Relatório Trabalho 3 - ML

Mariana Botelho Pereira
Universidade Federal Fluminense
Av. Gal. Milton Tavares de Souza, s/nº
Niterói, RJ, 24210-346

Abstract

Este trabalho utilizou um dataset de Pokémon para experimentar diversos métodos de aprendizado de máquina, utilizando a biblioteca Scikit Learn. Também foram utilizadas as seguintes bibliotecas: Pandas, para manipular o dataset e Matplotlib e Seaborn para plotar os gráficos. Dentre os métodos utilizados, os *Decision Tree* e o *Random Forest* se mostraram os mais precisos e acurados, com taxas de acerto acima de 90%.

Desrição do dataset e motivação

O dataset escolhido foi o *Complete Pokemon Dataset (Updated 30.04.20)*, disponível em `https://www.kaggle.com/mariotormo/complete-pokemon-dataset-updated-090420?select=pokedex_%28Update_05.20%29.csv`. No total, o dataset apresenta 51 categorias com 890 atributos, sendo eles numéricos, categóricos e textuais.

Pokémon

Pokémon é uma série de jogos que se inciou no fim da década de 90 e vem tendo versões novas até a data de escrita deste relatório. No jogo, existem diversos pokémons (uma abreviação de *pocket monsters*, em português, monstros de bolso), que são animais que vivem no mundo do jogo. O objetivo do jogador é capturar um de cada animal, batalhando contra estes animais e outros personagens humanos.
Cada pokémon tem características próprias como: tipo primário (grama, fogo, inseto, entre outros), tipo secundário (alguns não possuem tipo secundário), tamanho (peso e altura), e valores de vida, ataque e defesa. De tempos em tempos uma nova geração de pokémons é apresentada com, na média, 150 novos animais. Atualmente existem 898 pokémons (por motivos desconhecidos, 8 pokémons não estão presentes no dataset), distribuídos entre oito gerações. Mais informações sobre os jogos e os animais podem ser encontradas em `https://bulbapedia.bulbagarden.net/wiki/Main_Page` (em inglês).

Motivação

Dado o histórico de Pokémon não só nos jogos como em outras mídias (anime, filmes e jogos de carta), a série se fez muito presente na vida de crianças e jovens que cresceram durante a década de 1990 e 2000, e novamente desde 2016 com o lançamento do jogo PokémonGo. Como uma dessas pessoas, sempre tive apreço pelos jogos, que estiveram presentes em minha vida durante anos. Sendo assim, achei interessante explorar as personagens por um lado mais sério e objetivo, explorando informações que não são usualmente analizadas. Também foi uma oportunidade de explorar um dataset relativamente fora do padrão de seriedade.

Descrição das categorias

- *pokedex_number*: O número do Pokémon no *National Pokedex*;
- *name*: Nome em inglês;
- *german_name*: Nome em alemão;
- *japanese_name*: Nome (original) em japonês;
- *generation*: NNúmero da geração em que o Pokémon foi introduzido;
- *status*: Diz se o Pokémon é normal, sub-lendário, lendário ou mítico;
- *species*: A espécie do Pokémon;
- *type_number*: Número de típos que o Pokémon possui;
- *type_1*: O tipo primário do Pokémon. Ao total, são 18 tipos;
- *type_2*: O tipo secundário do Pokémon, caso possua;
- *height_m*: A altura do Pokémon em metros;
- *weight_kg*: O peso do Pokémon em quilogramas;
- *abilities_number*: O número de habilidades que o Pokémon possui;
- *ability_?*: Os nomes das habilidades;
- *ability_hidden*: O nome da habilidade secreta, caso possua;
- *total_points*: Valor do *effort value*;
- *hp*: Vida base do Pokémon;
- *attack*: Valor de ataque básico;

- *defense*: Balor de defesa básico;
- *sp_attack*: Valor base do ataque especial;
- *sp_defense*: Valor base da defesa especial;
- *speed*: Valor de velocidade básico;
- *catch_rate*: Taxa de captura do Pokémon;
- *base_friendship*: Valor de amizade base;
- *base_experience*: Valor de experiência do Pokémon selvagem ao ser capturado;
- *growth_rate*: Taxa de crescimento;
- *egg_type_number*: Número de grupos dos quais o Pokémon pode ser chocado;
- *egg_type_?*: Nome dos tipos de ovos dos quais o Pokémon pode ser chocado;
- *percentage_male*: Porcentagem de Pokémons da espécie que são macho. Vazio caso não possua gênero;
- *egg_cycles*: Número de ciclos (255-257 passos) necessários para chocar o Pokémon;
- *against_?*: Dezoito categorias que demonstram quanto de dano o Pokémon recebe de cada tipo.

Métodos e estratégias

Dadas as informações presentes no dataset, diversos problemas poderiam ser explorados, como previsão do valor de ataque de um pokémon (problema de regressão), se um pokémon é normal ou mítico (problema de classificação binária) ou qual o tipo de um pokémon (problema de classificação multiclasse).
Ao iniciar o jogo, o jogador tem que escolher um dentre três pokémons: um Bulbasaur, pokémon de grama; um Squirtle, pokémon de água; e um Charmander, pokémon de fogo. Para este trabalho, esses foram considerados os tipos básicos, e a ideia é avaliar se os modelos de aprendizado de máquina conseguem determinar pelas outras características, se um pokémon faz parte desses tipos básicos ou de outro tipo, sendo assim um problema de classificação binária.
Para fazer a avaliação, foram escolhidos os métodos mais tradicionais e conhecidos de classificação.
Para obter um modelo o mais homogêneo possível, foi utilizado o método *train_test_split*, que pega valores randômicos por todo o dataset, utilizando 70% do dataset para treino e 30% para teste.

Pré-processamento

Para o dataset refletir informações mais acuradas e mais fáceis para os modelos avaliarem, um pre-processamento foi feito.
Algumas colunas com informações consideradas irrelevantes (*Unnamed: 0*, *name*, *german_name*, *japanese_name*, *species*, *type_number*, *egg_type_number*) foram removidas, assim como algumas com informações que facilitariam a avaliação (*ability_1*, *ability_2*, *ability_hidden*, *egg_type_1*, *egg_type_2*).
Também foram removidos linhas que posuissem valor nulo nas colunas *type_1* e/ou *growth_rate*, já que todo pokémom precisa possuir ao menos um tipo e sua taxa de crescimento. Para as demais categorias, os valores nulos foram subsituídos por 0.
Em seguida, foi utilizado o OrdinalEncoder para encodar as colunas *status* e *growth_rate*, que passaram de textuais para categóricas.
A coluna *type_2* precisou ser pré-processada manualmente, pois possuia algumas linhas com valores nulos.
A coluna *type_1* também foi pré-processada manualmente, para que pudessem ser separadas as categorias desejadas (*Grass*, *Fire* e *Water* das outras).

LinearSVC

Primeiramente foi escolhido o método *Linear Support Vector Classification*. Este método utiliza um sistema em que ele separa os dados em classes utilizando um hiperplano.
Este método foi escolhido por recomendação da própria biblioteca utilizada, Scikit learn, em seu *algorithm cheat sheet* [1].
Neste modelo, foram utilizados dois valores distintos para o hiperparâmetro *random_state*: o valor padrão de 42 e o valor 0.

Decision Tree

Decision Tree é um método de aprendizado de máquina que ulitiza uma árvore de decisão para ir de observações sobre um item até uma conclusão sobre o item.
Este método foi escolhido por possuir uma complexidade de entendimento relativamente baixa, ou seja, não é um método com cálculos complexos.
Assim como no *LinearSVC*, foram utilizados dois valores distintos para o hiperparâmetro *random_state*: o valor padrão de 42 e o valor 0.

Logistic Regression

Logistic Regression é um método de aprendizado que utiliza uma função logarítmica para modelar uma variável binária a partir de outras variáveis binárias e/ou contínuas.
Este método foi escolhido por ser um dos mais comuns ao se fazer um modelo de classificação binária. Segundo o blog *Towards Data Science*, este método é utilizado desde o começo do século XX [2].
Assim como no *LinearSVC* e no *Decision Tree*, foram utilizados dois valores distintos para o hiperparâmetro *random_state*: o valor padrão de 42 e o valor 0.

K-Nearest Neighbors

KNN é um método que classifica um dado de acordo com quão similar ele é de seus vizinhos.
Este modelo se mostrou interessante pois depende que seus vizinhos sejam do mesmo tipo para que ele acerte. Dado que

[1] Disponível em https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

[2] Disponível em: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

eu não sei quão independentes são as variáveis do dataset, achei interessante utilizar esse método.
Para este método, foram utilizados duas quantidades de vizinhos distintas, expressas pelo hiperparâmetro *n_neighbors*: 4 vizinhos e o padrão com 5 vizinhos.

Perceptron

Perceptron é um modelo de rede neural artificial. Este é considerado o modelo de rede neural mais simples, pois utiliza apenas um neurônio, com uma saída binária.
Este modelo foi escolhido pois, por ser um modelo simples, poderia ser uma inicialização em redes neurais.

Random Forest

Random Forest é um método de *ensemble* que utiliza uma "floresta" de árvores de decisão, e calcula a moda entre seus resultados.
Este método foi escolhido para poder fazer uma análise entre ele e as árvores de decisão simples.

Stacking Classifier

Como solicitado, foi utilizado o *Stacking Classifier* para juntar todos os métodos. Este método de *ensemble* "empilha" os resultados de todos os outros métodos e utiliza um estimador final para calcular a predição final. Neste caso, foi utilizado o estimador padrão, *LogisticRegression*.

Resultados

Para definir se um método é ótimo para um modelo, utilizamos diversas medições a partir dos resultados positivos reais, negativos reais, falso-positivos e falso-negativos, que serão descritas nas sessões a seguir. Alguns gráficos para comparação entre modelos também serão apresentados.

Accuracy, precision, recall e F1

Para facilitar a comparação entre os métodos, as métricas também serão apresentadas em quatro gráficos.

Model	Accuracy	Precision	Recall	F1
LinearSVC42	0.670175	0.796091	0.822222	0.723100
LinearSVC0	0.718720	0.799310	0.840000	0.799384
Dec. Tree42	0.928662	0.936690	0.968986	0.952134
Dec. Tree0	0.931994	0.944648	0.964541	0.953995
Log. Regr.42	0.754151	0.801854	0.884928	0.840690
Log. Regr.0	0.754151	0.801854	0.884928	0.840690
KNN 5	0.605235	0.706440	0.787536	0.744545
KNN 4	0.537229	0.705634	0.628019	0.664024
Perceptron	0.589053	0.847128	0.556812	0.645481
Rand. Forest	0.932152	0.944215	0.964541	0.953958
Stack. Class.	0.589053	0.847128	0.556812	0.645481

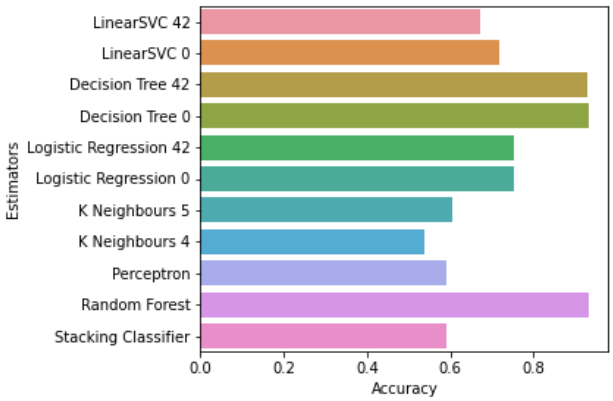

Figure 1: Accuracy

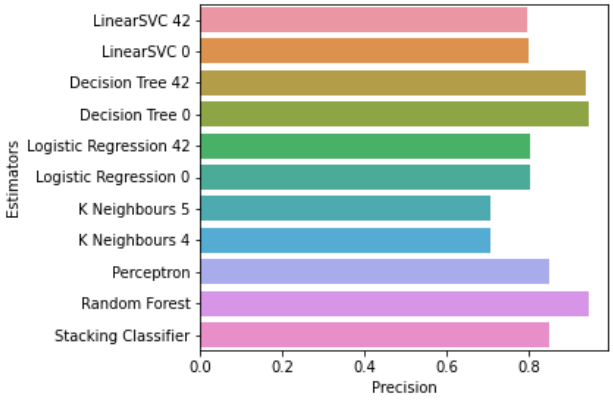

Figure 2: Precision

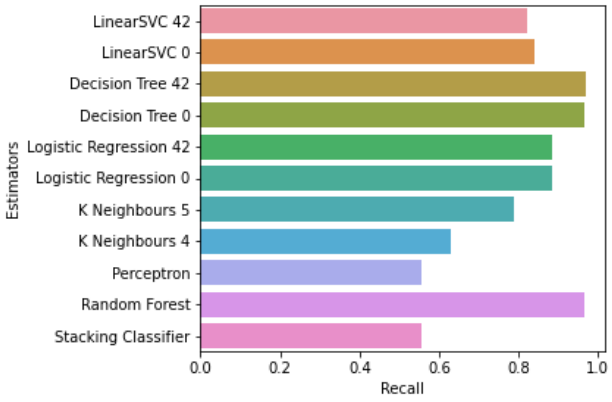

Figure 3: Recall

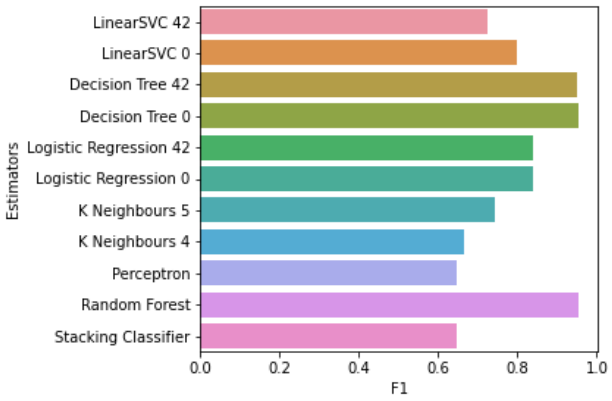

Figure 4: F1

Matrizes de confusão

Linear SVC 42		
	P_NOT_BASIC	P_BASIC
not_basic	21	62
basic	40	186

Linear SVC 0		
	P_NOT_BASIC	P_BASIC
not_basic	32	51
basic	36	190

Decision Tree 42		
	P_NOT_BASIC	P_BASIC
not_basic	68	15
basic	7	219

Decision Tree 0		
	P_NOT_BASIC	P_BASIC
not_basic	70	13
basic	8	218

Logistic Regression 42		
	P_NOT_BASIC	P_BASIC
not_basic	33	50
basic	26	200

Logistic Regression 0		
	P_NOT_BASIC	P_BASIC
not_basic	33	50
basic	26	200

K Neighbours 5		
	P_NOT_BASIC	P_BASIC
not_basic	9	74
basic	48	178

K Neighbours 4		
	P_NOT_BASIC	P_BASIC
not_basic	24	59
basic	84	142

Perceptron		
	P_NOT_BASIC	P_BASIC
not_basic	56	27
basic	100	126

Random Forest		
	P_NOT_BASIC	P_BASIC
not_basic	72	11
basic	7	219

Stacking Classifier		
	P_NOT_BASIC	P_BASIC
not_basic	56	27
basic	100	126

Curvas ROC

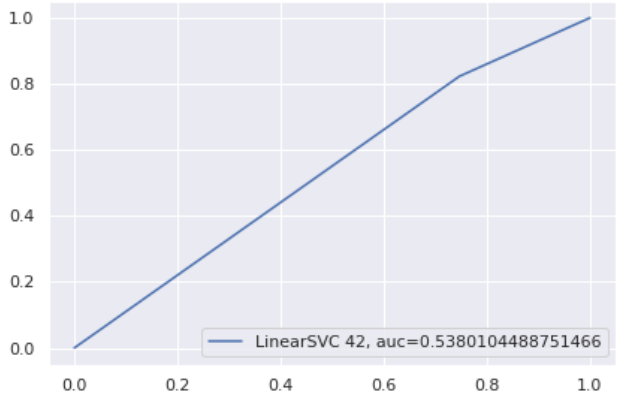

Figure 5: LinearSVC 42

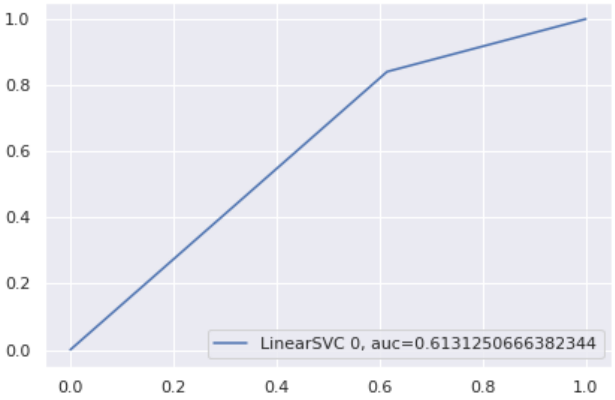

Figure 6: LinearSVC 0

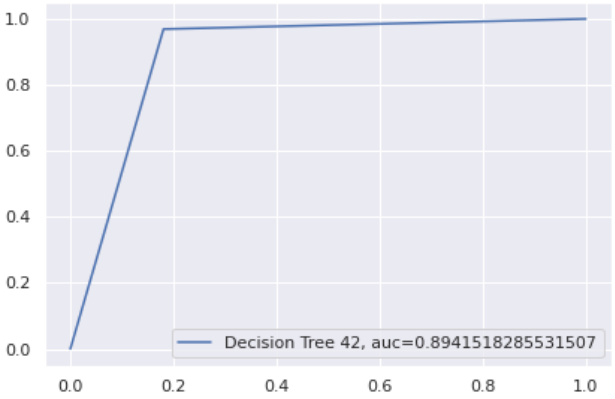

Figure 7: Decision Tree 42

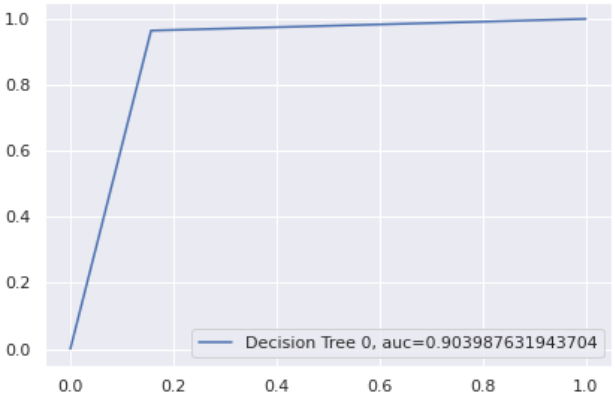

Figure 8: Decision Tree 0

Figure 9: Logistic Regression 42

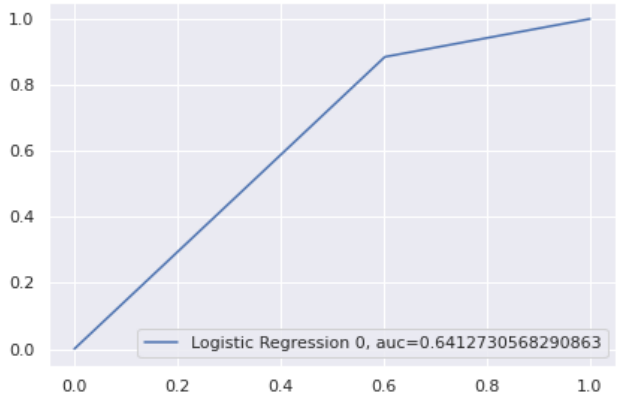

Figure 10: Logistic Regression 0

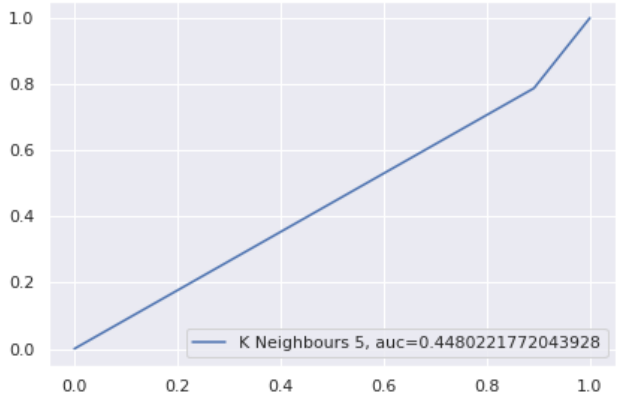

Figure 11: K-Neighbours 5

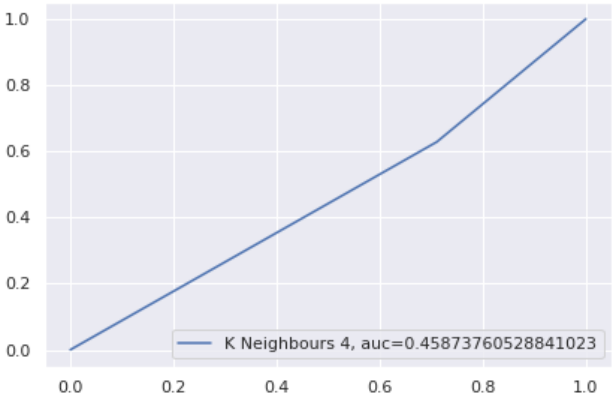

Figure 12: K-Neighbours 4

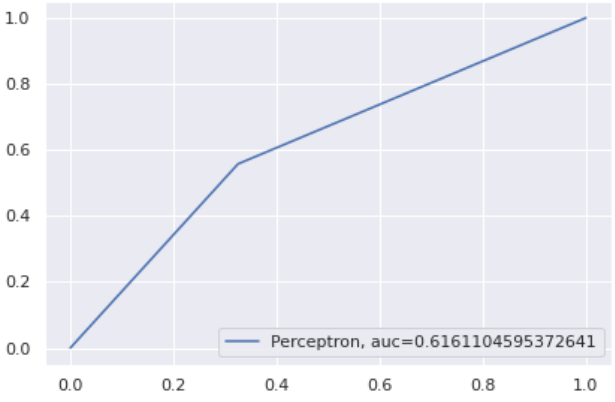

Figure 13: Perceptron

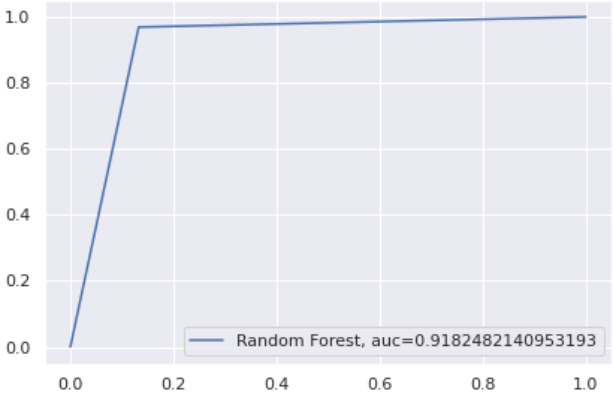

Figure 14: Random Forest

Resultados e Conclusões

Utilizando as informações acima, podemos concluir que os métodos de *Decision Tree* e *Random Forest* obtiveram os melhores resultados, com taxas de acerto na casa dos 90%.

Figure 15: Stacking Classifier

Enquanto os métodos de *K-Nearest Neighbors* obtiveram os piores resultados, com taxas na casa dos 60%.
O *Stacking Classifier* também não obteve resultados bons, sendo o modelo com a pior taxa de accuracy, e empatando na taxa de recall com o *Perceptron*. Talvez em um trabalho futuro possa-se reavaliar o método de stacking com outro *final_estimator* no lugar do padrão, *Logistic Regression*.
Também seria interessante testar outros valores de hiperparâmetros, principalmente no método *K-Nearest Neighbors*, aumentando a quantidade de vizinhos.
Os métodos de *Logistic Regression* não obtiveram diferença com hiperparâmetros diferentes, mas também seria interessante testar outros valores.
Dado que o dataset é baseado em um jogo e que seus valores são inventados, não necessariamente existe uma ligação direta entre eles, como por exemplo o tamanho de uma casa em relação ao seu valor de mercado. Os métodos *Decision Tree* e *Random Forest* mostraram que talvez exista sim uma relação.
As colunas *against_?* mostram quanto de dano cada Pokémon recebe de cada tipo, o que poderia ter facilitado para os modelos aprenderem. Como exemplo, os Pokémons do tipo *Grass* recebem o dobro de dano de ataques do tipo *Fire*, metade do dano de ataques to tipo *Water* e apenas um quarto de dano de ataques to tipo *Grass*. Logicamente isso não se reflete em todo o dataset, já que alguns Pokémons possuem tipos secundários que influênciam no dano que recebem.
De forma geral, este trabalho se mostrou extremamente interessante de ser feito ao explorar um mundo inventado com valores inventados pelos criadores do jogo. Como supracitado, futuramente, modificações podem ser feitas para aprimorar o modelo e também para aprender sobre outros métodos.