6CCS3PRJ Final Year Individual Project Report Title

Final Project Report

Author: Mohammad Ibrahim Khan

Supervisor: Name

Student ID: k22013981

March 10, 2025

Abstract

The abstract is a very brief summary of the report's contents. It should be about half-a-page long. Somebody unfamiliar with your project should have a good idea of what your work is about by reading the abstract alone.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Mohammad Ibrahim Khan

March 10, 2025

Acknowledgements

It is usual to thank those individuals who have provided particularly useful assistance, technical or otherwise, during
your project. Your supervisor will obviously be pleased to be acknowledged as he or she will have invested quite a lot of time overseeing your progress.

Contents

Chapter 1

Introduction

??????"""""This is one of the most important components of the report. It should begin with a clear statement of what the project is about so that the nature and scope of the project can be understood by a lay reader. It should summarise everything that you set out to achieve, provide a clear summary of the project's background and relevance to other work, and give pointers to the remaining sections of the report, which will contain the bulk of the technical material."""""????

Since the creation of chess over 1500 years ago [1], most agree that the biggest event in the games history was when IBM's Deep Blue defeated Kasparov [2], the world champion at the time. This was a turning point for chess engines and AI in general. Since then we have seen the rise of chess engines the likes of Stockfish and AlphaZero. Despite our progress in this field, the game is still unsolved. Shannon mentions there is 10^{120} possible positions in chess [3] which is more than the number of atoms in the observable universe. Therefore, it is infeasible to brute force the game and since 1997, there have been attempts to create the perfect chess engine. Solving chess could be a pathway to solve other problems in computer science like Optimisation and Decision-Making.

Machine Learning is at the heart of modern chess engines like ??MENTION WHAT IS USED BY STOCKFISH OR ALPHAZERO AND QUOTE???. The aim of this project is to create a chess engine that uses machine learning techniques that aren't popular within the chess engine field, to explore the potential of these techniques but also to explore what processes can be shared between different machine learning techniques. Chess engines are usually used as a benchmark for advancements in AI, so this project could be a stepping stone for future research.

Minimax is a popular algorithm which is fundemental in chess engines. This, in conjuction

with Alpha-Beta pruning, allows the engine to search through a game tree in order to find the best move. Due to the exponential growth of the game tree, its not feasible to searcht the entire tree. This is where machine learning is used to predict the best move in order to reduce the search space. Many techniques have been used from Neural Networks to Natural Language Process [REFERENCE]. ???GAPS???

This research focusses on how a Naive Bayes Classifier can be implemented in a search algorithm like Minimax to improve positional evaluation in a chess engine. Naive Bayes has been see to be effective in other contexts like spam detection and due to its simplicity and efficiency, it could be ideal for this application. Popular machine learning algorithms used in chess engines are usually very complex and require a lot of computational power however Occams Razor states that simplicity is usually the best option. This research wants to see if this principle holds in this domain.

The standard minimax algorithm is limited by the evaluation function. Usually they only base themselves on piece values. However, chess has more intricacies that may not be seen at face value, including piece positions relative to each other as well as identifying weaknesses. This is where I hope a Naive Bayes classifier can be utilised to try to learn these intricacies better than a standard evaluation function.

1.1 Report Structure

Chapter 2

Background

??????"""The background should set the project into context by motivating the subject matter and relating it to existing published work. The background will include a critical evaluation of the existing literature in the area in which your project work is based and should lead the reader to understand how your work is motivated by and related to existing work. """"????

2.1 Chess

Modern chess is a game that has its origins in India, dating back to the 6th century [REFERENCE] as a way of devising strategy and tactics in war. Today, this game is perceived as a benchmark for skill and intelligence, played by millions. Chess is an ideal candidate for AI research as it is a fully observable game, as both players can see everything related to the game state and the rules are well defined. Also there is no component of chance is the game, therefore the game state is only determined by the each players moves.

2.2 Search Algorithms

The most popular way to design a chess engine is by using a search algorithm. Commonly used is what is known as minimax. The concept of minimax was first proposed by Shannon in 1950 [REFERENCE]. It is used for zero-sum games which are games where if one player wins, the other player loses. The algorithm recursively alternates between the maximising player and the minimising player, until it reaches a terminal node. Then the algorithm backtracks to find the best moves for each player. The algorithm is shown below:

The issue with this algorithm is that the search tree it creates grows exponentially, so

Algorithm 1 Minimax Algorithm

```
function MINIMAX(Node, Depth, MaximizingPlayer)
    if Depth = 0 or Node = Leaf then
        return EVAL(Node)
    end if
    if MaximizingPlayer then
        Value ← −∞
        for each in Node do
            Value ← max(value, MINIMAX(child, depth − 1, false))
        end for
        return Value
    else
        Value ← ∞
        for each Child in Node do
            Value ← min(value, MINIMAX(child, depth − 1, false))
        end for
        return Value
    end if
end function
```

techniques to decrease the search space are used. Alpha-Beta pruning is a one such technique that is used to reduce the number of nodes that need to be evaluated. It does this by ignoring nodes that would not affect the final outcome of the algorithm. It introduces two new values, α and β, where α represents the maximum value that can be attained and β represents the minimum value that can be attained. If the value of a node is less than α or greater than β, then the node is pruned. In best case scenario the algorithm only needs to evaluate $O(b^{m/2})$ nodes [4], where b is the branching factor and d is the depth of the tree compared to $O(b^m)$ nodes with normal minimax. However, in the worst case scenario it doesn't help improve minimax at all.

However, even with alpha-beta pruning, the search space for chess is still too large to evaluate in a reasonable amount of time. This is why a Heuristic Alpha-Beta Tree Search is used, which is where the search is cut off early and apply a heuristic evaluation function to estimate which player is in the winning position. In chess engines what is usually used for this evaluation function is calculating the material balance, where each piece is given a value and the player with the higher value is currently "winning".

The way to improve chess is one of two ways. The first is to increase the depth of the search tree, however this is generally dependant on the computational power of the machine so over time as computational power increases (if Moore's Law still holds) the depth of the search tree can increase. The second way is to improve the evaluation function, which is what this research paper focusses on. The primary way this is done is by using machine learning techniques.

2.3 Naive Bayes

The most well-known chess engines are Stockfish and AlphaZero. AlphaZero, designed by DeepMind, uses a deep neural network in conjunction with reinforcement learning which allows it to teach itself how to play. Initially it has no understanding of the game other than the basic rules, then it plays against itself and uses the result of the game to update parameters in the neural network. Stockfish however uses a more traditional approach, utilising alpha-beta pruning with minimax with a evalutaion function that is based on numerous elements of the game. Recently it has also implemented a neural network to improve its evaluation function. However both these engines require a lot of computational power and also require a large dataset to generalise well, whereas Naive Bayes is a much simpler algorithm that requires less power and can generalise well even with a small dataset. This is the reason why this paper is concentrated on observing the impact of Naive Bayes on chess. This is because it is very simple and makes assumptions that are generally unrealistic, however it has been shown to be effective in domains like spam detection [5] despite the simplistic assumptions. [MENTION SOMEWHERE THAT IS THE WORST CLASSIFIER IN HEAD TO HEAD COMPARISON]

Naive Bayes, sometimes also known as Idiot Bayes or Simple Bayes, is a simple classification algorithm that is based upon Bayes' theorem (Equation 2.1) [6].

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.1}$$

It is referred to as naive as it assumes each input variable $X_1, X_2, ..., X_n$ is conditionally independant given the class. Despite this assumption not being true in most cases, it still performs well in practise. This assumption allows probability distributions to be efficiently represented as the product of the individual probabilities of the the input variables (Equation 2.2) [6].

$$P(X_1, X_2, ..., X_n, C) = P(C) \cdot \prod_{i=1}^{n} P(X_i|C) \tag{2.2}$$

Chapter 3

Literature Review

Chapter 4

Requirements and Specification

4.1 Introduction

TODO????

4.2 Functional Requirements

- The system should be able to play a game of chess
- The system should be able to evaluate the game state
- The system should use a Naive Bayes Classifier during evaluation
- The system should be able to generate moves
- The system should make sure generated moves are legal
- The system should generate moves based on evaluation

4.3 Non-Functional Requirements

- The system should be comptabile with an average device
- The classifier should be able to evaluate in real time
- The system should generate moves in a reasonable time
- The system should be testable against Stockfish
- The classifier should be scalable to utilise large datasets

Chapter 5

Methodology

5.1 Introduction

As mentioned previously, the focus of this paper is to explore the uses of Naive Bayes in chess and how this could impact other domains in computer science. This chapter will mention the methodology that was used to implement the Naive Bayes classifier in the chess engine.

5.2 Random Chess Engine

??????Initially I wrote code to visualise the chess board. This makes it easier to to understand the game state and see what moves my chess engine would make. This makes it easier to understand my engine as well as debugging issues it may have. ????[SHOULD I INCLUDE THIS]

I utilised the python-chess library to as a base to create my chess engine. The reason why I chose to use this library is because it has many features needed. This includes support for FEN notation, methods to get legal moves, and also I chose to use it due to the optimisations it uses including representing the board as a bitboard.

Then I created a chess engine that randomly picks moves. This implementation was very simple due to python-chess' method *board.legalmoves()* which provides all the legal moves available to the current player. Using this list of moves, one is then chosen at random. The purpose of this random engine is be used as a benchmark for the main chess engine to be created.

5.3 MiniMax and Alpha-Beta pruning

Minimax is used by the majority of chess engines. The aim is to implement Naive Bayes to improve the minimax algorithm. So intuitively the first step is implement minimax. Alpha-Beta is much more powerful than basic minimax as it will always give the same output as minimax and is much faster. Therefore, Alpha-Beta will be used directly. This is the pseudocode for the algorithm used. A similar method is use for alphaBetaMin. ????UPDATE pseudocode???

Algorithm 2 Alpha-Beta Pruning Algorithm

```
function AlphaBetaMax(Board, Alpha, Beta, Depth)
    if Depth = 0 or Game Over then
        return Eval(Board), None
    end if
    BestValue ← −∞
    BestMove ← None
    for each Move in legal_moves do
        MakeMove
        Score ← AlphaBetaMin(Board, Alpha, Beta, Depth − 1)[0]
        if Score > BestValue then
            BestValue ← Score
            BestMove ← Move
            Alpha ← max(Alpha, Score)
        end if
        if Score >= Beta then
            break
        end if
    end for
    return BestValue, BestMove
end function
```

When using alpha-beta pruning, a depth of 3 was used. This was due to the limited computational power available. The depth of the search tree is the main thing that determines the performance of the chess engine. The higher the depth, the more possibilities the engine can explore which means each move made would have been given more consideration resulting in a better move.

The other part of the minimax algorithm that is important is the evaluation function. This is what determines the moves that are made. It gives a value to a game state to approximate which player is winning at that point in time as it is infeasible to search the entire game tree. So the evaluation function is used to estimate the winner at a certain point in the game. The evaluation function used for the purpose of this project will be very simple and will be mainly based on material balance. This is a good indicator of the current state of the game as generally the more pieces a player has the more options they have, therefore they are more likely to win.

$$\text{Material Balance} = \text{Number of White Pieces} - \text{Number of Black Pieces} \tag{5.1}$$

However, considering each piece of equal value is not representative of the true value of pieces during the game. For example 1 queen is worth more than 2 pawns. Therefore, this research will use the values in Table 5.1 to calculate the material balance, which are the commonly accepted values for each piece [7].

Piece	**Value**
Pawn	100
Knight	300
Bishop	300
Rook	500
Queen	900
King	0

Table 5.1: Values of Chess Pieces

This evaluation function is relatively strong and incentivises the engine to take pieces when possible and also to protect its own pieces. After testing this evaluation function, as expected, the engine was winning initially by taking pieces. However, it never won against the random engine. After analysing the games played, the problem was in the end game. I realised that the evaluation function did not incentivise the engine to check the opponent, which is the most important part of chess. Therefore, checkmates were considered terminal nodes by giving them a value of $\pm\infty$ dependant on who is winning. A value of 10 was also given to check as this is a very strong move. This was then the ???final??? evaluataion function used.

5.4 Data Collection

The choice of dataset used for the Naive Bayes Classifier was an important task to ensure the classifier had enough data to be trained on but also have good data to be trained on. There were many datasets that I came across. In the end, I chose to use a dataset from kaggle.com [REFERENCE] which had 6.25 million chess games played on lichess.com in July 2016 . This gave me a lot of flexibility as there is a lot of data to train on. There were other datasets like a set of 4 million games on chess.com only played with grandmaster players. However, I wanted to train a model to work for all level of platers so I chose tp go with the lichess dataset.

The lichess dataset was in csv format. These are the features that were part of the dataset:

- **Event:** The type of game.

- **White:** White's ID.
- **Black:** Black's ID.
- **Result:** The outcome of the game (`1--0` if White wins, `0--1` if Black wins and `1/2--1/2` if they draw).
- **UTCDate, UTCTime:** The date and time (UTC) when the game was played.
- **WhiteElo, BlackElo:** ELO of the players.
- **WhiteRatingDiff, BlackRatingDiff:** The change in rating points after the game.
- **ECO:** Opening in ECO (Encyclopaedia of Chess Openings) encoding,
- **Opening:** Opening name.
- **TimeControl:** The time allocated for each player, plus any increment in seconds.
- **Termination:** The reason the game concluded.
- **AN (Algebraic Notation):** The sequence of moves in algebraic notation.

A lot of the features would not be very useful to train the model. The ID's of the players would not be useful to train with so was discarded. The UTCDate, UTCTime also would not be useful as the time of day a game is played wouldn't affect the outcome of the game or to decide the best move to make. The Elo ratings of the players would be useful as players of similar levels may play in similar ways however this also wasn't used as this information won't always be available when using the engine. The way the game terminated is also information that wouldn't be beneficial as all that is important is the final game result. The most important features that was utilised was the result of the game and the sequence of moves in algebraic notation.

The data then needed to be preprocessed. The moves in algebraic notation needed to be converted in a format that can be used by the Naive Bayes Classifier. For each game, I used the python-chess library to simulate the game, so for each move the board is updated. However using every move would be too much data to train on, especially since consecutive moves are highly correlated. Therefore, for most of the game I use every 6 moves however during end game, I use every other move. This is because during the end game, each move will most likely have a bigger impact on the result of the game. For the purpose of this project, end game is after 75% of the moves have been made. We also make sure that the last move is included in the data as well since this is the game state that determined the result of the game.

5.5 Feature Selection

Material balance is the first feature implemented. Arguably this is the most important feature to estimate the current state of a chess game. For the purpose of this project, the values in Table 5.1 were used to calculate the material balance, using Equation 5.2. The material balance is positive if the values of White's pieces are grater than the values of Black's pieces and vice versa.

$$\text{Material Balance} = \text{Number of White Pieces} - \text{Number of Black Pieces} \tag{5.2}$$

Another feature used is the positional value of pieces. The location of specific pieces on the board can influence how effective it is within the game. An example of a piece position table is given in Table 5.2 to calculate the positional value of a knight.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
8	-50	-40	-30	-30	-30	-30	-40	-50
7	-40	-20	0	5	5	0	-20	-40
6	-30	0	10	15	15	10	0	-30
5	-30	5	15	20	20	15	5	-30
4	-30	0	15	20	20	15	0	-30
3	-30	5	10	15	15	10	5	-30
2	-40	-20	0	5	5	0	-20	-40
1	-50	-40	-30	-30	-30	-30	-40	-50

Table 5.2: Positional Value Table for Knight

This table favours the knight to be in the centre of the board rather than the edge. This is because when the knight is in the centre it can control more squares and has more options whereas when it is near the edge, the knight is much more restricted, especially the corners where they have only 2 possible moves. Another example are pawns which are also more valuable in the centre of the board but also in squares that are close to promotion. They are usually not effective in the starting ranks. The positional values of all the black pieces are then summed up and subtracted from the sum of the positional values of the white pieces.

Another feature considered for the Naive Bayes Classifier is piece mobility. Piece mobility is the possible moves a piece can make. Generally the more moves available to a player, the more control they can have over the board. This is calculated as the difference between the number of legal moves White has and the number of legal moves Black has.

King Safety is another feature that is critical to the game. The winning or losing of the game solely lies upon how well you cant protect the king. There are many ways to define king safety. In this project, we will use the amount of pieces that are attacking the king. This is an important factor to consider since, the more pieces attacking the king, the more likely the

player is to lose.

Structure of pawns was also considered as a feature. The structure of pawns can determine how much control the player has, defending its pieces and preventing advancements from the enemy. The two that were considered was isolated pawns and doubled pawns. Isolated pawns are pawns that do not have any friendly pawns on adjacent files. Usually this is a weak structure since they can't be defended by other pawns and also can be easily blocked by the opponent pieces however they can be considered strong in some cases. This is because they can have more control over the board but also some openings use isolated pawns in order to allow more movement for rooks and bishops. Doubled pawns are pawns that are on the same file. Generally this is a weak structure since they are limiting each other's mobility and can generally become isolated. However creating doubled pawns can be used to open up files or diagonals for rooks and bishops.

The last feature considered is the castling rights of the player. Castling is a move that allows the king to move two squares towards a rook. This is a very strong move as it allows the king to move away from the centre which is generally more dangerous. It also allows the rook to have a more active role in the game. Whether the play has castling rights both kingside and queenside are considered.

After collecting all the features, I standardised the data. This is because the features are on different scales so it's important to normalise the data to ensure the Naive Bayes Classifier is trained properly and doesn't give importance to features that are on a larger scale. I utilised the StandardScaler from the scikit-learn library. This is done by the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{5.3}$$

- x: Feature value
- μ: Mean of the feature
- σ: Standard deviation of the feature

This allows all the features to have a mean of 0 and a standard deviation of 1 but also keep the original distribution of the data.

5.6 Naive Bayes Classifier

The Naive Bayes Classifier is a simple classifier that is based on Bayes' Theorem. Bayes' theorem is a fundemental principle that describes how the new evidence can change the probability of an event. Bayes' Theorem is given by the following equation:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{5.4}$$

Where:

- $P(A|B)$: Posterior probability
- $P(B|A)$: Likelihood
- $P(A)$: Prior probability
- $P(B)$: Evidence probability

Posterior probabilty is the probability of event A happening after knowing new evidence B. Likelihood is the probability of evidence B happening given that event A is true. Prior probability is the probabilty of event we intiially know about A before any new evidence.

The first step for implementing Naive Bayes is to calculate the prior probabilities of each class. This is calculated by going through the whole dataset and working out the proportion of each class. This is done by the following formula:

$$P(C) = \frac{N_c}{N} \tag{5.5}$$

Where:

- $P(C)$: Prior probability of class C
- N_c: Number of instances of class C
- N: Total number of instances

The next step is to calculate the likelihood of each feature given the class. Ther are two main type of Naive Bayes classifiers, Gaussian and Multinomial. Multinomial is used for discrete features, calcluating the likelihood based on the count of each feature.

$$P(X|C) = \frac{N_{X,C} + 1}{N_C + V} \tag{5.6}$$

Where:

- $N_{X,C}$: Number of instances of the feature X in class C
- 1: Laplace smoothing constant
- N_C: Number of instances in class C
- V: Number of possible values for X

However, most of the features used in this project are continuous so Guassian Naive Bayes is more ideal.

Chapter 6

Report Body

The central part of the report usually consists of three or four chapters detailing the technical work undertaken during the project. **The structure of these chapters is highly project dependent**. They can reflect the chronological development of the project, e.g. design, implementation, experimentation, optimisation, evaluation, etc (although this is not always the best approach). However you choose to structure this part of the report, you should make it clear how you arrived at your chosen approach in preference to other alternatives. In terms of the software that you produce, you should describe and justify the design of your programs at some high level, e.g. using OMT, Z, VDL, etc., and you should document any interesting problems with, or features of, your implementation. Integration and testing are also important to discuss in some cases. You may include fragments of your source code in the main body of the report to illustrate points; the full source code is included in an appendix to your written report.

6.1 Section Heading

6.1.1 Subsection Heading

Chapter 7

Design & Specification

7.1 Section Heading

Chapter 8

Implementation

8.1 Introduction

The implementation of the Naive Bayes classifier in the chess engine was done in Python. The implementation was done in two parts. The first part was to implement the Naive Bayes classifier and the second part was to implement the classifier in the chess engine. The implementation of the Naive Bayes classifier was done using the scikit-learn library. The implementation of the classifier in the chess engine was done using the python-chess library.

Chapter 9

Legal, Social, Ethical and Professional Issues

Your report should include a chapter with a reasoned discussion about legal, social ethical and professional issues within the context of your project problem. You should also demonstrate that you are aware of the regulations governing your project area and the Code of Conduct & Code of Good Practice issued by the British Computer Society, and that you have applied their principles, where appropriate, as you carried out your project.

9.1 Section Heading

Chapter 10

Results/Evaluation

10.1 Software Testing

10.2 Section Heading

Chapter 11

Conclusion and Future Work

The project's conclusions should list the key things that have been learnt as a consequence of engaging in your project work. For example, "The use of overloading in C++ provides a very elegant mechanism for transparent parallelisation of sequential programs", or "The overheads of linear-time n-body algorithms makes them computationally less efficient than $O(n \log n)$ algorithms for systems with less than 100000 particles". Avoid tedious personal reflections like "I learned a lot about C++ programming...", or "Simulating colliding galaxies can be real fun...". It is common to finish the report by listing ways in which the project can be taken further. This might, for example, be a plan for turning a piece of software or hardware into a marketable product, or a set of ideas for possibly turning your project into an MPhil or PhD.

References

[1] H. A. Davidson, *A Short History of Chess*. Crown.

[2] F.-H. Hsu, "IBM's Deep Blue Chess grandmaster chips," vol. 19, no. 2, pp. 70–81.

[3] C. E. Shannon, "XXII. Programming a computer for playing chess," vol. 41, no. 314, pp. 256–275.

[4] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence, Pearson, fourth edition, global edition ed.

[5] J. J. Eberhardt, "Bayesian Spam Detection," vol. 2, no. 1.

[6] D. Lowd and P. Domingos, "Naive Bayes models for probability estimation,"

[7] A. Gupta, C. Grattoni, and A. Gupta, "Determining Chess Piece Values Using Machine Learning," vol. 12, no. 1.

Appendix A

Extra Information

A.1 Tables, proofs, graphs, test cases, ...

The appendices contain information that is peripheral to the main body of the report. Information typically included in the Appendix are things like tables, proofs, graphs, test cases or any other material that would break up the theme of the text if it appeared in the body of the report. It is necessary to include your source code listings in an appendix that is separate from the body of your written report (see the information on Program Listings below).

Appendix B

User Guide

B.1 Instructions

You must provide an adequate user guide for your software. The guide should provide easily understood instructions on how to use your software. A particularly useful approach is to treat the user guide as a walk-through of a typical session, or set of sessions, which collectively display all of the features of your package. Technical details of how the package works are rarely required. Keep the guide concise and simple. The extensive use of diagrams, illustrating the package in action, can often be particularly helpful. The user guide is sometimes included as a chapter in the main body of the report, but is often better included in an appendix to the main report.

Appendix C

Source Code

C.1 Instructions

Complete source code listings must be submitted as an appendix to the report. The project source codes are usually spread out over several files/units. You should try to help the reader to navigate through your source code by providing a "table of contents" (titles of these files/units and one line descriptions). The first page of the program listings folder must contain the following statement certifying the work as your own: "I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary". Your (typed) signature and the date should follow this statement.

All work on programs must stop once the code is submitted to KEATS. You are required to keep safely several copies of this version of the program and you must use one of these copies in the project examination. Your examiners may ask to see the last-modified dates of your program files, and may ask you to demonstrate that the program files you use in the project examination are identical to the program files you have uploaded to KEATS. Any attempt to demonstrate code that is not included in your submitted source listings is an attempt to cheat; any such attempt will be reported to the KCL Misconduct Committee.

You may find it easier to firstly generate a PDF of your source code using a text editor and then merge it to the end of your report. There are many free tools available that allow you to merge PDF files.