6CCS3PRJ Final Year Individual Project Report Title

Final Project Report

Author: Mohammad Ibrahim Khan

Supervisor: Name

Student ID: k22013981

February 28, 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

February 28, 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

Methodology

4.1 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.

4.2 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 chode 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 avaialble 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.

4.3 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 pseudcode for the algorithm used. A similar method is use for alphaBetaMin.

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.

$$\text{Material Balance} = \text{Number of White Pieces} - \text{Number of Black Pieces} \tag{4.1}$$

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. Therfore, this research will use the values in Table 4.1 to calculate the material balance, which are the commonly accepted values for each piece [7].

Piece	**Value**
Pawn	1
Knight	3
Bishop	3
Rook	5
Queen	9
King	0

Table 4.1: 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. However, closer to the end game the evaluation function doesn't help the engine know how to win. The evaluations function should also give values to end game positions like checkmate. Also the evaluation function should incentivise the engine to check the opponent as well. Therefore we consider checkmates to be terminal nodes and give them a value of $\pm\infty$ dependant on who is winnning as well as a value of 10

Chapter 5

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.

5.1 Section Heading

5.1.1 Subsection Heading

Chapter 6

Design & Specification

6.1 Section Heading

Chapter 7

Implementation

7.1 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 8

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.

8.1 Section Heading

Chapter 9

Results/Evaluation

9.1 Software Testing

9.2 Section Heading

Chapter 10

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.