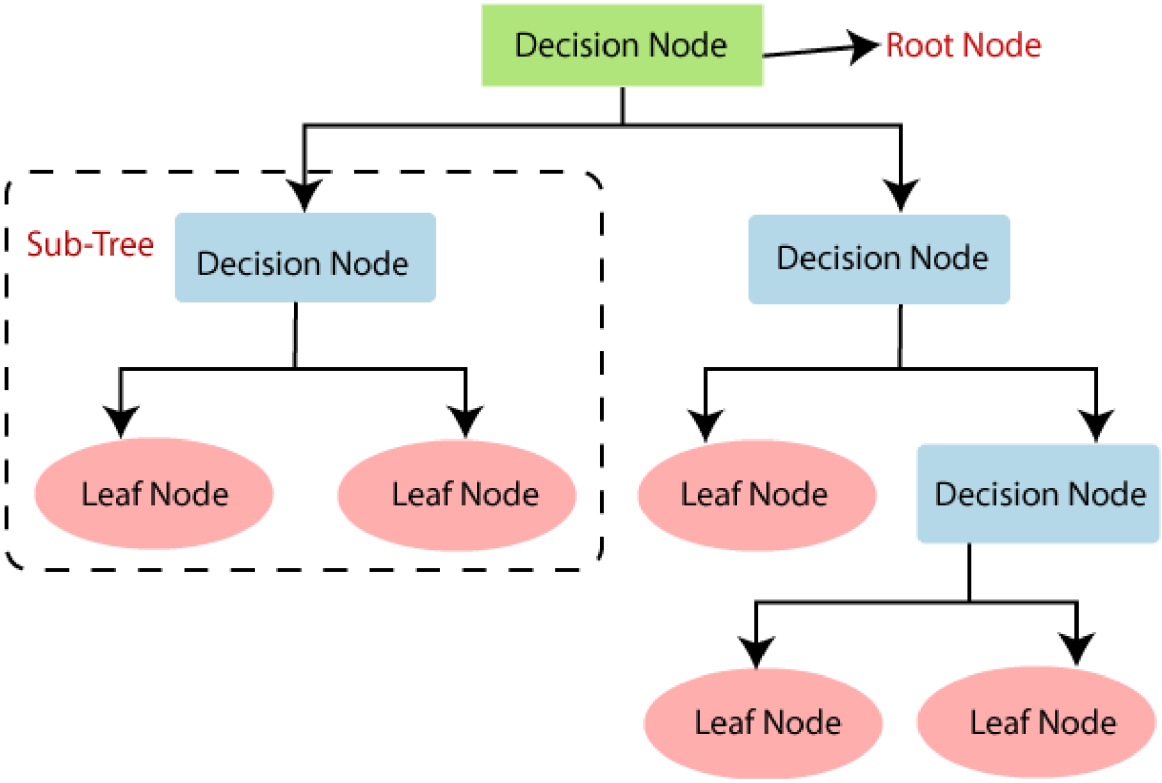

Figure 5: Decision tree structure (PiAnalytiX)

Terminology

- **Root Node**: It is the starting dataset, that is further split into more homogeneous ones.
- **Leaf Node**: Ending nodes with the outcomes.
- **Branch**: A tree formed by splitting.
- **Parent/Child Node**: The parent Node is a Root Node, and all the nodes created during the split are the Child Nodes.
- **Splitting**: It is a process of dividing the node (root or decision) into smaller ones according to conditions.
- **Pruning**: Pruning is a process of removing all the unwanted branches of the tree.

(*A Classification and Regression Tree (CART) Algorithm | Analytics Steps* [no date])

Random forest

"Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems."(*What is Random Forest? | IBM* [no date]) - this is IBM's definition of a Random Forest.

Before the training, Random Forest has three hyperparameters that need to be set. These are node size, number of features, and number of trees.

Strengths and flaws

Strengths:

- Low risk of overfitting: Considering the number of trees in a Random Forest, the model tends not to overfit since the averaging of uncorrelated trees lowers the prediction error and variance.
- Flexibility: Random Forest is highly accurate in most of the tasks given, either regression or classification, which makes it widely used by Data Scientists
- Feature Importance is easy to establish. Feature Importance is easy to evaluate using Random Forests. Techniques like Gini's importance and mean decrease in impurity(MDI) efficiently measures how much accuracy decrease with an exclusion of a variable, and Mean Decrease Accuracy (MDA) which identifies the decrease in accuracy by randomly permutating the feature values in samples, provide good help for it.

Weaknesses:

- Need of resources: Generally Random Forest works with vast amounts of data, meaning it needs more resources to store it.
- Time-consumption: because of the large datasets it is time-consuming to compute it all through every single decision tree.
- Complexity: the result of a single tree is usually easy to interpret, but when there is a forest of them it gets more complex.

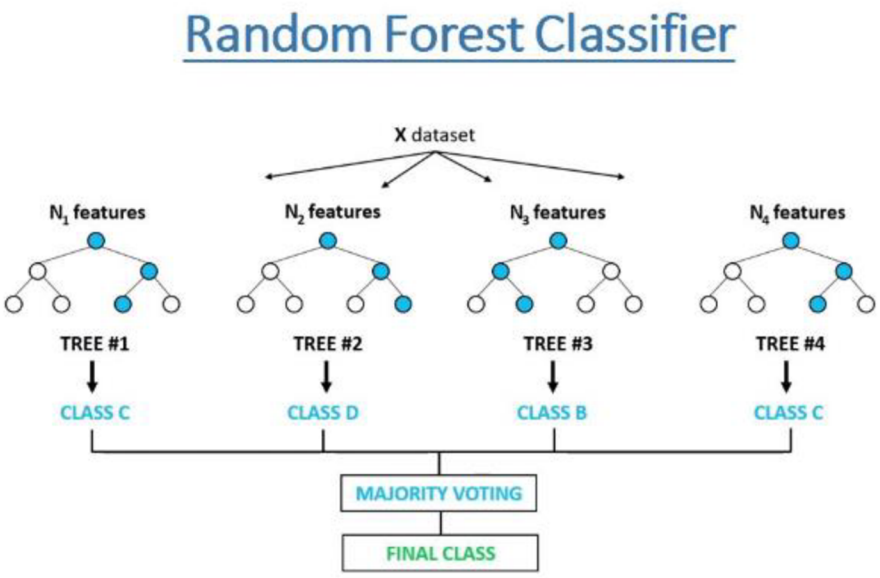

Figure 6: Random forest structure (FreeCodeCamp)

Support vector machines

SVM is a powerful tool in supervised learning. It is commonly used in Data Science, Face Detection, Image Classification, bioinformatics, Text Categorization, etc. Primarily, it is classification tasks.

SVM is a machine learning algorithm that separates data into different groups by drawing a line (called a "hyperplane") between them. The algorithm tries to draw the line in a way that maximizes the space between the line and the closest data points from each group. This space is called the "margin", and maximizing it helps the algorithm make better predictions about new data that it hasn't seen before.

For SVM, data must be previously transformed into a higher-dimensional space, it is done by applying a kernel function. The kernel function applies a linear classifier to data to separate it into a high-dimension feature space. There are a few types of Kernel functions that could be used such as linear, polynomial, and radial basis functions (RBF).

Types of SVM:

- **Linear SVM**: It is used for linearly separable data, which classifies it into classes using a straight line.
- **Non-linear SVM**: In the case of non-linearly separable data, it applies a kernel trick technique which transforms it into a higher-dimensional space, so it becomes separable by a hyperplane since non-linearly separable data cannot be classified using a straight line.

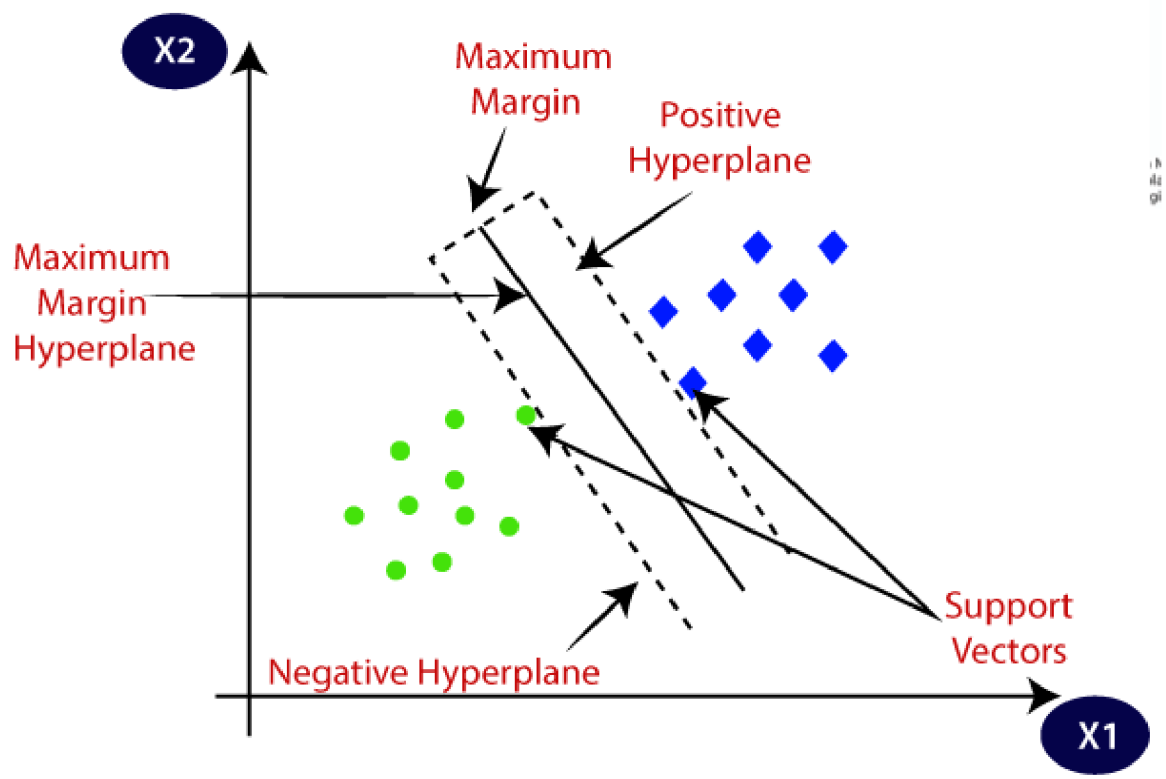

Figure 7: SVM (Medium)

(*Support Vector Machine (SVM) Algorithm - Javatpoint* [no date])

Neural networks

What are Neural Networks?

IBM definition goes as follows: "Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm."(*What are Neural Networks?* | *IBM* [no date])

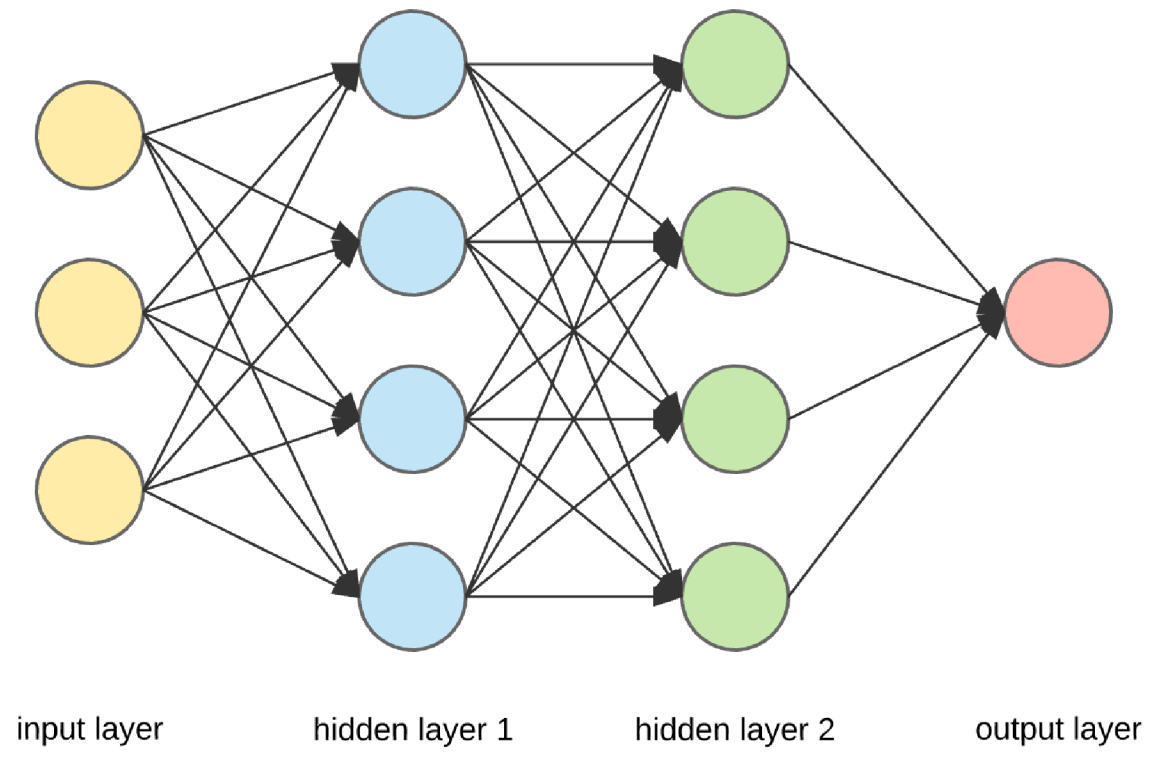

Figure 8: Neural Network Structure (Towards Data Science)

Types of Neural Networks:

- Perceptron
- Feed Forward Neural Network
- Multilayer Perceptron
- Convolutional Neural Network
- Radial Basis Function Network
- Recurrent Neural Network
- Sequence to sequence model
- Modular Neural Network

(*Explained: Neural networks* | *MIT News* | *Massachusetts Institute of Technology* [no date]; *What are Neural Networks?* | *IBM* [no date])

Perceptron

Perceptron is one of the oldest and simplest models of a Neural Network. It performs basic computations to detect some features of the input data. To get the final output, it applies an activation function to inputs. The second name of the model is Threshold Logic Unit (TLU). Perceptron is a binary classifier, meaning it classifies data into two categories and it is a supervised learning algorithm.

Advantages of the Perceptron:

- Logical Operators can be implemented such as AND, OR, and NAND;

Disadvantages of the Perceptron:

- Perceptron can only work with linearly separable problems, for non-linearly separable problems it does not work.

Feed Forward Neural Network

A Feed Forward Neural Network is a type of ANN where the information flows only forward through the input layer to output without any feedback loops.
It can and cannot have hidden layers but input and output layers are always present. All the layers relate to the next one. The input layer receives the data, and the output produces the prediction, hidden layers process and transform data by applying a nonlinear function to the sum of its inputs to produce an output. The number of layers assumes it is a single-layered or multi-layered Feed Forward Neural Network. Accordingly defines the complexity of the model. Nonlinear function is good for allowing nonlinear relationships in the data.
The weights of the connections in the model are set to minimize the error between the predicted and actual outcomes.

Feed Forward Neural Networks are widely used for Image Recognition, Financial Forecasting, Natural Language Processing, Computer Vision, etc.

Advantages:

- It has a wide range of applications, so it is flexible.
- Easy to create and maintain.
- Less complex.
- It is fast due to a one-way propagation.
- It is responsive to noisy data.

Disadvantages:

- Because of the lack of backpropagation and the absence of dense layers, it cannot be used for Deep Learning.

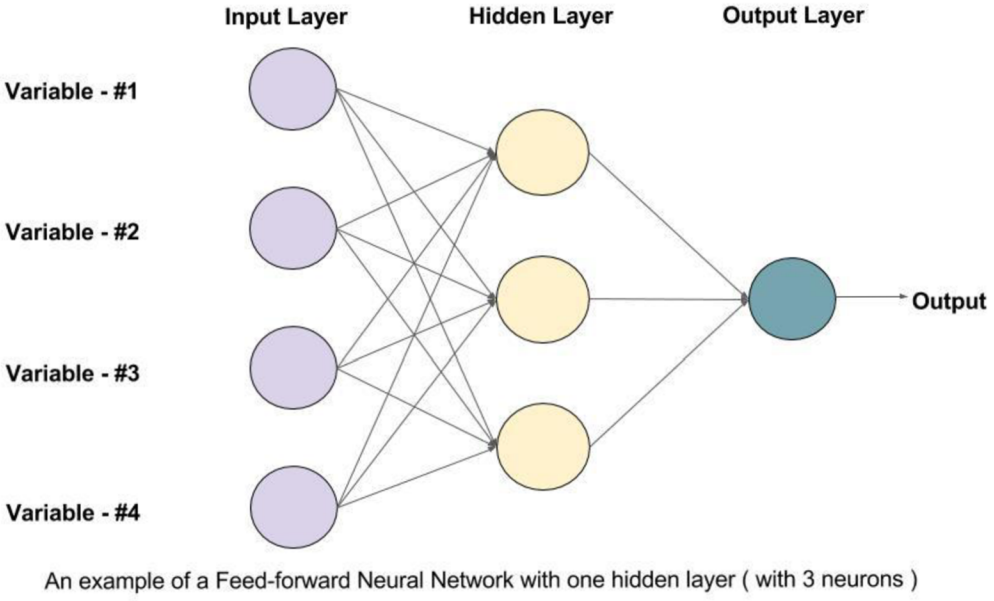

Figure 9: Feed Forward Network (Learn Open CV)

Multilayer Perceptron (MLP)

It is a more complex neural network that works by processing input data through multiple layers of artificial neurons.

Each node in one layer is connected to all of the neurons in the next layer, making it a fully connected network.

The Multilayer Perceptron model has multiple hidden layers, with at least three or more layers in total, without counting input and output layers. It applies both forward propagation and backward propagation, so-called bi-directional propagation.

The application of backpropagation involves adjustment of the weights between neurons respecting the error between predicted and actual outputs. The activation function is applied alongside softmax as the output layer activation function. The goal is to obtain the weights that minimize the error, therefore making the prediction more accurate.

Advantages:

- Because of the presence of backpropagation and fully connected layers, it can be used for Deep Learning.

Disadvantages:

- Hard to design and maintain.
- Can be slow depending on the number of layers.

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a branch of ANN, that shows the best result for image recognition and classification problems. The idea of a CNN is borrowed from the cortex in the human brain and is created to automate the extraction of the most relevant features in images.

The distinguishable element of a CNN is the presence of convolutional layers, as the name says. Convolutional layers apply filters to the input image to receive important features such as patterns, edges, shapes, etc.
As an output of the filter application, we receive a feature map which is then passed to the non-linear activation function to observe non-linearity. After convolutional layers come pooling layers, which then improve efficiency by reducing the number of parameters.
The final layers are fully connected and take the result of the previous layers to produce the output.
The training is done using backpropagation to improve accuracy.

Advantages:
Hierarchical Feature Extraction: with each layer, CNN dives deeper and learns more complex features.
Reduced Parameters: due to shared weights across the network, fewer parameters are needed, therefore it is less prone to overfit and is efficient.

Disadvantages:
Vast amounts of data are a necessity: to achieve a good performance, CNN must be trained on large amounts of data.
Need for resources: since CNNs tend to be computationally intensive, especially with large datasets and complex architectures, there is a need of having enough resources to store them.

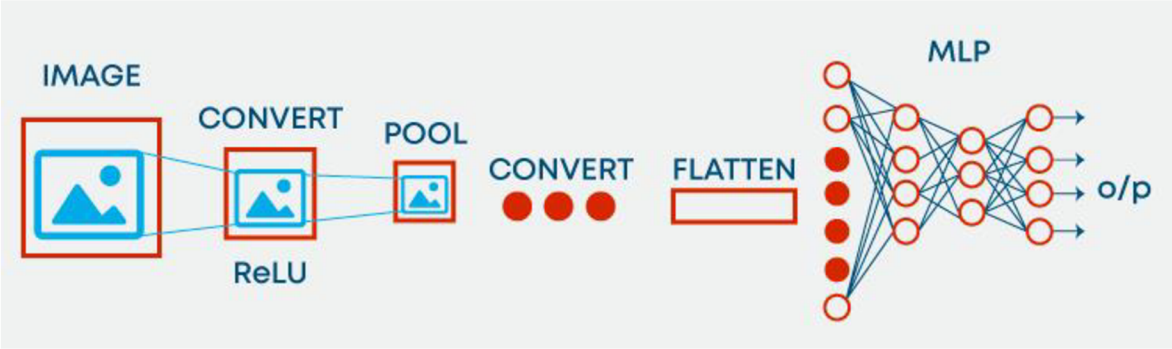

Figure 10: CNN steps (My Great Learning)

Radial Basis Function Network

A Radial Basis Function Network (RBFN) is an ANN that is mostly used for classification tasks and function approximation. RNFN consists of three layers: input, hidden, and output. A hidden layer of RBFN has a set of radial basis functions that transform the data into a higher-dimensional feature space. Functions are centered on the prototype vector, whose purpose is to determine the size and shape of a function. Each function produces an output which is then weighted by a scalar value found during the training (backpropagation).
The output layer's weights are set during the training of a supervised learning algorithm. The final output of the network is a linear combination of weighted radial basis functions.

Advantages:

- Interpretability: RBFNs output are easy to interpret.
- Generalization: RBFNs have a good performance on unseen data due to their great generalization properties.
- Fast to develop: simple structure and a small number of properties make it quick to train.

Disadvantages:

- Vulnerability: RBFNs tend to overfit in case of a higher number of functions or if data is incomplete or noisy.
- Accuracy depends on the prototype selection: prototype selection must be done carefully because it drastically affects the accuracy, especially in high-dimensional feature space.

Recurrent Neural Network

A Recurrent Neural Network is a branch of ANNs that works with sequential data. RNNs can process sequences like time series and NLP due to their ability to process data not only once, unlike other kinds of NN.

A special feature of RNNs is the feedback loop, so when the output of the layer is stored, it is fed back again, to help predict the next item in the sequence, such as the next word. In case of a false prediction, small changes are applied to gradually move it towards the correct prediction.

Advantages:

- Once used with convolutional layers, the effectiveness increases.
- Sequential data can be dependent on historical data, making it an advantage.

Disadvantages:

- Gradient Vanishing problem (Gradient Vanishing problem is a phenomenon that occurs when gradients of an error become too small as they are backpropagated, so the weights are not updated or there is a small change to them. It causes slow or no improvement at all.)
- RNNs are hard to train.

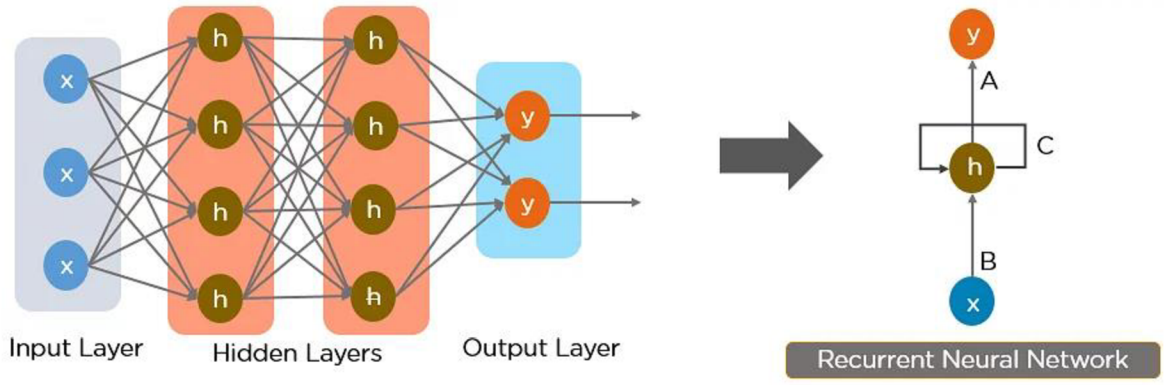

Figure 11: RNN structure (Simplilearn)

Sequence to sequence model

A Sequence-to-sequence model is made up of two RNNs. The first one is an encoder and the second is a decoder. An encoder works with an input, it processes it, and the decoder works with an output. Both work simultaneously.
The model is particularly good when the input data length equals the output data length.
The sequence to sequence model is mostly applied in chatbots, question-answering machines, and machine translation.

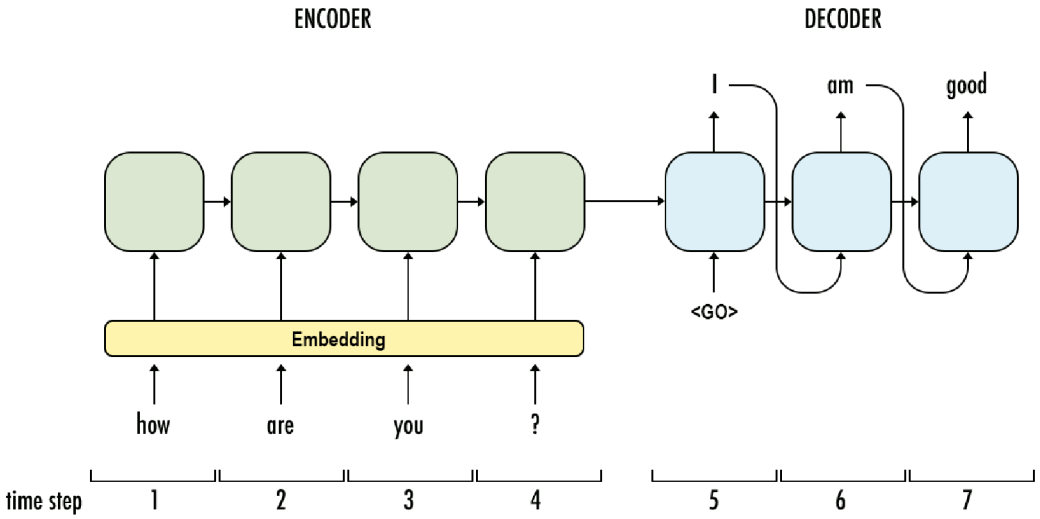

Figure 12: Sequence to Sequence model (Medium)

Modular Neural Network

A Modular Neural Network is a combination of different networks that function independently from each other, and each of them performs its tasks.
A complex problem is split into smaller tasks for each module to resolve, so as a result the whole process is done faster.

Advantages:

- Fast
- Efficient
- Independent

Disadvantages:

- Moving target problem, which occurs when the target function is trying to learn changes over time. In this case "dynamic modularization" must be applied, which allows the network to adapt to changes by adding or removing subnetworks.

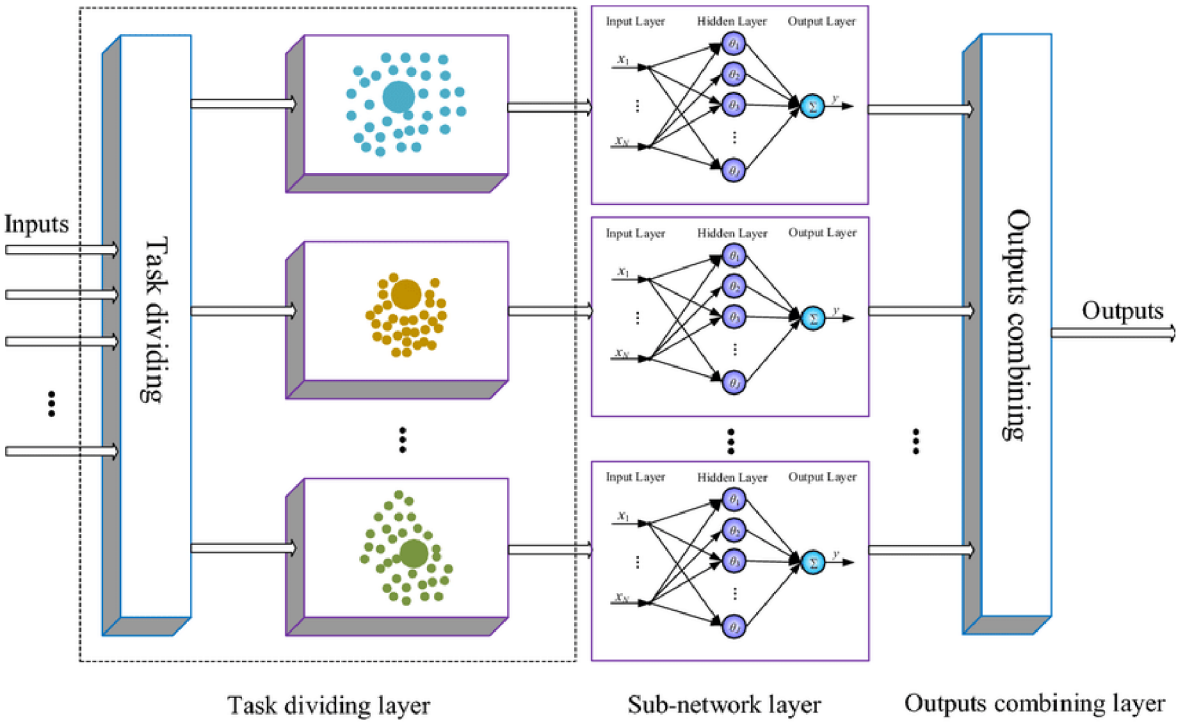

Figure 13: Modular Neural Network (ResearGate)

K-NN algorithm

K-Nearest Neighbor (KNN) algorithm is one of the simplest in supervised learning and is mostly used for classification tasks.

K-NN assumes based on available categories and labels it as one it has the most similarity to. The algorithm is non-parametric, so the prediction is done using the distance between data points.
The "K" in K-NN is the number of nearest neighbors which are taken into consideration when making a prediction for a new point.
The metric for distance in K-NN could vary depending on the problem and data given.

Some of the popular distance metrics include:

- **Euclidean Distance**: it is a measure of the distance between two points using a straight line.
- **Manhattan Distance**: it is a measure of distance between two points using a grid-like structure.
- **Cosine Similarity**: it is a measure of similarity between two non-zero vectors and is defined as a cosine angle between them.

Advantages:

- Simple to implement.
- Effectiveness increases with the amount of data.

Disadvantages:

- K-value has to be calculated at all times, which can be problematic.
- It is expensive.

(*K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint* [no date])

Naïve Bayes Classifier

Naïve Bayes is a supervised learning algorithm, based on the Bayes theorem, mostly used for solving text classification problems.

It is a probabilistic algorithm, which means it makes predictions based on probability. Bayes theorem used, provides a way to calculate the probability of a hypothesis given evidence. The Naïve Bayes classifier makes a "naive" assumption of independence between every pair of features. First, the probabilities of each class are calculated, based on the frequencies of the classes in the data. Next, for each feature, there is, probability of this feature given each class is calculated. At last, these probabilities are combined using the Bayes theorem to calculate the posterior probability of each class, considering given data. The class with the highest probability is then chosen as a prediction.

The classifier is used in spam filtration, article classification, and Sentimental analysis.

Types of Naïve Bayes model:

- **Gaussian**: this model that features have a normal distribution;
- **Multinomial**: this type of model works when the data has a multinomial distribution;
- **Bernoulli**: The Bernoulli classifier is similar to the Multinomial classifier, except the predictor variables are independent and Boolean.

(*Naive Bayes Classifier in Machine Learning - Javatpoint* [no date]; *1.9. Naive Bayes — scikit-learn 1.2.2 documentation* [no date])

2.4.4 Hyperparameter tuning

What are hyperparameters?
Hyperparameters are parameters that are set before the training begins by a user. They cannot be learned from data during the training like model parameters.
They directly affect the learning process and have a significant impact on the performance of the model. Hyperparameters are the number of hidden layers, learning rate, regularization, depth and width, or the number of clusters.

What is hyperparameter tuning?
Finding the best set of hyperparameters that maximize the performance of a model is called hyperparameter tuning.

Some of the popular hyperparameter tuning methods include:

- **Grid Search**: This method has a predefined grid of hyperparameters, and it is searching over until it finds the best combination of them.
- **Random Search**: Unlike grid search, there is no predefined grid, but a range of hyperparameters. The search is done randomly within the given range until the best combination is found.
- **Bayesian Optimization**: This is a probabilistic method, which uses an objective function model to iteratively search for the optimal combination of hyperparameters. This method compared to the previous ones requires fewer evaluations and can work with non-convex and non-smooth search spaces but is more expensive.
- **Evolutionary Algorithms:** These are optimization methods that take genetics and natural selection as inspiration. They create a population of solutions(hyperparameters), make them evolve by selection, reproduction, and mutation, then select the best of them for the next generation. The process can be computationally expensive.

(*Hyperparameter tuning - GeeksforGeeks* [no date])
(*Hyperparameter tuning for machine learning models.* [no date])

2.4.5 Model Training

Once data is prepared and the right model is chosen comes the training phase. During the training, patterns, and relationships in data are learned.

Steps of the training process:

1. Model Initialization: parameters are initialized randomly or with pre-trained weights.
2. Forward Pass: data is fed to the model and prediction is made.
3. Loss Computation: After comparing the actual output with desired, the loss or error metric is computed. This metric describes how well the model performed.
4. Backward Pass: using backpropagation, gradients of the loss are computed with respect to the model's parameters.
5. Update of the parameters: to minimize the loss, parameters are updated using an optimization algorithm.
6. Repeat: steps 2 to 5 are repeated, until the desired level of performance or stopping criteria is met.

The output is a trained model that is ready to make predictions on unseen data.
(*Support Vector Machine (SVM) Algorithm - Javatpoint* [no date]; *What is Random Forest? | IBM* [no date])

2.4.6 Model Evaluation

There are multiple ways of evaluating an ML model. For this, testing and validation datasets are needed, so it is good not to train the model on the whole dataset, but to split it into partitions of 60%, 20%, and 20%, for training, testing, and validation accordingly.

When performing a prediction, the following types of outcomes could occur:

- True positives (TP): occurs when the predicted positive and it is positive.
- False positives (FP): occurs when predicted positive but it is negative.
- True negatives (TN): occurs when predicted negative and it is negative.
- False negatives (FN): occurs when predicted negative but it is positive.

Metrics to evaluate a model include:

Classification:

- Accuracy: This metric measures how many times the model predicts the class correctly and is calculated by dividing the number of correct predictions by the total number of them.
- Precision: This metric is a proportion of TP and total predicted positive instances.
- Recall: It is a proportion of TP and all truly positive instances.
- F1-score: This score is the mean of both recall and precision.

Regression:

- Mean Absolute Error (MAE): It is an average absolute difference between the actual and predicted values.
- Mean Squared Error (MSE): It is an average squared difference between actual and predicted values.
- Root Mean Squared Error (RMSE): It is a square root value of MSE.
- R-squared: This metric describes how well the model fits the data. The measure lies in the range of 0 and 1.

Clustering:

- Davies-Bouldin Index: It is a measure that describes the average similarity between the most similar cluster and every other cluster.
- Silhouette Score: This score is a measure of the similarity of an instance to its cluster compared to others.
- Calinski-Harabasz Index: It is a ratio of between-cluster and within-cluster dispersions.

Recommender Systems:

- Precision: It is a proportion of recommended objects that are relevant.
- Recall: It is a proportion of relevant objects that were recommended.

(*Evaluating a machine learning model.* [no date]; *Various ways to evaluate a machine learning model's performance | by Kartik Nighania | Towards Data Science* [no date])

2.5 Used technologies

2.5.1 Python

Python is a versatile, high-level, open-source programming language that was created by Guido van Rossum and first released in 1991. The language is designed with readability and simplicity in mind, emphasizing clean and easily understandable code. Python's syntax allows developers to express complex concepts with fewer lines of code than other languages like C++ or Java, making it a popular choice for both beginners and experienced programmers alike.

Python is an interpreted language, which means that the source code is executed directly rather than being compiled into machine code beforehand. This enables rapid prototyping and debugging, as changes can be made and tested immediately without the need for lengthy compilation processes.

Python is known for its extensive standard library, which provides a wide range of functionalities out-of-the-box. This "batteries-included" philosophy allows developers to quickly build applications without having to search for and install external libraries. Furthermore, Python boasts a vibrant and active community that continuously contributes to the development of third-party packages, which can be easily installed and managed using package managers like pip.

Python is a general-purpose programming language, suitable for various applications, such as:

- **Web development:** Python's popular web frameworks, like Django and Flask, enable developers to create powerful and scalable web applications with ease.
- **Data analysis and visualization:** Python's rich ecosystem of data manipulation and visualization libraries, including Pandas, NumPy, and Matplotlib, makes it a great choice for processing, analyzing, and visualizing large datasets.

- **Artificial intelligence:** Python is the go-to language for AI projects, thanks to its vast array of libraries and frameworks like TensorFlow, PyTorch, and scikit-learn, and in our case xGBoost.
- **Automation and scripting:** Python's simplicity and cross-platform compatibility make it a popular choice for automating tasks, creating scripts, and developing tools for various platforms.
- **Game development:** Python's Pygame library provides a simple yet powerful foundation for creating 2D games and multimedia applications.

2.5.2 XGBoost

XGBoost is an open-source library, currently available in, C++, Java, Python, R, Julia, Perl, and Scala, for gradient boosting.
The name "XGBoost" stands for extreme gradient boosting, which refers to its optimization objective and its use of gradient boosting techniques.
It is widely used for various machine learning applications, like classification, regression, and ranking problems.
It has been developed by Tianqi Chen as a research project, and from there surged in popularity because of its high efficiency, and accuracy modeling complex data. Eventually packages with the algorithm started becoming available in more and more programming languages, making it easily, and widely available.
Because of its ability to handle missing values and outliers, XGBoost became widely adopted in various industries, including finance, e-commerce, healthcare, and of course football data analysis, as it is a robust approach for real-world data problems.
xGBoost uses a data structure called DMatrix to store and work with the training data.
The Dmatrix data structure is highly optimized, memory efficient and designed for the xGBoost library. It can store both the features, and the objective variable in an efficient way, as well as being memory efficient with sparse data, meaning it can represent zero or missing values without taking up much memory, which can be common in real world data sets.

2.5.3 Pandas

Pandas is an open-source Python library designed for data manipulation and analysis. Developed by Wes McKinney, Pandas has quickly become a vital component of the Python data science ecosystem. Its primary purpose is to provide efficient, flexible, and user-friendly data structures for managing and analyzing structured and semi-structured data, such as spreadsheets, SQL tables, and time series data.

The name "Pandas" originates from the term "panel data," which refers to multi-dimensional structured datasets commonly utilized in statistics and econometrics. The key data structures in Pandas are the DataFrame and the Series, which offer an intuitive and convenient way to work with data.

Pandas is widely used for various data-related applications, including:

- **Data cleaning and preprocessing:** Pandas offers a broad range of tools for filtering, sorting, renaming, reshaping, and aggregating data, as well as handling missing or null values. This allows users to efficiently clean and prepare their datasets for further analysis.
- **Data analysis and visualization:** With its seamless integration with other Python data analysis libraries like NumPy, Matplotlib, and Seaborn, Pandas enables users to perform in-depth data analysis and create visual representations of their findings.
- **Time series manipulation:** Pandas provides extensive support for working with time series data, making it an excellent choice for handling datasets with time-dependent elements, such as financial or sensor data.
- **Data import and export:** Pandas can read and write data in a variety of file formats, including CSV, Excel, SQL databases, JSON, and more, allowing for smooth integration with existing data workflows.

Like XGBoost, Pandas has garnered widespread adoption in various industries, such as finance, e-commerce, healthcare, and sports data analysis. Its robust approach to handling

real-world data challenges has made it a popular choice for professionals and researchers alike.

2.5.4 Matplotlib

The matplotlib library, a widely recognized tool within the Python programming ecosystem, serves as a powerful resource for generating a vast array of visualizations. Its primary goal is to facilitate the creation of static, animated, and interactive visual representations of data, making it an invaluable asset for data analysts, scientists, and researchers alike.

Boasting a comprehensive range of capabilities, matplotlib allows users to craft a multitude of charts and graphs, including line plots, scatter plots, bar plots, and histograms. This versatility ensures that data can be represented in the most appropriate and insightful manner, depending on the specific context and objectives.

One of the key strengths of matplotlib lies in its high level of customizability. Users can tweak various aspects of their visualizations, such as colors, styles, labels, and axes. This adaptability empowers individuals to create visually appealing and informative graphics that cater to their specific needs and preferences.

In addition, matplotlib's compatibility with other Python libraries, such as NumPy and pandas, further enhances its functionality. By seamlessly integrating with these popular tools, matplotlib enables users to effortlessly generate visualizations from data stored in various formats, streamlining the data analysis process.

In summary, the matplotlib library stands as an indispensable resource within the Python landscape, offering users a versatile and customizable solution for producing a wide range of data visualizations. Its seamless integration with other popular Python libraries and its adaptability make it a go-to choice for professionals seeking to convey their data in a visually engaging and informative manner.

3 Practical Part

The practical part of this thesis will focus on the development of a ML model that will estimate the chance of scoring a goal from a given situation in a football match. This process will involve an analysis of the best inputs for the given task, testing of the best parameters for the model training, and finally presenting a visual representation of the results.
The model will be trained using the xGBoost library in Python, and the visualization will be prepared using matplotlib.

The program will be able to:

- Load the base shot data from a txt file as a Pandas dataframe.
- Split the loaded data to a training and testing data set.
- Transform the data sets to a DMatrix needed for xGBoost.
- Train models with different parameters and input columns.
- Test the accuracy for each model.
- Save the test results for the models into a txt file.
- Determine the best parameters/model from the test results.
- Use the best model to visualize the results as a shot map using matplotlib.

3.1 Definition of intents

This program is meant to find the ideal parameters for training a ML model using xGBoost for determining the chance of scoring a goal from a given situation.
This has been done many times before by different companies and enthusiasts alike, however it is my intention to test maybe never before used features, and explore their effects on the final model evaluation.
This means I will train tens of thousands of models using different combinations of parameters and different situational information inputs(features), to determine the most valuable combination of features and parameters.
The best model will be determined by scoring the model with two statistical tests.

- Mean squared error
- R2

The chosen model will have high scores in both statistical tests.

After choosing the best model, I will select a sample of shots, and create a so called shot map, to visualize some of the results of the model.

3.2 Workspace configuration

The program will be run using Python. As for the environment we will create a virtual environment using Python 3.11.

For the packages I need to install:

- xGBoost
- pandas
- scikit-learn
- matplotlib

into the virtual environment.

As for the IDE I'm using PyCharm.

3.3 Training data

The basis of the training data is Wyscouts event data. With a lot of transformations, I have prepared features that I believe could be used by the model to enhance its ability to predict the chance of scoring a goal from a given situation. I will explain the individual inputs, and later test if the inputs, and different combinations of these inputs helped the accuracy of the predictions of the model.

3.3.1 The features

The features presented here are not the features used for the final model. These are the features that will be tested.

- **Px + Py:** Are the coordinate of the player on the pitch when he shoots his shot. This is expected to be one of the strongest indicators, as being close to the goal will have a major influence on the overall probability to score.
- **Distance from goal + Angle to goal:** A simpler representation of the position of the shooting player on the pitch, relative to the goal.
- **Previous event type:** A variety of event types, that indicate the type of event that preceded the shot. The types of events preceding the shot could indicate how well the defense is prepared for the following attack. This information is available for up to 3 preceding events.
- **Shot body part type:** Shot body part type tells the model whether the player is using his main foot, head, or his weaker foot for the shot.
- **Time since last event:** Seconds since last event before the shot. This can indicate how much time the shooter had with the ball before shooting.
- **Shot after Free kick**: Indicates if the shot was taken during a free kick scenario. Set pieces are very specific scenarios, therefore it could be useful to provide this information.
- **Shot after Corner:** Indicates if the shot was taken during a corner scenario. Set pieces are very specific scenarios, therefore it could be useful to provide this information.

- **Shot after Throw-in:** Indicates if the shot was taken during a Throw-in scenario. Set pieces are very specific scenarios, therefore it could be useful to provide this information.
- **Primary position:** A simplified position of the player, where the side which he is playing on is not considered. The position of the player can indicate his scoring abilities, and possibly indicate where other players might be relative to him.
- **Primary position Detailed:** Similar to Primary position, however with more detail. Here we indicate if the player is playing on the left, right, or middle as well as the type of position he is playing.
- **Meters per sec.:** Meters per second is an indicator which can tell us the pace of the game. Faster forward pace might indicate that the defense is less organized, and might lead to better goal scoring chances. This information is available for the past 3 events.
- **Preceding event by same player:** This indicates how many preceding actions our player had the ball in his control. If he has the ball for many events in a row, this might indicate that the defense had more time to prepare for the attack from him, or vice versa. In combination with other provided indicators, this could bring further context for the current situation for the model. This information is available for the preceding 3 events before the shot.
- **Preceding event by same team:** This indicates when the attacking team was in possession of the ball. Giving further context for what is happening before the shot. This information is available for the preceding 3 events before the shot.

Objective:

- **Goal:** Information if the shot was a goal or not. This is what the model is trying to predict.

3.4 Code implementation

Description of individual functions needed for the model training, testing, visualization and statistical outputs.

3.4.1 Loading the training data

I created a function called “get_shots”.
It uses pandas to read and transform the data into a pandas dataframe from a csv file.
The csv file contains the training data for the model. It includes over 180 000 shots and includes all the information described in “The inputs” for every one of them. This amount of data should be a good sample size for our training purposes.

```
import pandas as pd
import os


def get_shots():
    path = r'./events/data'
    df = pd.read_csv(os.path.join(path, "shots.csv"), sep=",")
    return df
```

Figure 14: GetShots (Own)

3.4.2 Final data transformation

Upon receiving the training data from the "get_shots" function, it is presented to us in the form of a pandas dataframe, encompassing all the available columns from the comprehensive training data set. Our task at hand is to cherry-pick only those columns which are of relevance to the current round of model training. This is essential since we intend to experiment with various input combinations. Consequently, the function will be designed to accept an array of column names as input, proceeding to select only these specific columns as part of the final format preparation process. Following this, the selected columns will be saved as a separate

data frame. In addition, it is imperative that we include the target variable, denoted as "goal", since the model's ultimate objective is to generate accurate predictions for this variable. This target variable, too, will be preserved in a distinct variable as a dataframe.

As a result of these operations, we find ourselves in possession of two dataframes – one encompassing all the feature variables, while the other focuses exclusively on the target variable. Nevertheless, for the sake of evaluating the performance of our trained model, it is crucial that we further segregate the data into distinct train and test sets. To accomplish this, I employ the "train_test_split" function, courtesy of the sklearn library. This process entails allocating 30% of the data to the test set, while the remaining 70% is designated for the actual training of the model. This split ensures a comprehensive and efficient training process, while still reserving an adequate portion of the data for testing purposes.

At this point, we have four data frames:

- x_train
- x_test
- y_train
- y_test

The variable names starting with x are the features, and the y variables are the target variables. They are still pandas dataframes.

However, that is not the correct format for the xGboost model training algorithm. xGBooost expects us to pass a DMatrix object. To solve this, we use the xgb.DMatrix function, which takes both the features and the target variable dataframes, and creates a DMatrix, which can now be used for the training of our model.

Here is the code:

```python
from sklearn.model_selection import train_test_split

import xgboost as xgb


def prepare_training_data(data, clist):
    y = data["goal"]
    X = data[clist]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    dtrain = xgb.DMatrix(X_train, label=y_train)
    dtest = xgb.DMatrix(X_test, label=y_test)

    return {
        "training_data": dtrain,
        "testing_data": dtest,
        "testing_data_correct_result": y_test,
        "clist": clist
    }
```

Figure 15: PrepareTrainingData (Own)

3.4.3 Model training algorithm

For the model training purposes, I have created a class called “ModelTraining”.

```python
class ModelTraining:
    num_round = 0
    run_for_input_cols_combos = True
    params = {'max_depth': 1,
              "eta": 0.1,
              'objective': 'multi:softprob',
              'num_class': 2
              }
    columns_for_model_training = ['shot_after_corner', 'shot_after_free_kick',
                                  'previous_event_type',
                                  'time_since_last_event', 'py', 'px']

    base_available_columns = ['py', 'px', "previous_event_type", "shot_body_part_type", "shot_after_corner",
                              'time_since_last_event', 'meters_per_sec', "primary_position_detailed",
                              'shot_after_free_kick', "previous_event_type_lag2", 'meters_per_sec_lag2',
                              'meters_per_sec_lag1', "lag_1_is_same_player", "lag_1_is_same_team",
                              "time_since_last_event_lag1"]

    columns_to_try = ["lag_2_is_same_player", "time_since_last_event_lag2", "time_since_last_event_lag3",
                      "meters_per_sec_lag3", "previous_event_type_lag3"]
```

Figure 16: ModelTraining class (Own)

I chose to use class variables to store some relevant information and parameters for the class.

- **num_round:** Is used as a parameter for the model training. It can be specified by hand, or if we use a function that will try different parameter combinations, this parameter will automatically be changed by the program.
- **run_for_input_cols_combos:** Is set manually to determine if the model training should use the different column(input) combinations.
- **params:** Stores a dictionary with some default settings for the model parameters. These are either set manually, or are changed by the program, depending on the training function used..
- The remaining variables store information about the columns that have already been chosen by the model, or should be tested when running the program that explores different parameters and inputs.

The most basic function in the class is the "train_model_for_params" function.

```
def train_model_for_params(self, columns_for_model_training):
    raw_data = get_shots()
    training_data = prepare_training_data(raw_data, columns_for_model_training)
    train_xg_model(training_data, self.params, self.num_round)
```

Figure 17:TrainModelForParams (Own)

This function takes the training parameters that are manually set in the class variables, and the columns which should be used as features for the model training as the inputs, and then calls the functions for loading and transforming the data, and runs the function "train_xg_model".

```
def train_xg_model(data, param, num_round):
    model_name = f"{param['max_depth']}_{param['eta']}_{num_round}"
    bst = xgb.train(param, data["training_data"], num_round)
    models_path = os.path.join(os.getcwd(), "xg", "model", f"{model_name}.json")
    bst.save_model(models_path)
    get_and_save_test_results(bst, data, param, num_round)
```

Figure 18: TrainxGModel (Own)

In this function, we prepare a name for the model from the given parameters, and proceed to train the model using "xgb.train", providing the training parameters and data. The train function will train our model, which we can then save using ".save_model" and providing a path.

At the end I am calling the function "get_and_save_test_results", while passing the model, data and the used parameters.

```
def get_and_save_test_results(model, data, params, num_round):
    preds = model.predict(data["testing_data"])
    xg = pd.DataFrame(preds,
                      columns=['col1', 'xg'])["xg"]
    mean_sq = mean_squared_error(data["testing_data_correct_result"], xg)
    mean_abs_err = mean_absolute_error(data["testing_data_correct_result"], xg)
    r2 = r2_score(data["testing_data_correct_result"], xg)

    save_test_results(params, num_round, data, mean_sq, mean_abs_err, r2)
```

Figure 19: GetAndSaveTestResults (Own)

This function will use the model, evaluate the performance, and save the results of the models performance, along with the used parameters and features.

Using "model.predict" with our testing data set, will evaluate the testing data set, and return the predictions for each shot.

I then transform the output to a pandas dataframe, extracting the "chance to score" column,and store it as "xg".

Now we can evaluate the performance. For the evaluation, I chose to save three results.

- Mean square error
- Mean squared absolute error
- and R2

I get these metrics by passing them into their corresponding functions with the prediction, and the actual correct results. Meaning the testing set target variable.

At the end of this function, another function is called, named “save_test_results”

```
def save_test_results(param, num_round, data, mean_sq, mean_abs_err, r2):
    path = os.path.join(os.getcwd(), f"test_results.txt")
    file1 = open(path, "a")  # append mode
    file1.write(f"{data['clist']};{param};{num_round};{mean_sq};{mean_abs_err};{r2}\n")
    file1.close()
```

Figure 20: SaveTestResults (Own)

Here we are simply appending to a text file, and saving the parameters, used features, as well as the test results, and the program stops.

This describes the whole process of calling the function “train_model_for_params”.
However this is only meant to train a specific model, when we know what parameters to use.
To training large amounts of different models, I have created another function, that will go through all the possible combinations, and save the results, just like in the previous example.

The function is called “train_models”.
Just like before, we need to load our training data.
By default, we will call a function called “combs” which takes all the columns that we are meant to test and create every possible combination of features from them.
Some of these combinations might not make sense, so depending on the columns that I'm trying to test, I will filter them using the function “clean_combos”, where the array of features is checked, and some are filtered out. For example, if we look at the features “primary_position” and “primary_position_detailed”, we will not expect there to be a benefit

having them both as features at the same time. So, to save some time, I filter out every combination of features, where both of these features are present, and at the end, we should be able to determine which one should be kept for the final model.

When the unwanted combinations are filtered out, we prepare the training data, and run the "run_for_num_round_funtion".

```
def train_models(self):
    raw_data = get_shots()
    if self.run_for_input_cols_combos:
        all_combinations = combs(self.columns_to_try)
        final_combinations = clean_combos(all_combinations)
        for row_combo in final_combinations:
            print(row_combo)
            self.columns_for_model_training = [*row_combo, *self.base_available_columns]
            self.training_data = prepare_training_data(raw_data, self.columns_for_model_training)
            self.run_for_num_round()
    else:
        self.training_data = prepare_training_data(raw_data, self.columns_for_model_training)
        self.run_for_num_round()
```

Figure 21: TrainModels (Own)

The "run_for_num_round" function is another step in the whole process of trying all different combinations.
We loop through multiple num_rounds, adjusting the class variable that is used as the parameter for model training, and go to the next parameter, max depth.

```
def run_for_num_round(self):
    for num_round in range(3, 30, 1):
        self.num_round = num_round
        self.run_for_max_depth()
```

Figure 22: RunForNumRound (Own)

Same as for num_round, we loop through different settings while changing the parameters, and run the next parameter manipulator each time. This time "run_for_eta".

```
def run_for_max_depth(self):
    for max_depth in range(2, 13, 1):
        self.params["max_depth"] = max_depth
        self.run_for_eta()
```

Figure 23: RunForMaxDepth (Own)

The final loop inside of all the loops, “run_for_eta”, is the one that is calling “train_xg_model”. This function has been explained above, so it will train the model with the current parameters, run the tests, and store the results.

```
def run_for_eta(self):

    for eta in range(2, 11, 2):
        self.params["eta"] = eta / 10
        train_xg_model(self.training_data, self.params, self.num_round)
```

Figure 24: RunForEta (Own)

3.4.4 Find best results

After running the model training algorithm, and going through all the possible input combinations, the file “test_results.txt” will be filled with the records for each trained model, in csv format. The task of the following function will be to print out the best model based on two evaluation metrics.

- Mean square error
- R2

This information is already stored, the program will only look for the best results, and print them out for us.

To take into account both values, I decided to look for entries, where both are indicating great results compared to the other trained models.

The first step is to get a very high quantile value for both metrics from the dataset.

Because I loaded the file as a pandas dataframe, I'm able to use the built in function ".quantile", and pass the desired quantile over a selected column.
After getting both quantile values, I can filter the dataframe from entries that do not satisfy both quantiles, and I will be left with only the models with the best results. I adjust the quantile value to the point of only getting a single entry, and that will be the chosen model.

```
import os

import pandas as pd


def get_test_results():
    path = os.path.join(os.getcwd(), "test_results.txt")
    file = pd.read_csv(path, sep=";")
    top_10_percentile_mean_sq = file.mean_sq.quantile(0.000025)
    #top_10_percentile_mean_abs_err = file.mean_abs_err.quantile(0.025)
    top_10_percentile_r2 = file.r2.quantile(0.999975)
    final = file[(file.mean_sq <= top_10_percentile_mean_sq) & (
                file.r2 >= top_10_percentile_r2)]
    print(final["columns_used"].values.tolist())
    print(final["num_rounds"].values.tolist())
    print(final["parameters"].values.tolist())
    print("MeanSq: ", final["mean_sq"].values.tolist())
    print("R2: ", final["r2"].values.tolist())
```

Figure 25: GetTestResults (Own)

3.4.5 Model showcase

Data preparation

The first step of the algorithm will be to load the shots we want to showcase and transform them into the appropriate format.

We load the csv with the shots using pandas.read_csv, giving us a pandas dataframe with all the shots.

Next step needs to be the transformation to the DMatrix data structure, so we need to choose only the relevant columns for the model and pass the pandas dataframe to the “DMatrix” function using xGBoost.

```
def load_and_prepare_data():
    path = r'./chosen_shots'

    chosen_shots = pd.read_csv(os.path.join(path, "haaland_shots.csv"), sep=",")
    X = chosen_shots[['shot_after_corner', 'shot_after_free_kick', 'opportunity', 'previous_event_type',
                      'time_since_last_event', 'py', 'px']]
    chosen_shots_dmatrix = xgb.DMatrix(X)
    return X, chosen_shots_dmatrix
```

Figure 26:LoadAndPrepareData (Own)

Evaluation

To evaluate the shots, we simply create a xGBoost booster instance, load the model, and use the “.predict” function while passing the DMatrix with the shots.

The result will be a Numpy array, where the model's evaluations are stored.

```
def evaluate_shots(shots):
    model = xgb.Booster()
    model.load_model("./chosen_model/model.json")

    predictions = model.predict(shots)

    return predictions
```

Figure 27: EvaluateShots (Own)

Shots preparation

With the model’s evaluation, it is time to connect them to the shots to which they belong, and create a final data structure that we will use for the visualization and other data showcasing.

The predictions numpy array stores both the chance to score, and the chance not to score. To get the chance to score, we transform the numpy array to a pandas dataframe, and name the columns col1 and the second one “xg”, as that is where the xG value is stored.

We can then use the ".assign" function to connect the shot pandas dataframe with the newly created predictions pandas dataframe, this will leave us with a dataframe with the xG value as well as the other shot details.

Finally, I transform the data frame to a regular dictionary.

```
def prepare_shots(predictions, X):

    evaluated_shots_pd = pd.DataFrame(predictions,
                                      columns=['col1',
                                               'xg'])

    """extract the evaluation and save as pandas DF"""
    shots_with_xg_pd = X.assign(xg=evaluated_shots_pd["xg"])[
        ["px", "py", "xg"]]

    shots = shots_with_xg_pd.to_dict('records')
    return shots
```

Figure 28: PrepareShots (Own)

Shot map

Finally, we draw the shot map.

We start with creating a new matplotlib figure and create a subplot.

Then we call the draw_pitch function.

```
def draw_shot_map(shots):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    fig, ax = draw_pitch(ax, fig)
```

Figure 29: DrawShotMap (Own)

In the draw_pitch function, I select the color of the lines, and set the width and length of the pitch.
The length of the pitch can vary from pitch to pitch, but I chose these parameters as a middle ground.

First, I create the outline of the pitch.
Since we are focusing on shots, I will draw only the opposite half of the pitch.

```python
from matplotlib.patches import Arc
import matplotlib.pyplot as plt

def draw_pitch(ax, fig):
    linecolor = "black"
    width = 68
    length = 105

    # Outline
    plt.plot([length/2, length], [width, width], color=linecolor)
    plt.plot([length, length], [width, 0], color=linecolor)
    plt.plot([length, length/2], [0, 0], color=linecolor)
```

Figure 30: DrawPitch (Own)

After that I draw the Goal post, small and regular Box onto the field.

```python
# Goal
plt.plot([length + 2, length], [(width / 2 - 7.32 / 2), (width / 2 - 7.32 / 2)], color=linecolor)
plt.plot([length + 2, length+2], [(width / 2 + 7.32 / 2), width / 2 - 7.32 / 2], color=linecolor)
plt.plot([length + 2, length], [(width / 2 + 7.32 / 2), (width / 2 + 7.32 / 2)], color=linecolor)

# Box
plt.plot([(length - 16.5), length], [(width / 2 + 16.5), (width / 2 + 16.5)], color=linecolor)
plt.plot([(length - 16.5), (length - 16.5)], [(width / 2 + 16.5), (width / 2 - 16.5)], color=linecolor)
plt.plot([(length - 16.5), length], [(width / 2 - 16.5), (width / 2 - 16.5)], color=linecolor)

# SmallBox
plt.plot([length, length - 5.5], [(width / 2 + 7.32 / 2 + 5.5), (width / 2 + 7.32 / 2 + 5.5)], color=linecolor)
plt.plot([length - 5.5, length - 5.5], [(width / 2 + 7.32 / 2 + 5.5), width / 2 - 7.32 / 2 - 5.5], color=linecolor)
plt.plot([length - 5.5, length], [width / 2 - 7.32 / 2 - 5.5, width / 2 - 7.32 / 2 - 5.5], color=linecolor)
```

Figure 31: DrawGoalpostAndBoxes (Own)

Finally, we add the Spots and arc

and our pitch is done.

```
    centre_spot = plt.Circle((length / 2, width / 2), 0.4, color=linecolor)
    ax.add_patch(centre_spot)

    pen_spot = plt.Circle((length - 11, width / 2), 0.2, color=linecolor)
    ax.add_patch(pen_spot)

    # Prepare Arc
    arc = Arc((length - 11, width / 2), height=18.3, width=18.3, angle=0, theta1=128, theta2=232, color=linecolor)

    ax.add_patch(arc)

    plt.xlim(length/2, length+3)

    plt.gca().set_aspect('equal')

    return fig, ax
```

Figure 32: DrawCircles (Own)

Here is the result of our drawing:

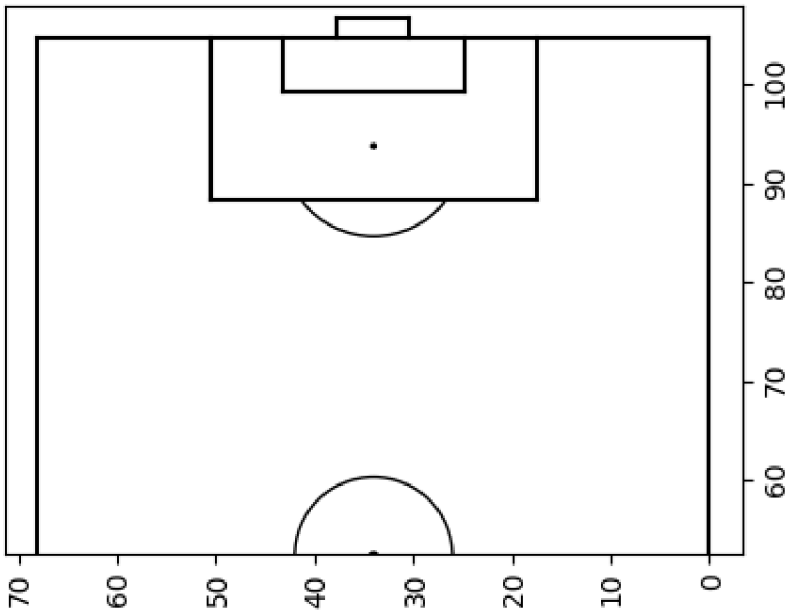

Figure 33: Pitch (Own)

Draw shots

To draw the shots, we choose a colormap from the matplotlib library, that will give us a color scale, which we can then use to assign different colors to different xG values.

We then loop though all the shots, and draw a circle using matplotlibs Circle, while providing the coordinates and the shots assigned color.

```
def draw_shots(shots, ax):
    cmap = colormaps["autumn"]
    scaled_norm = colors.Normalize(vmin=0.7, vmax=1)

    for shot in shots:
        shot_color = colors.to_hex(cmap(scaled_norm(1-shot["xg"])))
        shot_to_draw = plt.Circle((shot["px"], 68-shot["py"]), 0.5, color=shot_color)
        ax.add_patch(shot_to_draw)
    return ax
```

Figure 34: DrawShots (Own)

3.5 Training process

With the model training algorithm complete, as well to identify a good set of parameters and features, it is time to run the program.

The first obvious issue was the overwhelming number of possible combinations of the different features and parameters. So, my first focus was to gradually eliminate some columns and combinations, rather than running the program for every possible combination.

To reduce the number of models I needed to train, I would adjust the for loops for the parameters like eta, num_rounds, and max_depth, to jump by more than one at a time. This would dramatically reduce the number of models trained, while still allowing me to get a good idea for which features matter the most.

3.5.1 First iteration

In the first test, where I tested only 7 features, I was able to eliminate the following features:

- Shot after Throw-in
- Distance from goal + Angle to goal

The “Shot after Throw-in” feature seems to have no benefit when it comes to goal scoring probability. It was not present in the features list for any of the best performing models from the first round of training. It possibly even had some negative impact.
This is not very surprising, as this set piece does not change much when it comes to how players move on the field, when compared to corners or Free-kicks, which are very specific situations.

Another eliminated feature was the combination of Distance from goal, and Angle to goal. I was trying to determine which is better for the model. If the coordinates (px, py), or the combination of Distance and Angle to the goal. The coordinates ended up being the better indicator for the model.

This left me with the following features:

- Px + Py
- Previous event type
- Time since last event
- Shot after Free kick
- Shot after Corner

These columns would from this point on be always present in the future training cycles. Thus, reducing the number of possible combinations for all my future runs.

3.5.2 Second iteration

In the second test, I would try to give the model more information about the events leading up to the shot.

In my data set, I have information going back up to 3 events into the past, for the following features:

- Time since last event:
- Previous event type:
- Meters per second:
- Previous event same player:
- Previous event same team:

To again address the issue of too many combinations, for this iteration I will only use the features for 2 events into the past.
If some of them prove themselves to be useful, I will add their counterparts of the proven features for the third, fourth and fifth events back, in the next iterations.

I will also include:

- Primary position
- Primary position detailed
- Shot body part type

as features to be tested.

After running the second iteration, the results between “Primary position”, and “Primary position detailed” have been inconclusive, but having at least one of them seems to be valuable information for the model.
There are multiple models which have done well with both Primary position detailed, and Primary position, so I will choose Primary position Detailed moving forward, as it had just little more presence in the top results, and stop testing Primary position altogether.

As for the other features:

- **Time since last event (1-2 events back)**
    - Time since last event seems to be a decent indicator for the indicator, it was present for four of the top 5 models trained.
- **Previous event type (2 events back)**
    - Has been present in all the top performing models.
- **Meters per second** (1-2 events back)
    - Has been present in all the top performing models.
- **Previous event same player (1-2 events back)**
    - This feature for one event back was present in all the top 5 models. For two events back, it was only present for two of the five.
- **Previous event same team (1-2 events back)**
    - This feature has been successful with the information for one event back, but 2 events back seem to not be important for the models performance, It only appeared in one of the top 5 models.

From this we can see that all the information seems to have some relevance for the model, at least in specific combinations.

The features that were present for at least four of the top five models will be added to the models features in the future iterations. This would include:

- Previous event type (2 events back)
- Meters per second (1-2 events back)
- Previous event same player (1 event back)
- Previous event same team (1 events back)
- Time since last event (1 event back)

As for the eliminations, "Previous event same team" seems to not be very relevant to the model even just two events into the past, so I will remove it.
As the previous event same player feature did not fare much better, I will keep the two events back for further testing, but I will not be testing 3 events back going forward.

3.5.3 Third iteration

In the final iteration we are left to test the benefits of the following features:

- Time since last event (3 events back)
- Previous event type (3 events back)
- Meters per second (3 events back)
- Previous event same player (3 events back)

The results of the testing show, that adding the combination of features: Meters per second, and Time since last event, have trained the best performing model, based on the Mean square and R2 values.

Therefore, I will use them as the last two features to add to the model.
This means I have found the features I want to use for the final model.

Now, since I skipped over some possible parameter combinations, I will run the training one more time, with the selected features, but for every possible parameter combination.

And the best training parameters were the following:

- Eta: 0.4
- Num_rounds: 29
- Max_depth: 4

4 Results

4.1 Model performance

I will be comparing the performance of my model, to the model that Wyscout uses in their data.

The data set used for the comparison, is not part of the training set.
The training has been conducted on shots from three European leagues,
Bundesliga(Germany), Premier league(England), and Ligue 1 (France).

The data set used for comparing the model's evaluation to Wyscouts evaluation, will be shots from the last four seasons (2019-2023) of the Spanish "La Liga" league.

Here are the results.

Wyscout:

- Mean Squared error: 0.08150984163618753
- R2: 0.1262127932934366
- Mean Absolute error: 0.16464572738752645

Trained Model:

- mean_sq: 0.08002742794309388
- r2: 0.1421042990805953
- mean_abs: 0.16468648589966053

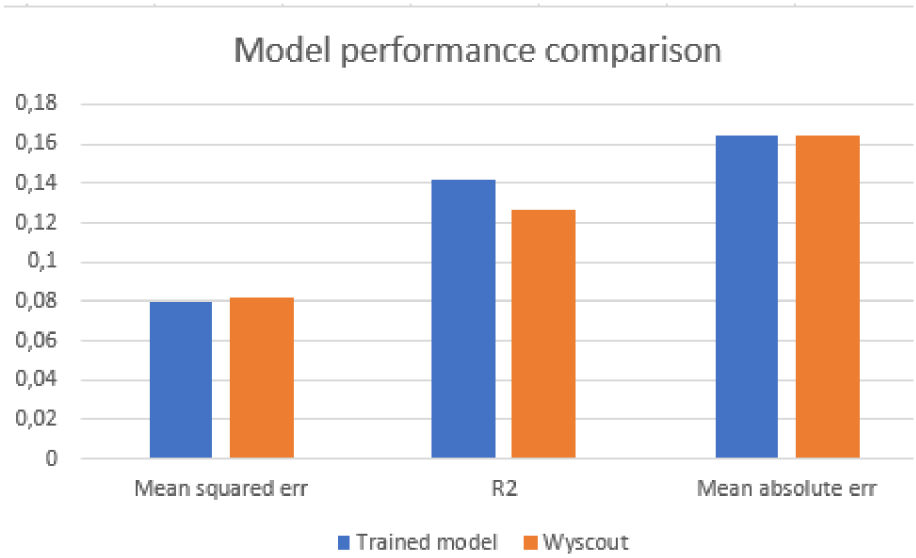

Figure 35: Model performance comparison (Own)

Mean squared error: Smaller is better

R2: Bigger is better.

Mean absolute error: Smaller is better.

When looking at the numbers and the graph, we can see that our model has generally performed better in all the evaluation metrics.

When it comes to Mean squared error, the slightly lower value in my model suggests a slightly better accuracy.

The R2 values indicate, that the trained model explains more of the variance of the data, making it a better fit.

The mean Absolute error however, indicates that the magnitude of errors is very similar for both models.

Another way of evaluating the model's performance, is how the total xG compares to the actual goals scored over a large sample of shots, as that is one of the use cases for the model, to see how many goals "should have" been scored.

Figure 36: Goals vs. Expected goals (Own)

In the figure above, we can see that my model is a lot closer the actual number of goals scored than the Wyscout model.

4.2 Shot map

For an example of the shot map, I have chosen to visualize the shots of Erling Haaland for the current, still on-going, season.
He is an excellent finisher, so it is to be expected for him to overperform what the model will predict.

Haaland has 77 shots(non-penalty) so far, from which he has scored 23 goals. The model predicted only 15,49 xG. This could indicate that the player is exceptionally good at finishing. Here is his shot map:

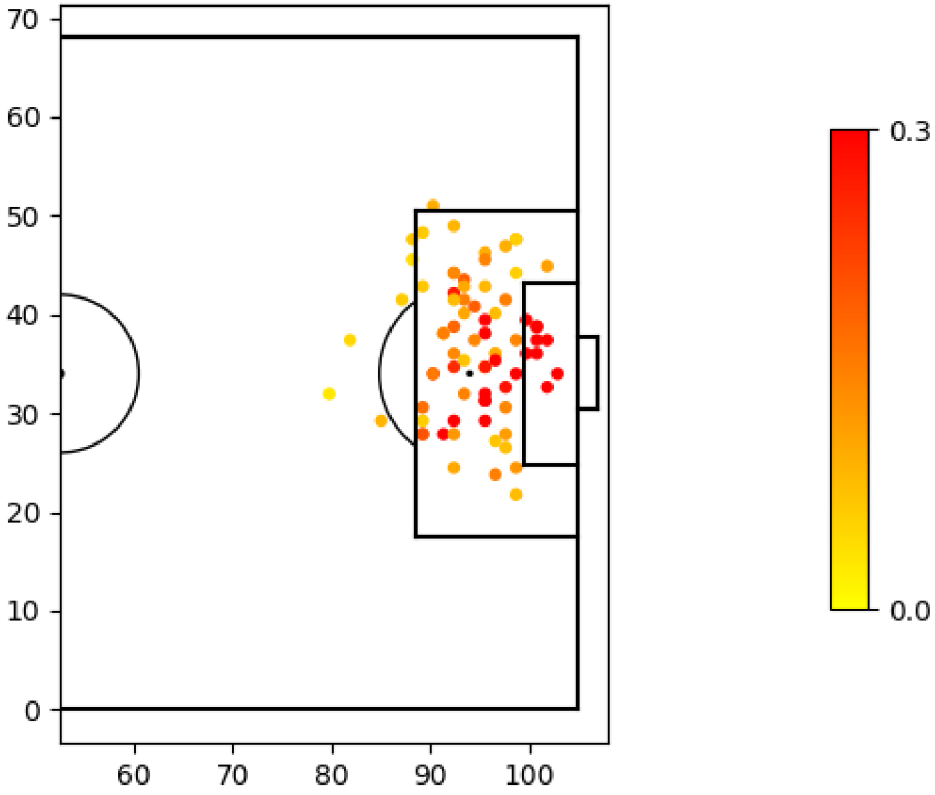

Figure 37: Haaland Shot-map(Own)

The scale next to the pitch shows the colors corelated to the xG value of the shots.

The scale goes from 0.0 to 0.3 xG. This means, the red dots on the pitch are shots, that had a very high probability (around 30%) for scoring a goal, at least according to my model.

5 Conclusion

This thesis aimed to explore the potential of machine learning in football analytics.
The literature review served to present the historical context of data analysis in football, as well as to describe the key machine learning concepts and methodologies employed in the study. The practical part of the thesis was dedicated to implementing a machine learning model that tries to predict the probability of a goal.

The practical component of this work focused on developing and refining our machine learning model. We utilized widely adopted tools and libraries such as Python, XGBoost, Pandas, and Matplotlib to accomplish this goal. Through multiple iterations of training and fine-tuning and trying to find new helpful features for this specific purpose, we found a well performing set of parameters and features.
Our model demonstrated a slight improvement in performance when compared to the Wyscout model, and there are still many more potentially helpful features to be explored for further enhancements of the xG model.

Moreover, our study underlines the value of visualization tools, such as the shot map, for improving the understanding and interpretation of football analytics data. Visualization tools can provide crucial insights for coaches, analysts, and decision-makers in the football industry.

In summary, this thesis has demonstrated that machine learning techniques hold promise for the future of football analytics by offering more accurate and insightful models for analyzing event data. As machine learning becomes more accessible and widely adopted, it is expected that its application in football analytics will continue to grow, ultimately leading to more informed decision-making within the sport. Future research can build upon this foundation by exploring alternative machine learning algorithms, incorporating additional features, like tracking-data, and refining the training process to further optimize performance and unlock the full potential of machine learning in football analytics.

6 References

- 1.9. Naive Bayes — scikit-learn 1.2.2 documentation, [no date]. Online. [Accessed 31 March 2023]. Available from: https://scikit-learn.org/stable/modules/naive_bayes.html?fbclid=IwAR2Qa0X5Pmz0Zz7SJRW0dhOxMFdgEhwM_lIVMwQgKvftE8L-PlfdC293NDw
- A Classification and Regression Tree (CART) Algorithm | Analytics Steps, [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.analyticssteps.com/blogs/classification-and-regression-tree-cart-algorithm?fbclid=IwAR0QbxUPyFpJ73uyHwaxWu6QYnyXKRhYp-XWOj9UlgMtHQRrSQIME3qvTMc
- Evaluating a machine learning model., [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.jeremyjordan.me/evaluating-a-machine-learning-model/?fbclid=IwAR0yij8jP2N6RxBb9FaqjVbisb5uxezMU1EexrbC4fknvK0FDKRR1Nl4DnI
- Explained: Neural networks | MIT News | Massachusetts Institute of Technology, [no date]. Online. [Accessed 31 March 2023]. Available from: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414?fbclid=IwAR1aXu6iF_5X4a-jsIW8iXBCzvFljD7d0sRY1DmKKoV85knkb9re_hqw-uE
- Hyperparameter tuning - GeeksforGeeks, [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.geeksforgeeks.org/hyperparameter-tuning/?fbclid=IwAR1Ztq8bCzNGcBZpuyxlHHhRTDKG1NZMCUfZc7u4l0SwsHWB3KG7elwBFPg
- Hyperparameter tuning for machine learning models., [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.jeremyjordan.me/hyperparameter-tuning/?fbclid=IwAR0pA8lw1UityBeyFAfXxUpaalrnJWDrnHWnbT8uLEu-NUARLZiVAwE5hZ0

- K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint, [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning?fbclid=IwAR0yij8jP2N6RxBb9FaqjVbisb5uxezMU1EexrbC4fknvK0FDKRR1Nl4DnI
- Naive Bayes Classifier in Machine Learning - Javatpoint, [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.javatpoint.com/machine-learning-naive-bayes-classifier?fbclid=IwAR2l0LwQAbXdxS0-huODvvbLPShwhPfulwRCLN8XCUMSiZab0Pa5inqCKyM
- POLLARD, Richard, 2002. Charles reep (1904 – 2002): Pioneer of notational and performance analysis in football. *Journal of Sports Sciences*. Online. 2002. Vol. 20, no. 10, p. 853–855. [Accessed 28 March 2023]. DOI 10.1080/026404102320675684.
- sklearn.feature_selection.RFE — scikit-learn 1.2.2 documentation, [no date]. Online. [Accessed 27 March 2023]. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html
- Support Vector Machine (SVM) Algorithm - Javatpoint, [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm?fbclid=IwAR222JWHFHDp48CpXqSgy2PCp17D-_V-29ekIsd8ka2ak0OCqNK3WTWnLpc
- The history of football analytics - Part 2., [no date]. Online. [Accessed 28 March 2023]. Available from: http://www.xfbanalytics.hu/blog/blog-post/32
- Types of Machine Learning - Javatpoint, [no date]. Online. [Accessed 31 March 2023]. Available from: https://www.javatpoint.com/types-of-machine-learning?fbclid=IwAR0N81PLXg5xJuuJWgoQW6D0aDaduRt9GcCjnt0GdsU5drMJrOvnLAHMDrE
- Upgrading Expected Goals - StatsBomb | Data Champions, [no date]. Online. [Accessed 28 March 2023]. Available from: https://statsbomb.com/articles/soccer/upgrading-expected-goals/

- Various ways to evaluate a machine learning model’s performance | by Kartik Nighania | Towards Data Science, [no date]. Online. [Accessed 31 March 2023]. Available from: https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15
- What are Neural Networks? | IBM, [no date]. Online. [Accessed 28 March 2023 a]. Available from: https://www.ibm.com/topics/neural-networks
- What is Random Forest? | IBM, [no date]. Online. [Accessed 28 March 2023 a]. Available from: https://www.ibm.com/topics/random-forest
- Wyscout xG, [no date]. Online. [Accessed 28 March 2023]. Available from: https://dataglossary.wyscout.com/xg/

7 List of pictures, tables, graphs and abbreviations

7.1 List of pictures

7.2 List of abbreviations

AI - Artificial Intelligence

xG – Expected Goals

ML – Machine Learning

LSTM – Long Shot-term memory

RNN – Recurrent Neural Network

RFE – Recursive Feature Elimination

LASSO – Least Absolute Shrinkage and Selection Operator

GBM – Gradient Boosting Machine

SVM – Support Vector Machines

LR – Linear Regression

MDI – Mean Decrease in Impurity

MDA – Mean Decrease Accuracy

RBF – Radial Basis Functions

ANN – Artificial Neural Network

SNN – Simulated Neural Network

MLP – Multilayer Perceptron

CNN – Convolutional Neural Network

RBFN – Radial Basis Function Network

K-NN – K-Nearest Neighbour

TP – True positive

FP – False Positive

TN – True Negative

FN – False Negative

MAE – Mean Absolute Error

MSE – Mean Squared Error

RMSE – Root Mean Squared Error

IDE – Integrated Development Environment

8 Appendix

Files attached to the thesis:

A Folder named xGModelTraining that includes -

• events – A folder with the training data and the script for loading the shots.

• chosen_model – A folder with the final model

• chose_shots – A folder with a csv of the shots for the selected players shots for the shot-map visualization

• visualizations – A filder that hold all the scripts necessary for drawing the field and shots.

• xg – A folder with the training scripts.

• main.py – Script that runs the different functions.