
CLASIFICADOR POSICIONAL DE VICTORIA EN AJEDREZ

MODELO CLASIFICADOR ENTRENADO CON DATOS
HISTÓRICOS Y CARACTERÍSTICAS POSICIONALES A
PARTIR DE UNA APERTURA COMPLETA.

PRESENTA:

ING. ALEJANDRO NOEL HERNÁNDEZ GUTIÉRREZ

TRABAJO DE OBTENCIÓN DE GRADO



ITESO

INSTITUTO TECNOLÓGICO DE ESTUDIOS
SUPERIORES DE OCCIDENTE

MAESTRÍA EN CIENCIA DE DATOS

NOVIEMBRE 2023

Abstract

Reading chess books can be overwhelming, especially for beginner players, and the existing web resources are often paid tools or designed for passive study in which there are multiple options (it has already been mentioned that there are more than 1,327 variants). of openings) and its analysis is difficult and exhaustive.

From a positional chess perspective, these opening explorers lack quantitative information about the quality of the position in each move and the probability of victory for each player (counts and percentages are shown based on history, but not positional, speaking clearly of these opening explorers).

This is where this work becomes important, since it aims to extract quantitative characteristics of the positional advantage achieved in each movement, as well as classify victory, draw or defeat of the current position and future scenarios or positions.

Agradecimientos

A mis padres y maestros.

Contenido

Abstract	2
Agradecimientos.....	3
Lista de Imágenes	6
Lista de tablas.....	7
Introducción	8
Descripción del problema	14
Objetivos específicos	15
Revisión de literatura	16
Lasso corresponde a la abreviación de Least Absolute Shrinkage and Selection Operator,	16
Redes Neuronales.....	17
Distribución de los Datos:.....	18
Árboles de decisión	18
Datos y metodología	20
Obtención, descripción y análisis exploratorio de los datos	20
Análisis de valores atípicos.....	27
Análisis de la asimetría de datos	28
Criterio de Correlación	29
Desarrollo de prototipo.....	31
Extracción de datos	31
Tratamiento de valores faltantes y limpieza.....	31
Codificación de variables categóricas.....	31
Ingeniería de características.....	32
Casillas controladas por piezas.....	33
Diagonales controladas	34
Columnas y filas controladas.....	35
Puntos de presión.....	36
Modelos alimentados con datos históricos y nuevas características	37
Máquina de soporte vectorial	37
Xgboost.....	37
Regresión logística.....	37
Reca 0.8565527065527065	38
Resultados y análisis.....	39

Discusión	40
Bibliografía	41
Apéndices	42

Lista de Imágenes

Ilustración 1. Tablero de ajedrez. Recuperado de chegg.com	8
Ilustración 2. Tablero tradicional de ajedrez. Recuperada de chessfox.com	8
Ilustración 3. Ajedrez como arbol de decisión, ejemplo extraido de Research Gate	10
Ilustración 4. Explorador de aperturas Chesstempo	11
Ilustración 5. Chess position trainer.	11
Ilustración 6. ChessBase Reader 2017. Recuperado de chessbase.com	12
Ilustración 7. Explorador de partidas de grandes maestros, recuperada de chess.com	13
Ilustración 8. Explorador de aperturas, recuperada de 365chess.com	13

Lista de tablas

Tabla 1. Descripción del conjunto de datos..... 20

Introducción

El ajedrez es un juego popular y emocionante en el que personas o máquinas participan en enfrentamientos de dos jugadores. Una partida de ajedrez cuenta con 16 piezas que inician en una posición sobre un tablero de 8x8 casillas y se mueven a través de un tablero con reglas bien definidas, por tanto, es un juego determinista en el que el único elemento estocástico es el color que con el que cada participante jugará que comúnmente se decide por azar. Por lo tanto, cuando un rey está bajo ataque o en «Jaque» debe escapar.

64	63	62	61	60	59	58	57
49	50	51	52	53	54	55	56
48	47	46	45	44	43	42	41
33	34	35	36	37	38	39	40
32	31	30	29	28	27	26	25
17	18	19	20	21	22	23	24
16	15	14	13	12	11	10	9
1	2	3	4	5	6	7	8

Ilustración 1. Tablero de ajedrez. Recuperado de chegg.com

Según Bobby Fisher (1972) Gran Maestro estadounidense en su libro *Bobby Fisher teaches chess*, el objetivo del ajedrez es atacar el rey del enemigo en una forma tal que no pueda escapar de su captura. Una vez que la captura se ejecuta, el rey entra en «jaque mate» y el juego se termina.



Ilustración 2. Tablero tradicional de ajedrez. Recuperada de chessfox.com

Por consenso de la comunidad, conceptualmente una apertura de ajedrez se divide en tres fases: apertura, medio juego y final.

De forma sencilla, la apertura es el nombre que recibe una serie de movimientos en un juego, no existe y según Di Luca (2021) puede constar desde un movimiento hasta alrededor de 25.

Muchas aperturas en el ajedrez se conocen como “aperturas estándar”. Lo que significa que han sido jugadas y estudiadas por miles de jugadores a través de la historia. Cada apertura producida genera cierta ventaja o equilibrio posicional entre las piezas que defienden y las que atacan al oponente.

Las aperturas son tan importantes que muchos jugadores pasan años de sus vidas simplemente estudiando aperturas (Di Luca, 2012).

Según Di Luca, en el artículo ya citado existen razones por las que las aperturas son muy importantes en el ajedrez. Algunas de ellas son:

La posición inicial del tablero. La apertura está destinada a ayudar a colocar en buena posición las piezas en el tablero y principalmente establecerlas para controlar o amenazar el centro del tablero o la mayor cantidad posible de casillas cruciales y darle forma así a la batalla que los jugadores que participan en la apertura quieren llevar.

Tomar el control del juego. Cuando un principiante se enfrenta a un jugador que conoce aperturas de ajedrez su recurso suele ser reaccionar al oponente que entiende conceptualmente los movimientos que está haciendo, sus objetivos estratégicos y posicionales. Como en la apertura participan de forma secuencial siempre dos jugadores, entre más aperturas y variantes conozcan más posibilidades tiene un jugador de controlar el tipo de juego que quiere tener.

Evitar trucos estudiados que te pueden dejar en seria desventaja. Las aperturas, como ya se estableció, son secuencias de movimientos bien estudiadas a lo largo del tiempo y a veces esconden ventajas estratégicas, si un jugador ignora los planes o ventajas de una apertura puede caer en trampas que arruinen su posición, le cuesten piezas clave o incluso perder la partida.

Prepararse para el medio juego. Después de la apertura, la siguiente etapa del ajedrez se llama medio juego y surge cuando termina de adoptarse una posición estudiada y comienzan movimientos más caóticos con el objetivo siempre de llegar a la etapa final del juego en la que se da el jaque mate. El lector puede imaginar una apertura de ajedrez como un árbol de decisión en el que cada decisión es un tablero de ajedrez y las diferentes respuestas son cada rama del árbol. Entre más se avanza en una partida más ramas y posibilidades existen por lo que una buena apertura es importante para tener opciones sólidas:

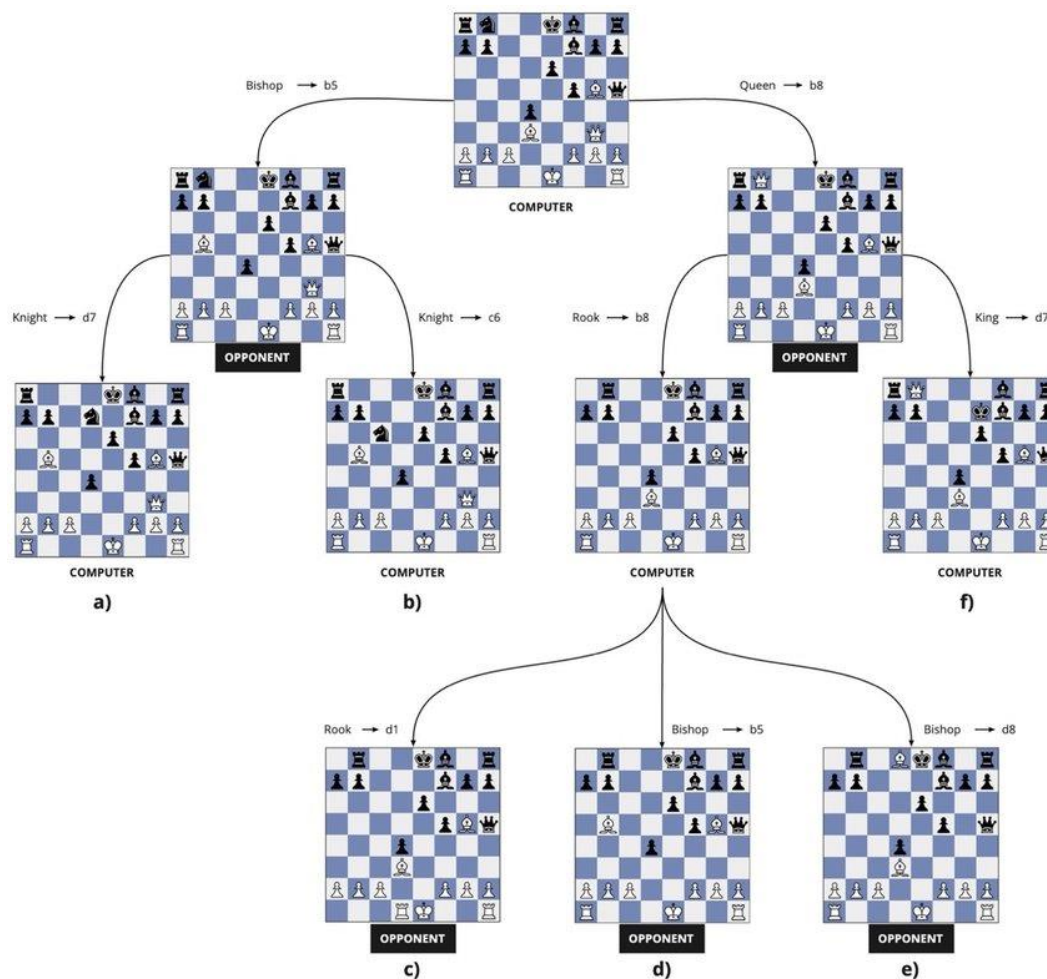


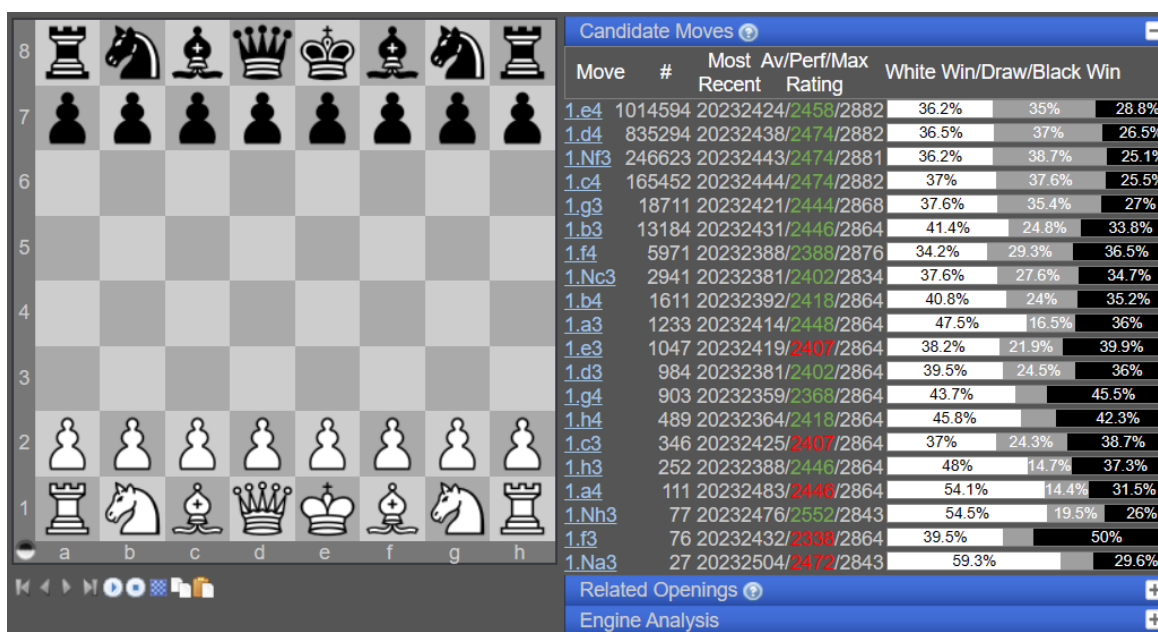
Ilustración 3. Ajedrez como árbol de decisión, ejemplo extraído de Research Gate

Según The Oxford Companion to Chess (1984), que es un catálogo de conceptos de ajedrez, hasta el año de publicación existían alrededor de 1327 aperturas o variantes de aperturas, entonces no es difícil imaginar la dificultad que enfrentan los jugadores que comienzan a jugar de forma habitual y cuando llegan a cierto nivel se enfrentan con oponentes que conocen más aperturas y entienden sus ventajas.

Existen innumerables libros y recursos web para que los jugadores aprendan las aperturas. En lo que respecta a los libros, la metodología de enseñanza que siguen la gran mayoría (por nombrar algunos *El ajedrez. Aprender y progresar*, de Karpov; *Ajedrez lógico jugada a jugada*, de Irving Chernev; *Chess Openings for Black/White explained*, del GM Lev Alburt; *Learn Chess Tactics*, de John Nunn; *The complete idiot's guide to chess*, de Patrick Wolff; *A first book of morphy*, de Frisco del Rosario; *Back to basic tactics*, de Dan Heisman) consiste en mostrar una secuencia de ilustraciones comentadas en la que se presentan partidas de forma secuencial y los autores van narrando las ventajas de cada posición alcanzada. Esto suele ser abrumador para jugadores principiantes que apenas conocen las reglas o tienen poca experiencia.

Dentro de los recursos web existen algunas herramientas gratuitas o de paga entre las cuales figuran:

Chess tempo base de datos (gratuito o de paga según características):

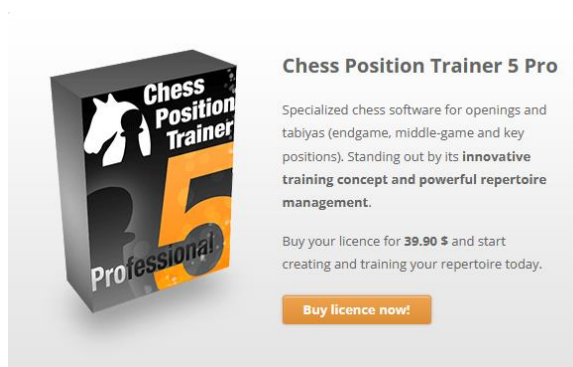


Move	#	Most Recent	Av/Perf/Max Rating	White Win	Draw	Black Win
1.e4	1014594	20232424/2458	2882	36.2%	35%	28.8%
1.d4	835294	20232438/2474	2882	36.5%	37%	26.5%
1.Nf3	246623	20232443/2474	2881	36.2%	38.7%	25.1%
1.c4	165452	20232444/2474	2882	37%	37.6%	25.5%
1.g3	18711	20232421/2444	2868	37.6%	35.4%	27%
1.b3	13184	20232431/2446	2864	41.4%	24.8%	33.8%
1.f4	5971	20232388/2388	2876	34.2%	29.3%	36.5%
1.Nc3	2941	20232381/2402	2834	37.6%	27.6%	34.7%
1.b4	1611	20232392/2418	2864	40.8%	24%	35.2%
1.a3	1233	20232414/2448	2864	47.5%	16.5%	36%
1.e3	1047	20232419/2407	2864	38.2%	21.9%	39.9%
1.d3	984	20232381/2402	2864	39.5%	24.5%	36%
1.g4	903	20232359/2368	2864	43.7%		45.5%
1.h4	489	20232364/2418	2864	45.8%		42.3%
1.c3	346	20232425/2407	2864	37%	24.3%	38.7%
1.h3	252	20232388/2446	2864	48%	14.7%	37.3%
1.a4	111	20232483/2446	2864	54.1%	14.4%	31.5%
1.Nh3	77	20232476/2552	2843	54.5%	19.5%	26%
1.f3	76	20232432/2338	2864	39.5%		50%
1.Na3	27	20232504/2472	2843	59.3%		29.6%

Ilustración 4. Explorador de aperturas Chesstempo

Es un explorador de aperturas en el que el jugador puede ver el número de veces que se ha jugado un movimiento, el porcentaje de veces que en partidas que jugaron ese movimiento ganaron blancas, negras o empate, el nombre de las aperturas relacionadas entre otras cosas (Chess Tempo , 2023).

Chess position trainer (de paga) es un software de escritorio que permite acceder a una base de datos de aperturas y entrenarlas una vez que fueron escogidas (Chess position trainer, 2019).



Chess Position Trainer 5 Pro

Specialized chess software for openings and tabiyas (endgame, middle-game and key positions). Standing out by its **innovative training concept and powerful repertoire management**.

Buy your licence for **39.90 \$** and start creating and training your repertoire today.

Buy licence now!

Ilustración 5. Chess position trainer.

Chessbase Reader 2017 (de paga) es un lector de partidas de ajedrez que muestra bases de datos de partidas y puede leer múltiples tipos de archivo (.cbh, .cbf, .pgn), usualmente usado por grandes maestros para sus análisis (ChessBase Reader, 2017).

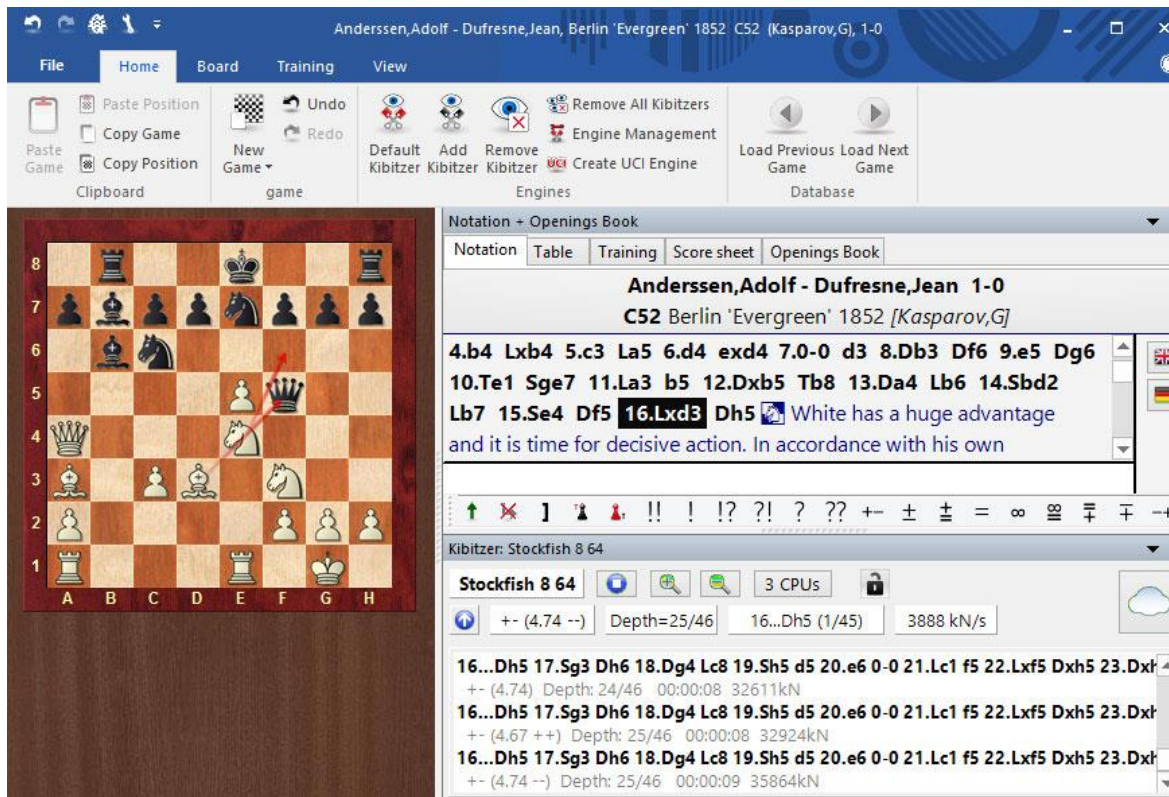


Ilustración 6. ChessBase Reader 2017. Recuperado de chessbase.com

Chess Explorer, de la plataforma online chess.com (de paga) Es un explorador de partidas de grandes maestros similar al ofrecido por chesstempo, pero que es actualizado en todo momento con partidas de grandes maestros en la misma plataforma. Para acceder a este explorador hay que pagar una membresía mensual (Chess.com, 2017).



Ilustración 7. Explorador de partidas de grandes maestros, recuperada de chess.com

365 Chess Opening explorer, similar a los exploradores de chesstempo y chess.com. Es posible explorar aperturas iniciando con una posición FEN o desde la posición inicial (365 chess, 2023).



Ilustración 8. Explorador de aperturas, recuperada de 365chess.com

Descripción del problema

Establecidas las condiciones descritas en la introducción, es posible entender de forma general la importancia de las aperturas en el ajedrez y las opciones que tienen los jugadores para aprenderlas, pero ¿A qué problemáticas busca contribuir el presente documento? ¿En qué se centrará este trabajo de obtención de grado?

Como ya fue establecido, la lectura de libros de ajedrez puede resultar abrumadora sobre todo para jugadores principiantes y los recursos web existentes son herramientas en muchas ocasiones de paga o diseñadas para un estudio pasivo en el que se cuenta con múltiples opciones (ya se mencionó que existen más de 1327 variantes de aperturas) y es difícil y exhaustivo su análisis.

Desde una perspectiva de ajedrez posicional, estos exploradores de apertura carecen de información cuantitativa sobre la calidad de la posición en cada movimiento y la probabilidad de victoria para cada jugador (se muestran conteos y porcentajes basados en históricos, pero no posicionales, hablando claro de estos exploradores de aperturas).

Aquí es donde este trabajo toma importancia, ya que pretende extraer características cuantitativas de la ventaja posicional alcanzada en cada movimiento, así como clasificar victoria, empate o derrota de la posición actual y escenarios o posiciones futuras.

Objetivos específicos

Entrenar un modelo base que considere un número considerable de partidas históricas (20058 datos) y clasifique victoria de negras o blancas o empate.

Por medio de ingeniería de características, codificar funciones que extraigan los siguientes atributos cuantificables sobre la posición

- Casillas controladas por peones, caballos, alfiles, torres, reina y rey.
- Puntos de presión con más de una pieza atacando.
- Diagonales controladas por el jugador en turno.
- Columnas verticales y horizontales controladas por el jugador en turno.
- Evaluación cuantitativa de la estructura defensiva alrededor del rey.
- Número de piezas vulnerables
- Número de piezas defendidas por más de una pieza
- Número de casillas de escape del rey
- Número de piezas defendiendo al rey
- Amenazas potenciales que apuntan o que en 1 turno podrían apuntar al rey
- Valor del total de las piezas en el tablero, potenciando las que atacan puntos de presión.

Aplicar las funciones creadas a las 20058 observaciones de partidas de ajedrez y convertirlas en nuevas *features* de cada observación. Entrenar un modelo de aprendizaje supervisado que considere las partidas históricas y sus nuevas características.

Diseñar un sistema de entrenamiento que permita a un jugador elegir una apertura a estudiar y, utilizando la base de datos de partidas históricas, realizar con la máquina movimientos al azar para que se juegue de forma aleatoria cualquier variante y el usuario obtenga una clasificación de victoria, empate o derrota; así como un despliegue de características posicionales en cada movimiento. Una vez alcanzada una apertura, permitir que la máquina tome decisiones con la inteligencia artificial stockfish para que el usuario pueda seguir entrenando el juego obteniendo la misma información que en la fase de apertura.

Implementar un modelo que, alcanzada cada posición, estudie con una profundidad específica las vertientes en las que usuario y máquina pueden entrar y brinde una calificación a cada movimiento que el usuario escoja y la probabilidad de victoria que conlleva el análisis de profundidad.

Constituir así un sistema didáctico y gratuito para el entrenamiento de aperturas.

Revisión de literatura

En este proyecto se tiene una base de datos en la que el objetivo es realizar clasificación, se aplican los algoritmos de máquinas de soporte vectorial (SVM) y regresión logística. Las SVM tienen como objetivo encontrar un hiperplano que separe el conjunto de datos de la mejor manera posible, esto siempre y cuando la separación sea fácilmente lineal; en caso de que la separación no sea evidentemente lineal se usa la función kernel para transformar las características de las observaciones y asignarlas a un espacio dimensional diferente que permita hacer ajustes a diferentes situaciones. Existen diferentes tipos de kernel, en este caso se realizaron pruebas con el kernel lineal, polinomial y radial.

Kernel lineal: puede ser utilizado como el producto normal de dos observaciones. El producto entre dos vectores es la suma de la multiplicación de cada par de valores de entrada. Una función kernel lineal es recomendable si la separación lineal de los datos es sencilla.

$$K_{x,x_i} = \sum x x_i$$

Kernel polinomial: es una forma más generalizada del núcleo lineal. Considera tanto las características dadas por las muestras de entrada para determinar su similitud, como también las combinaciones de éstas. Con “n” características originales y “d” grados de polinomio produce n^d características expandidas.

$$K_{x,x_i} = 1 + \sum x x_i^d$$

Kernel de base radial o RBF: recibe también el nombre de kernel gaussiano. RBF puede mapear un espacio de entrada en un espacio dimensional infinito. Gamma es un parámetro que va de 0 a 1. Un valor más alto de gamma se ajusta perfectamente en el conjunto de datos de entrenamiento, lo que provoca un ajuste excesivo. Gamma igual a 0.1 se considera un buen valor por defecto.

$$K_{x,x_i} = e^{-\gamma \sum x - x_i^2}$$

Corresponde a una proyección en un espacio de dimensiones finitas. Se caracteriza porque

$$K_{x,x_i} = e^0 = 1 \quad \forall x \in X, \text{ luego } \|x\| = 1$$

También sucede que $k_{x,z} > 0$ para todo $x, z \in X$ como consecuencia de esto el ángulo entre cualquier par de imágenes es menor que 2, de tal forma que las imágenes de los vectores del espacio de entrada caen dentro de una región restringida del espacio de características.

El segundo de los algoritmos utilizados es la regresión logística, este es similar al modelo de regresión lineal, pero tiene la característica esencial de funcionar para variables dicotómicas, hay que considerar la necesidad de realizar un ajuste en los valores para poder aplicar regresión logística. Se aplica la regularización Lasso como un método de reducción y selección de variables. En la regresión logística se aplica la regularización Lasso.

Lasso corresponde a la abreviación de Least Absolute Shrinkage and Selection Operator, que se traduce como operador de selección y contracción mínima absoluta. Se trata de un método de reducción y selección de variables para la interpretación del modelo de regresión lineal. Es muy aplicada en el aprendizaje automático.

La regularización Lasso tiene como objetivo obtener el subconjunto de predictores que minimice el error de predicción para una variable de respuesta cuantitativa. Hace esto al imponer una variable

de restricción en los parámetros del modelo que hace que los coeficientes de regresión de algunas variables se reduzcan a cero.

En cuanto a la segunda parte del proyecto se tienen en cuenta las características e idoneidad de las redes neuronales y de los árboles de decisión para el presente trabajo. Se presenta a continuación lo más relevante en cuanto a la teoría de estos dos tópicos.

Redes Neuronales.

Una red neuronal es un método usado en ciencia de datos que forma parte de la inteligencia artificial, tiene como objetivo enseñar a las computadoras a procesar datos de una manera similar a la forma en que el cerebro humano procesa la información. Se trata de un algoritmo de machine learning llamado aprendizaje profundo, que utiliza los nodos o las neuronas interconectados en una estructura de capas haciendo similitud con el cerebro humano.

Una neurona es el elemento esencial de cualquier red neuronal. Al combinar múltiples neuronas, se puede obtener una red neuronal con capacidad de realizar regresión y clasificación.

Esto es posible en gran parte al uso de una función de en cada una de las neuronas de la red, es importante recordar que esta función debe satisfacer una característica esencial: su comportamiento debe ser no lineal.

Una función de activación es una función matemática de la forma $f(x)$, que se agrega a una red neuronal artificial para ayudar a la red a que aprenda patrones complejos en los datos.

Seleccionar la función de activación es importante debido a que la derivada de la función de activación determina la magnitud del gradiente al realizar la propagación hacia atrás. Si la forma de la función de activación es muy plana en los extremos, la derivada en esos puntos será pequeña y por lo tanto el gradiente será cercano a cero. Este fenómeno se conoce como saturación de gradiente. Esto es un problema en particular cuando las redes tienen varias capas de profundidad puesto que puede ocasionar el desvanecimiento del gradiente.

Al inicio de las redes neuronales se utilizó la función de activación sigmoide debido a que mapea el dominio de los reales al intervalo (0,1), con lo cual se asemeja a una distribución de probabilidad.

$$x = \frac{1}{1 + e^{-z}}$$

$$x' = x(1-x)$$

La función de activación de la tangente hiperbólica se puede considerar como una versión escalada y desplazada de la función sigmoide.

$$\tanh x = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$x = \frac{1 - \tanh x}{2}$$

Otra función de activación, que fue la que se aplicó en este trabajo es la función ReLU. ReLU significa Rectified Linear Unit. Es muy rápida de calcular y tiene efectividad en varios dominios de aplicación. Se define como:

$$\text{ReLU}x = \max(0, x)$$

$$\text{ReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

ReLU permite mitigar el problema de la saturación de gradiente cuando los valores de la combinación lineal de la entrada son mayores a 0 permitiendo una propagación hacia atrás efectiva incluso en redes profundas.

Otras funciones de activación son:

Softmax: Utilizada en la capa de salida para clasificación multiclase. Convierte las salidas en probabilidades. Bueno para: Capa de salida en clasificación multiclase.

Leaky ReLU: Una variante de ReLU que permite un pequeño gradiente cuando la unidad no está activa. Bueno para: Evitar el problema de las neuronas "muertas" en ReLU.

Swish: Una función de activación basada en investigaciones recientes que a veces puede superar a ReLU en términos de rendimiento. Bueno para: Experimentación en problemas donde ReLU no funciona bien.

ELU (Exponential Linear Unit): Similar a Leaky ReLU, pero con una aproximación exponencial para valores negativos. Bueno para: Problemas donde se requiere una convergencia rápida y se quiere evitar el problema de neuronas muertas.

Criterios para Elegir una Función de Activación

Tipo de Problema (Clasificación vs Regresión):

Para clasificación binaria, hay que considerar usar 'sigmoide' en la capa de salida.

Para clasificación multiclase, 'softmax' en la capa de salida es la opción estándar.

Para regresión, a menudo no se usa ninguna función de activación en la capa de salida (o se usa una función lineal).

Manejo del Gradiente Desvaneciente:

Si la red es profunda y hay gradientes desvanecientes, ReLU o sus variantes (Leaky ReLU, ELU) pueden ser útiles.

Distribución de los Datos:

Si los datos están normalizados y centrados en cero, 'tanh' podría ser una buena opción.

Experimentación:

A menudo, la elección de la función de activación puede depender de la experimentación y la optimización específica para el problema y los datos.

Árboles de decisión

Es una técnica fácilmente interpretable, presenta estabilidad y generalmente tiene buenas coincidencias, se aplica para regresión y para clasificación, aunque no es tan recomendable para

regresión, tiene más aplicaciones en clasificación. Es un algoritmo de Machine Learning que fue registrado por Leo Breiman y Adele Cutler.

Un árbol consiste básicamente en una serie de preguntas en las que la respuesta llevará a una decisión final que permitirá la clasificación.

El concepto de Random Forest hace alusión a un conjunto de árboles, es posible, de acuerdo con el problema que se use más de un árbol, pueden llegar a ser varios, de tal forma que en conjunto permitirán una mejor decisión a través de un entrenamiento que tenga una mejor validación cruzada. La validación cruzada permite ajustar el número de árboles y dicho número se considera a su vez un parámetro a considerar en el análisis correspondiente.

Cada uno de los árboles se entrena con un subconjunto de la base de datos y arroja un resultado. Posteriormente se combinan los resultados de todos los árboles para dar una respuesta final. Cada árbol aporta una parte de la respuesta, por decirlo de alguna manera, es como si en cada árbol se realizar un voto y al final gana la mayoría de los votos para realizar la clasificación de interés. Este procedimiento recibe el nombre de bagging.

Bagging: consiste en dividir los datos en subconjuntos aleatorios; en cada uno de esos subconjuntos se entrena un modelo; finalmente se combinan todos los resultados de los modelos para obtener un resultado final. La intención es que el modelo final sea robusto aunque cada uno de los árboles en lo individual no sea tan robusto.

Los árboles de decisión buscan la mejor forma de hacer los subconjuntos de datos, Para validar su eficacia se utilizan métricas, como la impureza de Gini, la ganancia de información o el error cuadrático medio (MSE), para evaluar la calidad de la división de los datos.

Los árboles de decisión son algoritmos comunes de aprendizaje supervisado, entre los problemas que pueden presentar es que puede haber sesgos y sobreajuste. Una forma de solucionarlo es hacer varios árboles esto recibe el nombre de random forest, **cuando varios árboles de decisión forman un conjunto en el algoritmo de random forest, predicen resultados más precisos**, especialmente cuando los árboles individuales no están correlacionados entre sí.

Los algoritmos de random forest **tienen tres hiperparámetros principales**, que deben configurarse antes del entrenamiento:

Tamaño del nodo

Cantidad de árboles

Cantidad de características muestreadas

Entre los desafíos del random forest está el costo computacional, puede ser un algoritmo lento para procesar grandes cantidades de datos y que son muchos árboles. Otro de los problemas que puede presentarse es que mientras más árboles es más complejo realizar la interpretación adecuada, por ello hay que elegir muy bien las variables que formarán parte del árbol y del random forest.

Datos y metodología

Obtención, descripción y análisis exploratorio de los datos

La base de datos cuenta en total con 17 columnas y 20058 observaciones, se tienen variables de tipo entero, objeto y booleanos.

Las características de las variables se describen más a detalle en la siguiente tabla:

Tabla 1. Descripción del conjunto de datos.

Nombre	Tipo	Valores faltantes	Valores únicos	Valor mínimo	Valor máximo
game_id	Entero	no	20058	1	20058
rated	Booleano	no	2	0	1
turns	Entero	no	211	1	349
victory_status	Objeto	no	4	Draw	Resign
winner	Objeto	no	3	Black	White
time_increment	Objeto	no	400	0+12	90+8
white_id	Objeto	no	9438	--jim--	zzzimon
white_rating	Entero	no	1516	784	2700
black_id	Objeto	no	9331	-0olo0-	zztopillo
black_rating	Entero	no	1521	789	2723
moves	Objeto	no	18920		
opening_code	Objeto	no	365	A00	E98
opening_moves	Entero	no	23	1	28
opening_fullname	Objeto	no	1477	Alekhine Defense	Zukertort Opening: Wade Defense
opening_shortcode	Objeto	no	128	Alekhine Defense	Zukertort Opening
opening_response	Objeto	18851	3	No disponible	No disponible
opening_variation	Objeto	5660	615	No disponible	No disponible

A continuación, se presenta la descripción de las variables:

1. `game_id`: Identificador único para cada partida de ajedrez.
2. `rated`: Indica si la partida está clasificada (True) o no clasificada (False). En los juegos de ajedrez en línea, las partidas clasificadas afectan el rating de los jugadores.
3. `turns`: Es el número total de movimientos realizados en la partida, en este caso va desde 1 hasta 349.
4. `victory_status`: Indica el estado de la victoria una vez concluida la partida. Puede tomar diferentes valores, en el caso de esta base de datos dichos valores son "mate" (jaque mate), "resign" (rendición), "outoftime" (agotamiento de tiempo), "draw" (empate).
5. `winner`: Indica el jugador que ganó la partida. En la base de datos están presentes los valores "white" (blanco), "black" (negro) o "draw" (empate).
6. `time_increment`: Incremento de tiempo en segundos después de cada movimiento.
7. `white_id`: Identificación única del jugador blanco.
8. `white_rating`: Rating del jugador blanco en el momento de la partida.
9. `black_id`: Identificación única del jugador negro.
10. `black_rating`: Rating del jugador negro en el momento de la partida.
11. `moves`: Lista de movimientos realizados en la partida.
12. `opening_code`: Código que identifica la apertura de ajedrez jugada.
13. `opening_moves`: Número de movimientos iniciales realizados en la apertura.
14. `opening_fullname`: Nombre completo de la apertura.
15. `opening_shortname`: Nombre abreviado de la apertura.
16. `opening_response`: Respuesta a la apertura por parte del oponente.
17. `opening_variation`: Variación específica de la apertura.

En cuanto a los valores faltantes, destacan las variables que se encuentran en las dos últimas columnas, `opening_response` y `opening_variation` en las cuales faltan 18851 y 5660 respectivamente.

En un primer momento, en cuanto a las columnas a considerar para el análisis, se tomarán en cuenta las que sean más relevantes para clasificar las observaciones como corresponde a la variable `winner`, es decir, `winner` es la variable de salida, considerando a las demás como las variables que van a influir en el resultado correspondiente a la variable dependiente.

Las columnas que son candidatas para ser eliminadas en un primer momento y antes de realizar algún análisis son: las dos últimas "`opening_response`" y "`opening_variation`" dada la cantidad de datos faltantes; la columna de "`moves`", la cual contiene la secuencia de movimientos realizados en la partida de ajedrez y tiene extensión muy variable además de que representa dificultad para ser codificada; las columnas "`white_id`" y "`black_id`" ya que para ganar no resulta relevante el número de identificación.

En ajedrez, la "apertura" hace referencia a la fase inicial del juego en la cual los jugadores mueven sus piezas para desarrollar sus posiciones y prepararse para la parte intermedia y el final del juego. Comienza desde el primer movimiento y generalmente termina después de aproximadamente 10 a 20 movimientos realizados por cada jugador.

La elección de la apertura es una decisión estratégica muy importante en el ajedrez, esto se debe a que puede influir en el desarrollo futuro del juego. Cada movimiento de apertura tiene un nombre específico y se caracteriza por una secuencia particular de movimientos de apertura.

En el contexto de la base de datos "Online chess match prediction", las variables "opening_fullname", "opening_shortname" y "opening_variation" se refieren a diferentes aspectos del movimiento de apertura utilizada en una partida de ajedrez. Aquí las diferencias entre ellas:

1. opening_fullname (Nombre completo de la apertura):

Proporciona el nombre completo de la apertura utilizada en la partida, al inicio. Por ejemplo, si se jugó la "Apertura Española", esta variable contendría el nombre completo de esa apertura.

2. opening_shortname (Nombre abreviado de la apertura):

Proporciona una forma abreviada del nombre de la apertura. Es una versión concisa del nombre completo. Por ejemplo, para la "Apertura Española", el nombre abreviado podría ser "Ruy López".

3. opening_variation (Variación específica de la apertura):

Indica la variación específica de la apertura jugada en la partida. La apertura puede tener varias variantes, y esta variable especifica cuál de esas variantes se utilizó en la partida. Por ejemplo, dentro de la "Apertura Española", hay diferentes variantes como la "Defensa Morphy" o la "Defensa Steinitz".

En otras palabras, mientras que "opening_fullname" proporciona el nombre completo de la apertura, "opening_shortname" muestra una versión abreviada del nombre, y "opening_variation" se centra en la variante específica de la apertura jugada en la partida de ajedrez. Las variables mencionadas permiten identificar y categorizar las aperturas utilizadas en las partidas de ajedrez en el conjunto de datos.

Dado lo anterior, se anticipa que será necesario valorar si en realidad se pueden eliminar las dos columnas finales por la simple razón de que faltan datos o será necesario realizar algún tratamiento a éstas e incluirlas si es que se encuentra que tienen relevancia para determinar el triunfo en el juego.

Gráficos de la base de datos.

La siguiente gráfica muestra el histograma de cada una de las variables, en estos histogramas se omitió la variable que corresponde al índice de cada juego (game_id), también la que indica que la partida está clasificada o no (rated) que es una variable booleana, así como la de salida (winner). Se puede apreciar que algunas de estas variables pueden adquirir una gran cantidad de valores de tal forma que no es posible visualizar las etiquetas en el eje horizontal de la gráfica, aunque se hagan ajustes en el área de graficación. Algunas por simple observación, se puede afirmar que muestran sesgo en su distribución como time increment aunque aún no se realizan análisis. En la gráfica de opening_response se aprecia que los valores en el eje vertical son mucho menores que en el resto de las gráficas, indica datos faltantes como se verá más adelante.

Distribución de las variables

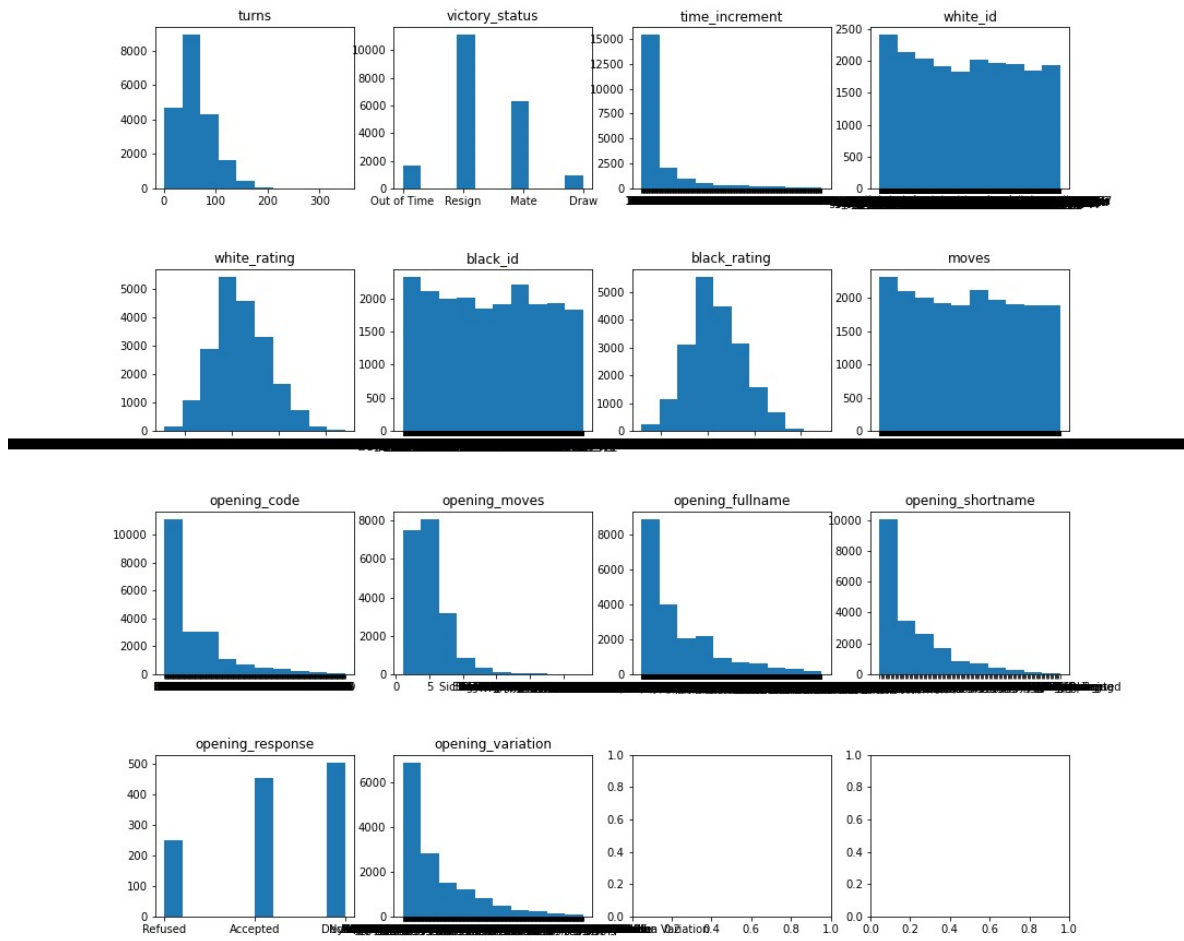


Ilustración 9. Distribución de las variables.

En la siguiente gráfica, se muestra el porcentaje de partidas ganadas por las fichas blancas, las de color negro y los empates, se puede apreciar que este último evento se presenta con mucho menor frecuencia que los otros dos. Al jugar una partida de ajedrez online menos del 5% serán empate.

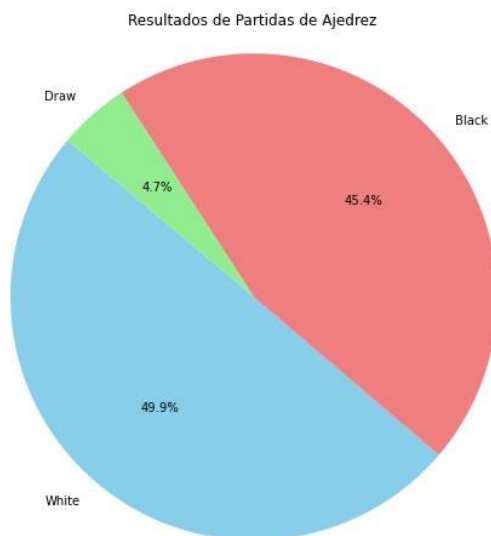


Ilustración 10. Porcentaje de victoria en el conjunto de datos.

En la gráfica que se muestra a continuación se permite apreciar que la mayoría de las partidas de ajedrez tienen entre 25 y 75 movimientos, también se puede apreciar que en el caso de los empates se mantiene más estable la variación en el número de movimientos realizados.

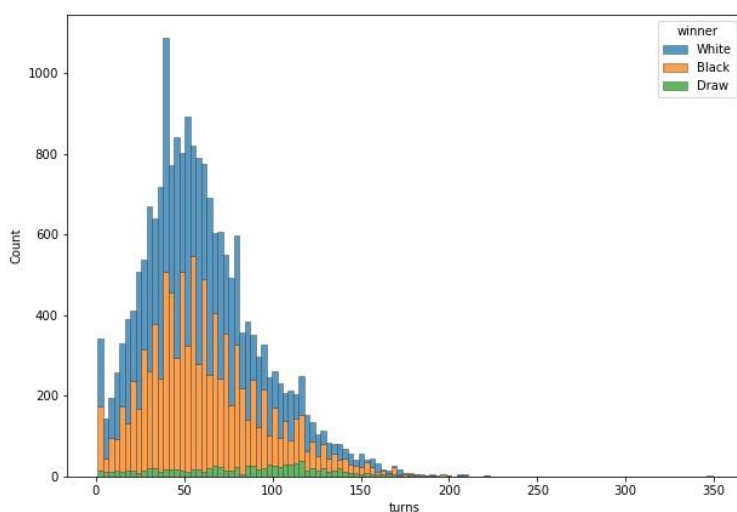


Ilustración 11. Comparación de turnos por victoria entre cada color.

En las tres siguientes gráficas se muestran los movimientos de apertura para los casos de empate, que ganen las fichas negras y que ganen las fichas blancas. Se puede apreciar que hay tres aperturas que son muy populares, puesto que para cualquiera de las opciones de desenlace del juego están presentes, estas son Sicilian Defense, French Defense y Queen's Pawn Game.

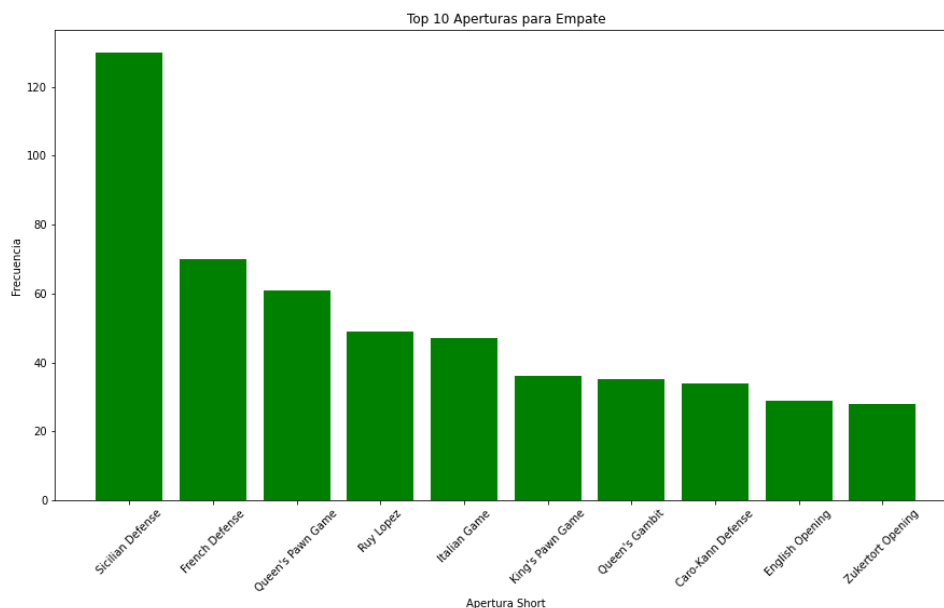


Ilustración 12. Top 10 aperturas. Empate.

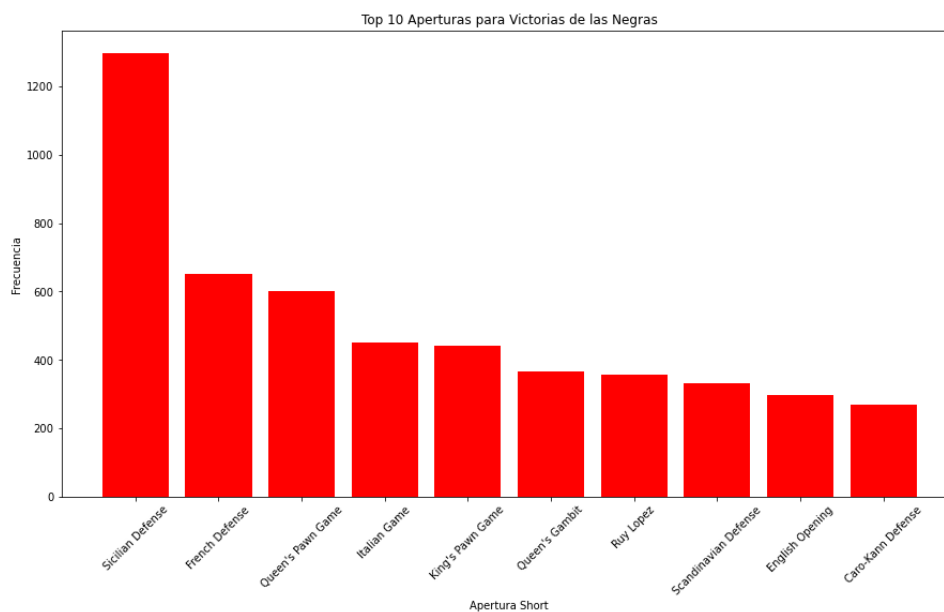


Ilustración 13. Top 10 aperturas. Victoria Negras.

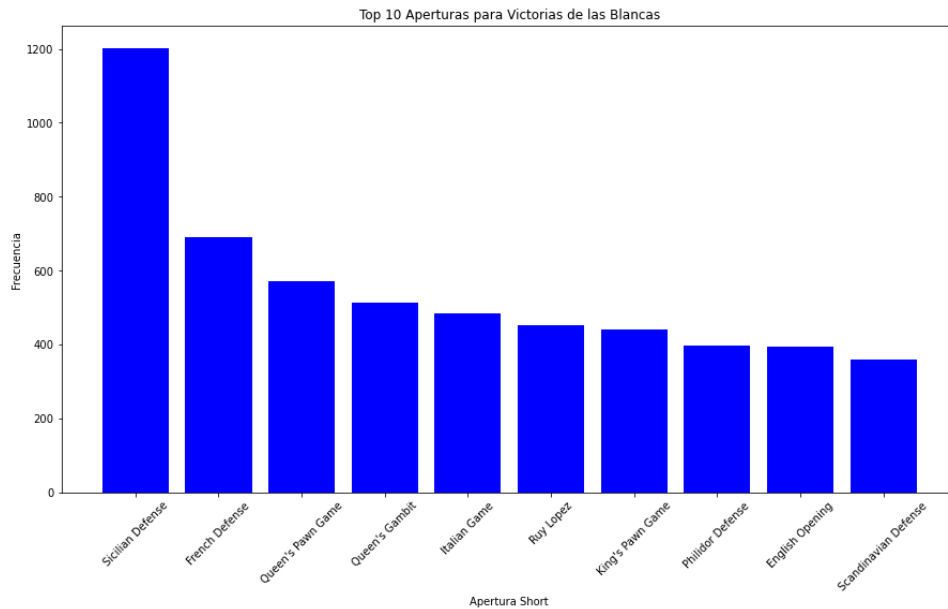


Ilustración 14. . Top 10 aperturas. Victoria blancas..

En el gráfico que relaciona las victorias con los estados de la victoria, se puede apreciar que las son menos las victorias que se obtienen porque uno de los participantes se excedió en el tiempo y eso le dio el triunfo al oponente, lo cual está marcado en color verde en la gráfica. En un tono de color naranja se tiene las victorias contundentes que son las de jaque mate. La mayoría de los ganadores, obtienen este resultado porque el contrincante así lo reconoce como se muestra en la opción *Resign* que se muestra en color amarillo. También podemos apreciar que muchas menos partidas, con respecto a las demás, en donde se finaliza con empate. La variable estado de victoria se elimina del entrenamiento del modelo, porque es un dato que no estaría disponible al momento de evaluar una posición de una partida en curso y porque puede tener correlación con el color ganador.



Ilustración 15. Relación entre ganador y estatus de la victoria.

Análisis de valores atípicos

Se analizaron las variables del dataset y se observan outliers en las variables numéricas “turns”, “white_raiting”, “black_raiting”, “opening_moves”

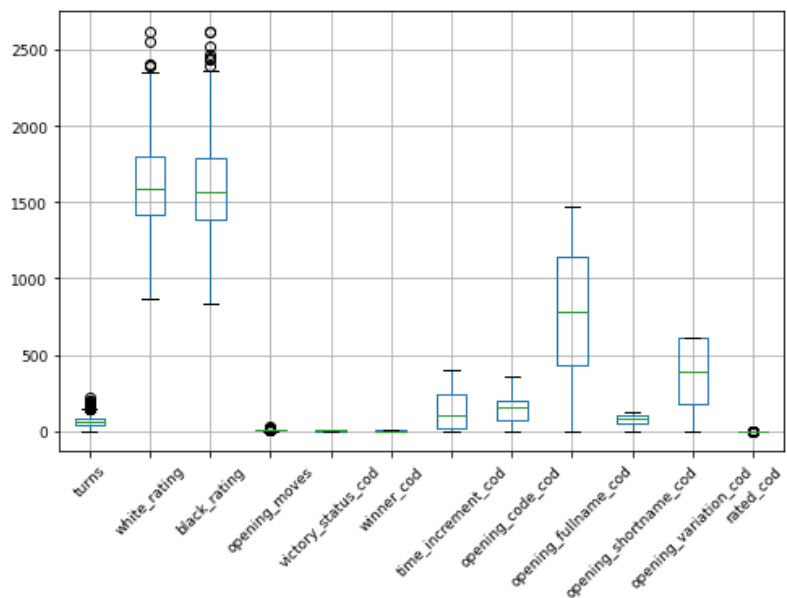


Ilustración 16. Valores atípicos.

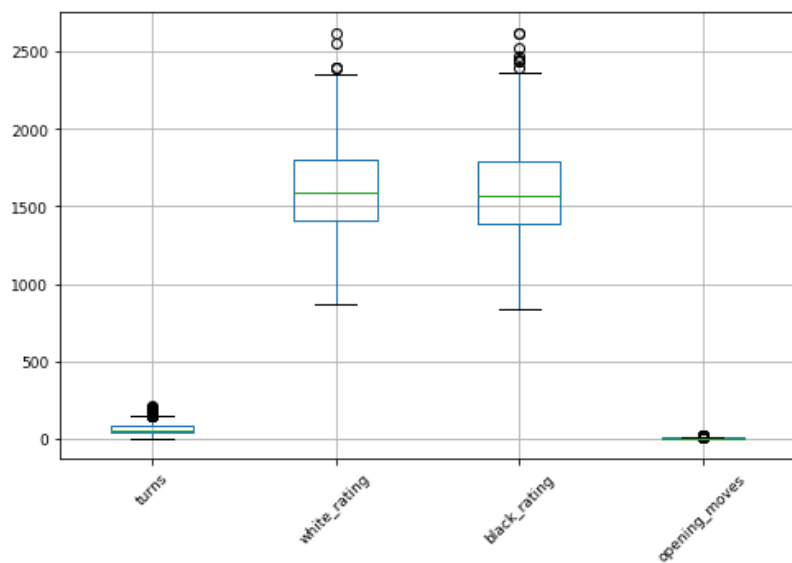


Ilustración 17. Valores atípicos.

Los datos atípicos no se consideran significativos.

Análisis de la asimetría de datos

Se realizó análisis de asimetría. Los resultados para cada variable concluyen que todas las variables están dentro del rango de -1.5 a 1.5 por lo que ninguna de ellas requiere transformación para mitigar asimetría.

Index	0
turns	0.51437
white_rating	0.184358
black_rating	0.178775
opening_moves	0.571839
victory_status_cod	-0.523309
winner_cod	-0.121072
time_increment_cod	0.795591
opening_code_cod	0.0496014
opening_fullname_cod	-0.105364
opening_shortcode	-0.389626
opening_variation_cod	-0.376158
rated_cod	-1.49703

Ilustración 18. Asimetría de los datos.

Criterio de Correlación

Se realizó el análisis de correlación para identificar aquellas variables altamente correlacionadas y así evitar problemas de estabilidad durante la corrida de algoritmos. Primeramente, se generó un Heat-map de Correlación el cual se puede apreciar a continuación:

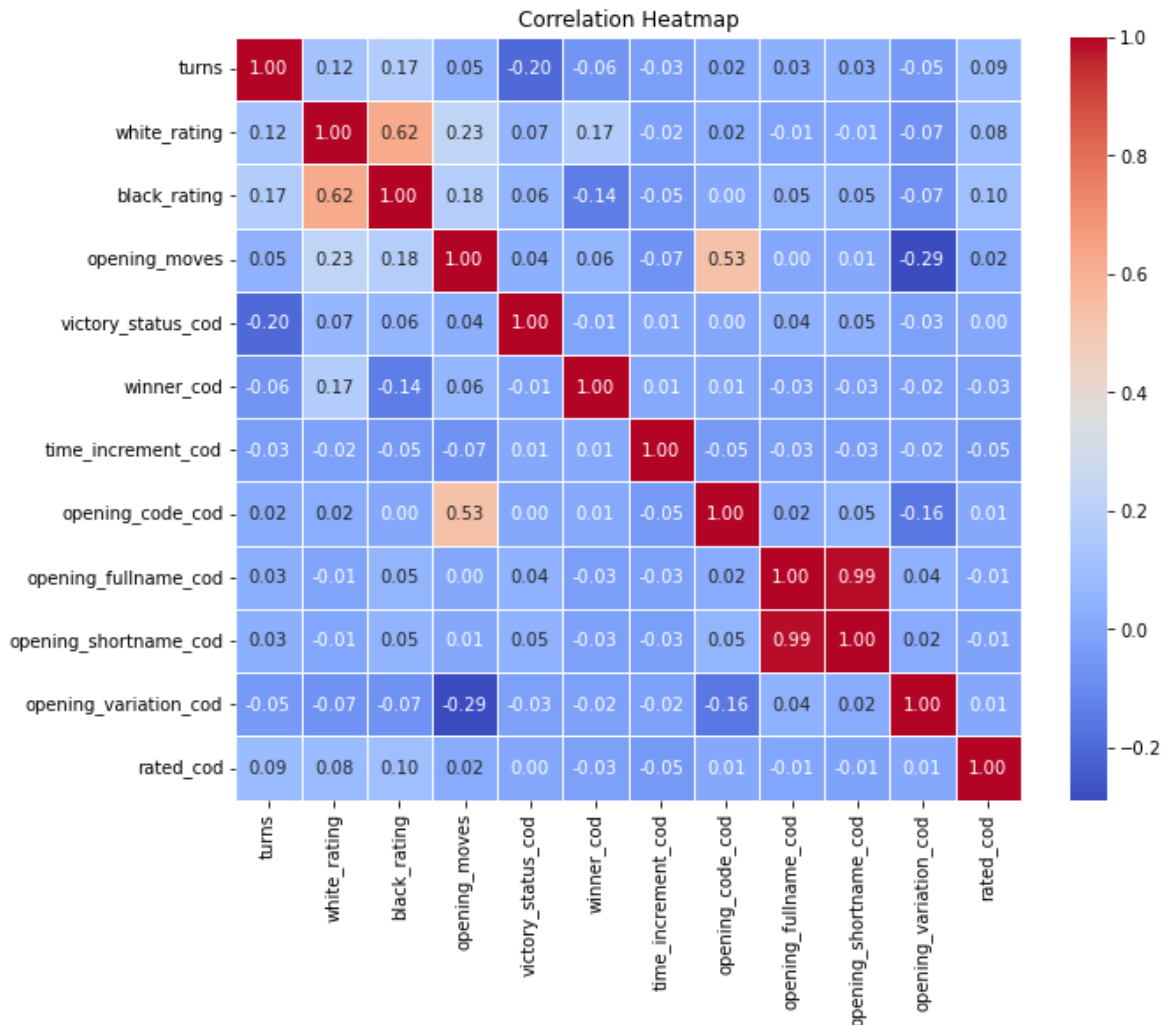


Ilustración 19. Mapa de calor de correlación.

Se puede deducir del Heat Map de Correlación lo siguiente:

- las variables “opening_fullname_cod” y “opening_shortcode” muestran alta correlación. Se decide dejar la variable “opening_shortcode” para procesamiento.
- las variables “white_rating” y “black_rating” muestran alta correlación. Se decide únicamente tomar “white_rating” para procesamiento.
- las variables “opening_code_cod” y “opening_moves” también muestran fuerte correlación, por lo que se deja únicamente la variable “opening_moves” para procesamiento.

Lo anterior se puede reforzar al analizar el algoritmo de clustering en donde se puede apreciar los clústeres de las variables relacionadas:

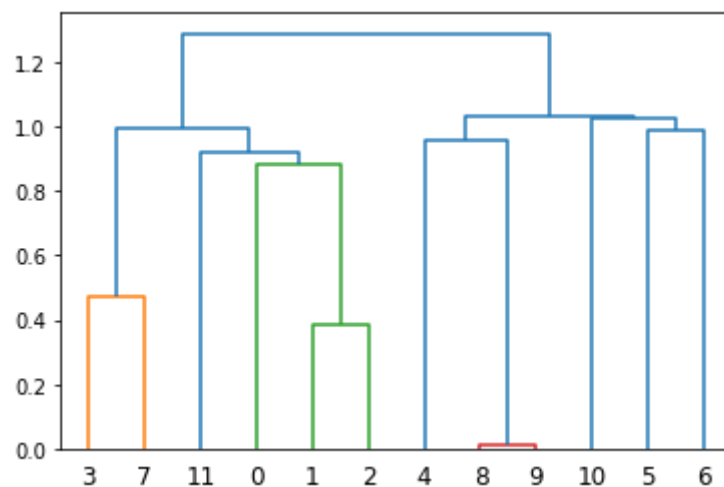


Ilustración 20. Clusters de las variables relacionadas.

La visualización arriba refuerza las conclusiones obtenidas anteriormente sobre la alta correlación en las variables 8 y 9 (“opening_fullname_cod” y “opening_shortcode”), seguido de 1 y 2 (“white_rating” y “black_rating”) y finalmente 3 y 7 (“opening_code_cod” y “opening_moves”).

Desarrollo de prototipo

Extracción de datos

Para el procesamiento de datos, se decidió tomar las muestras totales con escalamiento de datos en algunas corridas, mientras que en otras se decidió no realizar escalamiento, pero si hacer un remuestreo pasando de 20,058 muestras a 2,000 muestras seleccionadas aleatoriamente.

Para entender más acerca cada uno de los archivos utilizados (original y transformados) favor de revisar el Apéndice A en este documento.

Tratamiento de valores faltantes y limpieza

- La variable “opening_variation” contaba con 28% de datos faltantes. En base a análisis de la misma se concluyó que aquellas muestras en blanco eran porque no tenían “opening variation” (variación inicial). Esto se pudo observar con ayuda de la variable “opening_fullname”. Se decidió completar aquellas muestras en blanco con “traditional opening” (inicio tradicional)
- Las variables “white_id” y “black_id” fueron removidas del dataset. Estas columnas son nombres de usuarios que no influyen en nuestra variable de salida “winner”
- La variable “moves” fue removida del dataset. Son movimientos concatenados que no pueden procesarse o no generan valor para nuestra variable de respuesta, por la representación de la información.
- La variable “opening_response” está con el 93% de datos faltantes. Se decidió no incluirla en el análisis.

Codificación de variables categóricas

Se decidió usar LabelEncoder() de la librería sklearn para transformar las variables categóricas y codificarlas para poderlas procesar. Las variables transformadas fueron:

- victory_status
- winner
- time_increment
- opening_code
- opening_fullname
- opening_shortcode
- opening_variation
- rated

Modelos benchmark sin características nuevas

Máquina de soporte vectorial

Performance del modelo de Prueba

Accu 0.6679960119641076

Prec 0.6683791866345842

Reca 0.6679960119641076

Performance del modelo de Entrenamiento

Accu 0.6686609686609687

Prec 0.6703451456303168

Reca 0.6686609686609687

Ilustración 21. Performance SVM

Xgboost

Performance del modelo de Prueba

Accu 0.8853439680957128

Prec 0.8852826599908453

Reca 0.8853439680957128

Performance del modelo de Entrenamiento

Accu 0.948076923076923

Prec 0.9480746212677614

Reca 0.948076923076923

Ilustración 22. Performance benchmark. XGboost.

Regresión logística

Performance del modelo de Prueba

Accu 0.6693253572615487

Prec 0.6693539047039456

Reca 0.6693253572615487

Performance del modelo de Entrenamiento

Accu 0.6666666666666666

Prec 0.6674975765878098

Reca 0.6666666666666666

Ilustración 23. Performance benchmark. Regresión logística..

Ingeniería de características

Casillas controladas por piezas

Son el total de casillas atacadas por una pieza en las que hay una casilla vacía o una pieza de otro jugador. La función se encuentra adjunta en el apéndice archivo `extract_features.py`

```
piece_type = KNIGHT
piece_name = PIECE_NAMES[piece_type]
column_name = f"ctrl_{piece_name}"
df_1[column_name] = df_1['moves_fen'].apply(lambda x: ef.count_legal_moves_from_piece_type(x, piece_type))
```

FEN: rnbqk1nr/pp4pp/2p1p3/b7/3P1B2/2N2N2/PP2PPPP/R2QKB1R b KQkq - 5 7.

El knight controla 5 casillas.

Casillas controladas por knight (5): ['Ne7', 'Nh6', 'Nf6', 'Nd7', 'Na6']



Ilustración 24. Casillas controladas por caballo.

Diagonales controladas

Son el total de diagonales (2 o más casillas vacías o con piezas del oponente) controladas por reina y alfiles. La función se encuentra adjunta en el apéndice archivo `extract_features.py`

FEN: `rnbqk1nr/pp4pp/2p1p3/b7/3P1B2/2N2N2/PP2PPPP/R2QKB1R b KQkq - 5 7.`

Tiene 4 diagonales controladas

Diagonales controladas (4): {'a5': 2, 'c8': 0, 'd8': 2}

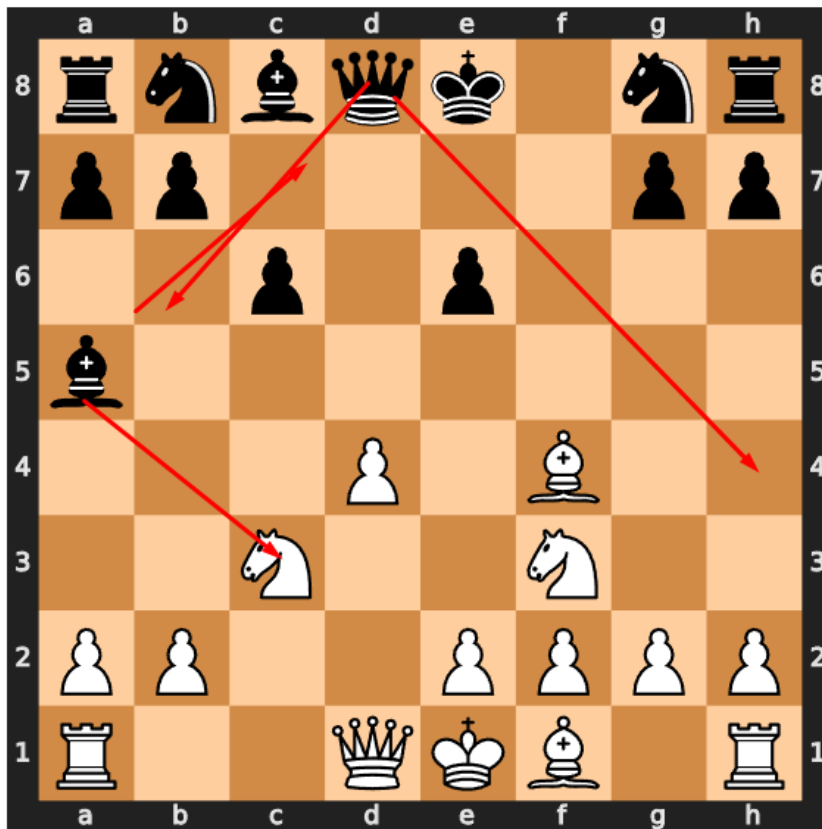


Ilustración 25. Diagonales controladas por reina y alfiles.

Columnas y filas controladas

Son el total de filas o columnas (2 o más casillas vacías o con piezas del oponente) controladas por torres y reina. La función se encuentra adjunta en el apéndice archivo `extract_features.py`

FEN: `rnbqk1nr/pp4pp/2p1p3/b7/3P1B2/2N2N2/PP2PPPP/R2QKB1R b KQkq - 5 7.`

Tiene 1 columnas y filas controladas.

Columnas y filas controladas (1): `{'a8': 0, 'h8': 0, 'd8': 1}`

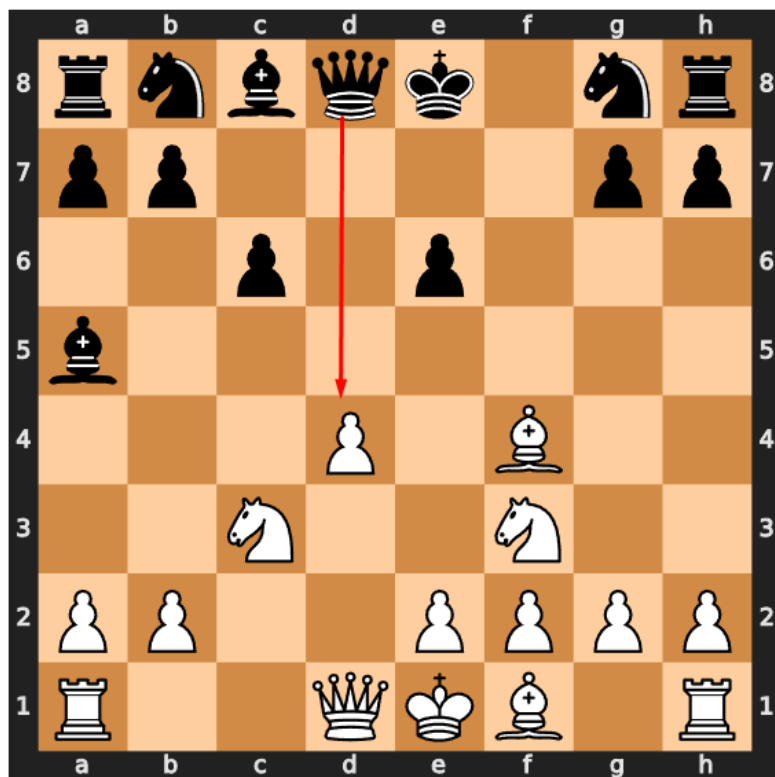


Ilustración 26. Columnas y filas controladas.

Puntos de presión

Son el total de casillas vacías o con piezas del oponente a las que 2 o más piezas del jugador en turno está atacando de forma simultánea, vital para medir de forma cuantitativa la ventaja sobre un oponente. La función se encuentra adjunta en el apéndice archivo `extract_features.py`

FEN: `rnbqk1nr/pp4pp/2p1p3/b7/3P1B2/2N2N2/PP2PPPP/R2QKB1R b KQkq - 5 7.`

Tiene 8 puntos de presión

Puntos de presión (8): `['d5', 'a6', 'b6', 'f6', 'h6', 'c7', 'd7', 'e7']`

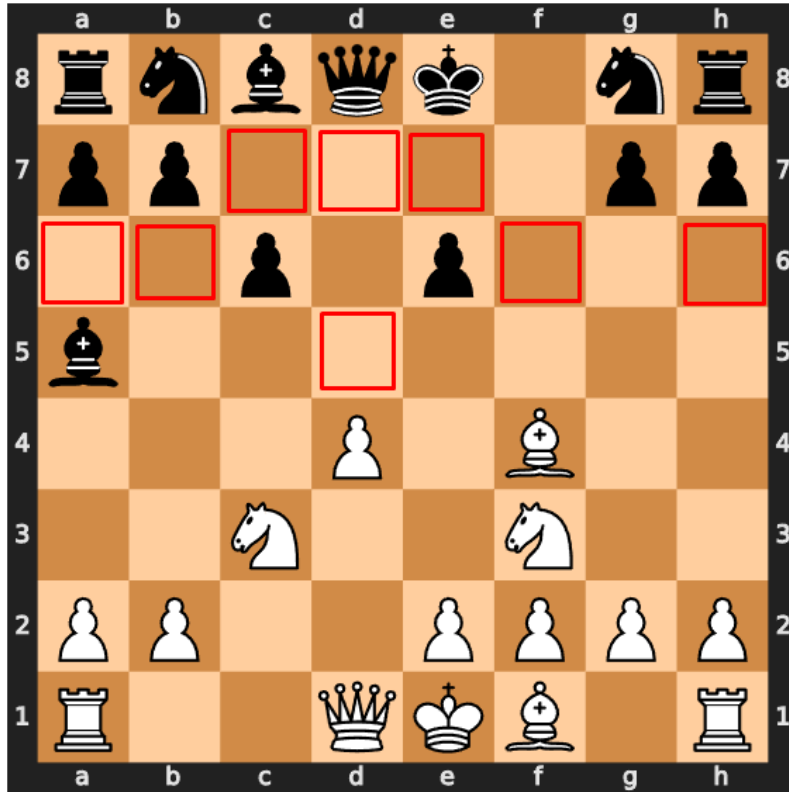


Ilustración 27. Puntos de presión.

Más características planean añadirse para IDI IV. Incluidas, pero no limitadas a:

- Número de piezas que están defendidas
- Número de peones que están correctamente conectados
- Número de peones que defienden al rey
- Número de casillas de escape del rey
- Número de piezas atacando o que podrían atacar en 1 turno al rey del oponente
- Número de piezas totales defendiendo al rey.

Modelos alimentados con datos históricos y nuevas características

Máquina de soporte vectorial

Performance del modelo de Prueba

Accu 0.8688933200398804

Prec 0.8510976792205357

Reca 0.8688933200398804

Performance del modelo de Entrenamiento

Accu 0.888960113960114

Prec 0.8939622361874665

Reca 0.888960113960114

Ilustración 28. Performance. SVM.

Xgboost

Performance del modelo de Prueba

Accu 0.8772017281488866

Prec 0.8698236947016245

Reca 0.8772017281488866

Performance del modelo de Entrenamiento

Accu 0.9878205128205129

Prec 0.9878922903623923

Reca 0.9878205128205129

Ilustración 29. Performance. XGboost.

Regresión logística

Performance del modelo de Prueba

Accu 0.8517779993353274

Prec 0.8474827911056829

Reca 0.8517779993353274

Performance del modelo de Entrenamiento

Accu 0.8565527065527065

Prec 0.84222834883223

Reca 0.8565527065527065

Ilustración 30. Performance. Regresión logística.

Siguientes pasos idi 4

Implementar redes neuronales con optimización de hiperparámetros

Entrenar el modelo nuevamente cortando las partidas a únicamente la apertura, para que sea más enfocado en los objetivos de este trabajo.

Entrenar un modelo de aprendizaje profundo que sea capaz de evaluar alternativas a futuro en una posición y brindarte bias de victoria ejecutando en ensamble el modelo ideal ya entrenado en pasos anteriores.

Complementar la información teórica y detallar más el tratamiento de datos en este reporte.

Resultados y análisis

Discusión

Bibliografía

365 chess. (2023). *365 chess*. Obtenido de <https://www.365chess.com/opening.php>

Chess position trainer. (2019). *Chess position trainer*. Obtenido de <http://www.chesspositiontrainer.com/index.php/en/buy>

Chess Tempo . (2023). *Chesstempo*. Obtenido de <https://old.chesstempo.com/game-database.html>

Chess.com. (2017). *Chess.com*. Obtenido de <https://www.chess.com/home>

ChessBase Reader. (2017). *ChessBase Reader 2017*. Obtenido de <https://en.chessbase.com/pages/download>

Di Luca, G. (2012). *How Important Are Chess Openings? (7 Reasons To Study)*. Obtenido de <https://chesspulse.com/how-important-are-openings-in-chess>

Hooper, D., & Whyld, K. (1984). *The oxford companion to chess*. Brithis Library Cataloguing.

Apéndices

Tabla 2. *extract_features.py*

```
"""
Este archivo de python tiene la intención de proveer una serie de metodos
para extraer características
desde posiciones dadas por cadenas FEN.
"""

from chess import Board
from chess import square_rank, square_file, square
from chess import PIECE_NAMES, PIECE_TYPES, SQUARES, square_name
from chess import BISHOP, ROOK, QUEEN

def get_legal_moves_from_piece_type(fen, piece_type):
    """
    Devuelve los movimientos legales de una pieza dada en el turno
    correspondiente a la posición fen

    Args:
    fen (str): The FEN string representing the current board position.
    piece_type (int):
        PAWN = 1
        KNIGHT = 2
        BISHOP = 3
        ROOK = 4
        QUEEN = 5
        KING = 6

    Returns:
    list: A list of legal moves for given piece in uci notation.
    """
    board = Board(fen)
    legal_moves = board.legal_moves
    moves = [board.san(move) for move in legal_moves if
board.piece_at(move.from_square).piece_type == piece_type] # move.uci for
uci format
    return moves

def count_legal_moves_from_piece_type(fen, piece_type):
    """
    Cuenta el número de movimientos legales de una pieza dada en el turno
    correspondiente a la posición fen

    Args:
    fen (str): The FEN string representing the current board position.
```

```

piece_type (int):
    PAWN = 1
    KNIGHT = 2
    BISHOP = 3
    ROOK = 4
    QUEEN = 5
    KING = 6

Returns:
list: A list of legal moves for given piece in uci notation.
"""

board = Board(fen)
legal_moves = board.legal_moves
moves = [1 for move in legal_moves if
board.piece_at(move.from_square).piece_type == piece_type] # move.uci for
uci format
return sum(moves)

def get_pressure_points_san(fen):
    """
    Devuelve una lista de las casillas que son puntos de presión en
    notación SAN.
    Un punto de presión se define como una casilla que es atacada por más
    de una pieza
    del jugador cuyo turno es actualmente.

    Args:
    fen (str): La cadena FEN que representa la posición actual del
    tablero.

    Returns:
    list: Lista de las casillas de puntos de presión en notación SAN.
    """

    board = Board(fen)
    current_turn = board.turn # TRUE (WHITE) FALSE (BLACK)
    attack_map = {square: 0 for square in SQUARES}

    # Iterar solo sobre las piezas del color que tiene el turno
    for piece_type in PIECE_TYPES:
        for square in board.pieces(piece_type, current_turn):
            attacked_squares = board.attacks(square)
            for attacked_square in attacked_squares:
                attacked_piece = board.piece_at(attacked_square)
                # Incrementar si la casilla atacada está vacía o tiene una
                # pieza del color opuesto

```

```

        if not attacked_piece or attacked_piece.color !=
current_turn:
            attack_map[attacked_square] += 1

    # Los puntos de presión son cuadrados a los que más de una pieza ataca
    pressure_points = [square_name(sq) for sq, count in attack_map.items()
if count > 1]

    return pressure_points

def count_pressure_points(fen):
    """
    Cuenta las casillas que son puntos de presión.
    Un punto de presión se define como una casilla que es atacada por más
de una pieza
    del jugador cuyo turno es actualmente.

    Args:
    fen (str): La cadena FEN que representa la posición actual del
tablero.

    Returns:
    int: Conteo de las casillas de puntos de presión.
    """
    board = Board(fen)
    current_turn = board.turn # TRUE (WHITE) FALSE (BLACK)
    attack_map = {square: 0 for square in SQUARES}

    # Iterar solo sobre las piezas del color que tiene el turno
    for piece_type in PIECE_TYPES:
        for square in board.pieces(piece_type, current_turn):
            attacked_squares = board.attacks(square)
            for attacked_square in attacked_squares:
                attacked_piece = board.piece_at(attacked_square)
                # Incrementar si la casilla atacada está vacía o tiene una
pieza del color opuesto
                if not attacked_piece or attacked_piece.color !=
current_turn:
                    attack_map[attacked_square] += 1

    # Los puntos de presión son cuadrados a los que más de una pieza ataca
    pressure_points = [1 for _, count in attack_map.items() if count > 1]

    return sum(pressure_points)

```

```

def get_legal_moves_san(fen):
    """
    Devuelve una lista con todos los movimientos legales de cada pieza en
    el tablero
    para el jugador en turno

    Args:
        fen (str): La cadena FEN que representa la posición actual del
        tablero.

    Returns:
        list: Lista de las casillas de puntos de presión en notación SAN.
    """
    legal_moves = {}
    # Fill legal moves according one piece type
    for piece_type in PIECE_TYPES:
        legal_moves[piece_type] = get_legal_moves_from_piece_type(fen,
        piece_type)
    return legal_moves

def get_all_controlled_diagonals(fen):
    """
    Obtiene el número de diagonales controladas por alfiles o reinas en un
    tablero de ajedrez,
    dado por la notación FEN.
    Una diagonal está controlada si al menos dos casillas están libres o
    contienen una
    pieza del color opuesto.
    """
    # Crear un tablero a partir de la notación FEN
    tablero = Board(fen)
    current_turn = tablero.turn # TRUE (WHITE) FALSE (BLACK)
    diagonales_controladas = {}
    diagonales_controladas_sum = 0

    # Por cada casilla entre las 64 posibles
    for piece_type in BISHOP, QUEEN:
        for square in tablero.pieces(piece_type, current_turn):
            # Sumar diagonales controladas de alfiles y reinas
            result = get_controlled_diagonals_by_square(square, tablero)
            diagonales_controladas[square_name(square)] = result
            diagonales_controladas_sum += result

    return (diagonales_controladas, diagonales_controladas_sum)

```

```

def count_all_controlled_diagonals(fen):
    """
    Cuenta el número de diagonales controladas por alfiles o reinas en un
    tablero de ajedrez,
    dado por la notación FEN.
    Una diagonal está controlada si al menos dos casillas están libres o
    contienen una
    pieza del color opuesto.
    """

    # Crear un tablero a partir de la notación FEN
    tablero = Board(fen)
    current_turn = tablero.turn # TRUE (WHITE) FALSE (BLACK)
    diagonales_controladas_sum = 0

    # Por cada casilla entre las 64 posibles
    for piece_type in BISHOP, QUEEN:
        for square in tablero.pieces(piece_type, current_turn):
            # Sumar diagonales controladas de alfiles y reinas
            result = get_controlled_diagonals_by_square(square, tablero)
            diagonales_controladas_sum += result

    return diagonales_controladas_sum

def get_controlled_diagonals_by_square(casilla, board: Board):
    # Direcciones de las diagonales: superior izquierda, superior derecha,
    inferior izquierda, inferior derecha
    direcciones = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
    controlled_diagonals = 0
    # Por cada diagonal en las direcciones
    for dx, dy in direcciones:
        # Verificar las dos primeras casillas en cada dirección diagonal
        for i in range(1, 3):
            nueva_fila, nueva_columna = square_rank(casilla) + i * dx,
            square_file(casilla) + i * dy
            # Si alguna de las dos casillas está fuera del tablero, la
            diagonal no esta controlada o es inutil
            if not (0 <= nueva_fila <= 7 and 0 <= nueva_columna <= 7):
                break
            nueva_casilla = square(nueva_columna, nueva_fila)
            pieza_original = board.piece_at(casilla)
            pieza_en_diagonal = board.piece_at(nueva_casilla)

            # Si alguna de las dos casillas tiene una pieza del mismo
            color, la diagonal no está controlada o no es util

```

```

        if pieza_en_diagonal and pieza_en_diagonal.color ==
pieza_original.color:
            break
        else:
            controlled_diagonals += 1
    return controlled_diagonals

def get_all_controlled_lines(fen):
    """
    Cuenta el número de líneas horizontales y verticales controladas por
    torres y reina en un tablero de ajedrez,
    dada una determinada posición FEN
    """
    # Crear un tablero a partir de la notación FEN
    tablero = Board(fen)
    current_turn = tablero.turn # TRUE (WHITE) FALSE (BLACK)
    lineas_controladas = {}
    lineas_controladas_sum = 0

    # Por cada torre del jugador actual
    for piece_type in ROOK, QUEEN:
        for square in tablero.pieces(piece_type, current_turn):
            # Sumar líneas horizontales y verticales controladas por la
pieza
            result = get_controlled_lines_by_square(square, tablero)
            lineas_controladas[square_name(square)] = result
            lineas_controladas_sum += result

    return (lineas_controladas, lineas_controladas_sum)

def count_all_controlled_lines(fen):
    """
    Cuenta el número de líneas horizontales y verticales controladas por
    torres y reina en un tablero de ajedrez,
    dada una determinada posición FEN
    """
    # Crear un tablero a partir de la notación FEN
    tablero = Board(fen)
    current_turn = tablero.turn # TRUE (WHITE) FALSE (BLACK)
    lineas_controladas_sum = 0

    # Por cada torre del jugador actual
    for piece_type in ROOK, QUEEN:
        for square in tablero.pieces(piece_type, current_turn):

```

```

        # Sumar líneas horizontales y verticales controladas por la
pieza
        result = get_controlled_lines_by_square(square, tablero)
        lineas_controladas_sum += result

    return lineas_controladas_sum

def get_controlled_lines_by_square(casilla, board: Board):
    """
    Cuenta el numero de líneas controladas por determinada casilla
    Una línea está controlada si al menos dos casillas contiguas a la
pieza están libres o contienen una
pieza del color opuesto.
    """
    # Direcciones: izquierda, derecha, arriba, abajo
    direcciones = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    controlled_lines = 0
    # Por cada dirección
    for dx, dy in direcciones:
        # Verificar las dos primeras casillas en cada dirección
        for i in range(1, 3):
            nueva_fila, nueva_columna = square_rank(casilla) + i * dy,
square_file(casilla) + i * dx
            # Si alguna de las dos casillas está fuera del tablero, la
línea no está controlada o es inútil
            if not (0 <= nueva_fila <= 7 and 0 <= nueva_columna <= 7):
                break

            nueva_casilla = square(nueva_columna, nueva_fila)
            pieza_original = board.piece_at(casilla)
            pieza_en_linea = board.piece_at(nueva_casilla)

            # Si alguna de las dos casillas tiene una pieza del mismo
color, la línea no está controlada o no es útil
            if pieza_en_linea and pieza_en_linea.color ==
pieza_original.color:
                break
            else:
                controlled_lines += 1
        return controlled_lines

def get_fen_from_moves(moves):
    game = Board()
    for move in moves.split():
        game.push_san(move)

```



```

    # Get FEN string of position
    fen = game.fen()
    return fen

def get_current_turn(fen):
    # Crear un tablero a partir de la notación FEN
    tablero = Board(fen)
    current_turn = tablero.turn # TRUE (WHITE) FALSE (BLACK)
    return current_turn

# Example FEN string
# fen_example = "rnbqkbnr/pppppppp/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
# fen_example = 'rnbqkb1r/p6p/6pn/1p1pB3/1p6/N2Q2P1/P1P1PP1P/1R2KBNR w Kkq - 0 12'
fen_example = 'rnbqk1nr/pp4pp/2p1p3/b7/3P1B2/2N2N2/PP2PPPP/R2QKB1R b KQkq - 5 7'

print(f"Obteniendo datos de la posición FEN: {fen_example}")

for piece_type in PIECE_TYPES:
    legal_moves = get_legal_moves_from_piece_type(fen_example, piece_type)
    print(f"{PIECE_NAMES[piece_type]}'s legal moves({len(legal_moves)}): {legal_moves}")

# for piece_type in PIECE_TYPES:
#     count_legal_moves = count_legal_moves_from_piece_type(fen_example, piece_type)
#     print(f"{PIECE_NAMES[piece_type]}'s legal moves({count_legal_moves}).")

pressure_points = get_pressure_points_san(fen_example)
print(f"Puntos de presión ({len(pressure_points)}): {pressure_points}")

# count_pp = count_pressure_points(fen_example)
# print(f"Puntos de presión ({count_pp})")

diagonals, diagonals_sum = get_all_controlled_diagonals(fen_example)
print(f"Diagonales controladas ({diagonals_sum}): {diagonals}")

lines, lines_sum = get_all_controlled_lines(fen_example)
print(f"Lineas controladas ({lines_sum}): {lines}")

moves = "e4 e5 Nf3 d6 d4 Nc6 d5 Nb4 a3 Na6 Nc3 Be7 b4 Nf6 Bg5 O-O b5 Nc5 Bxf6 Bxf6 Bd3 Qd7 O-O Nxd3 Qxd3 c6 a4 cxd5 Nxd5 Qe6 Nc7 Qg4 Nxa8 Bd7 Nc7 Rc8 Nd5 Qg6 Nxf6+ Qxf6 Rfd1 Re8 Qxd6 Bg4 Qxf6 gxf6 Rd3 Bxf3 Rxf3 Rd8 Rxf6"

```

```
Kg7 Rf3 Rd2 Rg3+ Kf8 c3 Re2 f3 Rc2 Rg5 f6 Rh5 Kg7 Rd1 Kg6 Rh3 Rxc3 Rd7  
Rc1+ Kf2 Rc2+ Kg3 h5 Rxb7 Kg5 Rxa7 h4+ Rxh4 Rxg2+ Kxg2 Kxh4 b6 Kg5 b7 f5  
exf5 Kxf5 b8=Q e4 Rf7+ Kg5 Qg8+ Kh6 Rh7#"
```

```
print(f"Fen from moves: {get_fen_from_moves(moves)}")
```

```
print(f"Current turn: {'White' if get_current_turn(fen_example) else  
'Black'}")
```