Análise de Algoritmos de Aprendizado de Máquina para Classificação de Pokémons

Arthur Gabriel
DCOMP, Universidade Federal de São João del-Rei, Brasil
arthurgabrielbd@gmail.com

Resumo

Este trabalho apresenta uma análise das principais técnicas de aprendizado de máquina aplicados ao problema de classificação de Pokémons com base em seu tipo. A ideia é utilizar como base de dados a informação oficial dos 718 Pokémons existentes tanto para o treinamento quanto para teste dos algoritmos. Seu objetivo é observar e analisar as vantagens e desvantagens da aplicação de cada técnica, além do erro gerado pela classificação errada.

1 Introdução

De acordo com Arthur Samuel (1959), aprendizado de máquina é a área de estudo que fornece aos computadores a habilidade para aprender sem ser explicitamente programados. De uma maneira mais formal pode-se dizer que um programa de computador aprende a partir de uma experiência E com respeito a uma classe de tarefas T e com medida de desempenho P, se seu desempenho P na tarefa T melhora com a experiência E [Mitchell, 1997].

A aplicação de técnicas de aprendizado de máquina tem se dado em diversas áreas de estudo como aprender a reconhecer palavras faladas, dirigir um veículo autônomo, classificar novas estruturas astronômicas, classificar objetos presentes em uma imagem. Estas aplicações utilizam principalmente do conceito de aprendizado supervisionado que é aquele onde é fornecido um conjunto de exemplos de treinamento para os quais o rótulo da classe associada (ou valor de saída) é conhecido.

Neste trabalho será abordado a análise de algumas das principais técnicas de aprendizado de máquina, sendo elas a Árvore de Decisão e Redes Neurais. Para a análise de Árvore de Decisão será usado dois algoritmos famosos: o ID3 e o CART. Para a análise de Redes Neurais será usado o algoritmo MLP (**multilayer perceptron**) com aprendizagem baseada em gradiente descendente.

2 Fundamentação Teórica

Para iniciar a discussão sobre as técnicas é importante ressaltar o tipo de aprendizado utilizado e o tipo do problema abordado. Para o tipo de aprendizado os algoritmos trabalham somente com o conceito de aprendizado supervisionado, ou seja, a entrada e saída desejadas são conhecidas e fornecidas por um supervisor (professor) externo. De forma análoga o tipo do problema é o mesmo para todos os algoritmos, sendo eles um problema de classificação, ou seja, tentar prever os resultados em uma saída discreta mapeando variáveis de entrada em categorias distintas.

2.1 Métodos de Aprendizado da Árvore de Decisão

A aprendizagem em árvore de decisão é um dos métodos mais amplamente utilizados e práticos para inferência indutiva. É um método para aproximar funções de valor discreto, e é robusto para dados ruidosos, além disso, é capaz de aprender expressões disjuntivas. Métodos de aprendizado da árvore de decisão procuram um espaço de hipóteses completamente expressivo e assim evitar as dificuldades de espaços de hipótese restritos. Seu viés indutivo é uma preferência por pequenas árvores sobre grandes árvores. A base do aprendizado destes algoritmos é a busca “top-down"(de cima para baixo) por todas as possíveis árvores utilizando um paradigma de busca “gulosa”. A figura 1 mostra um exemplo de árvore criada a partir do algoritmo ID3 para o problema “PlayTennis"apresentado em [Mitchell, 1997].

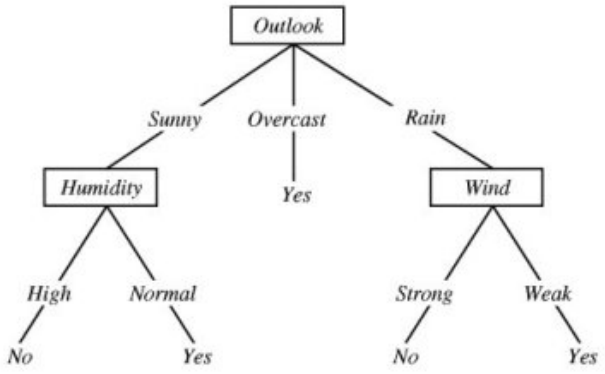

Figura 1: Árvore de Decisão para PlayTennis.

Algoritmo ID3

É o algoritmo desenvolvido por J. Ross Quinlan [Quinlan, 1986]. É baseado na ideia de entropia, ou seja, no grau de desordem dos dados. Dado um conjunto S que representa um

conjunto de instâncias, C a quantidade classes diferentes contidas em S, e p_i a proporção de uma classe dentro do conjunto S, a entropia pode ser calculada como:

$$E(S) = \sum_{i=1}^{C} -p_i \log_2 p_i$$

A partir disso é possível calcular o ganho de informação de cada atributo (característica) de acordo com a própria entropia do atributo para com as classes. Dado um conjunto A que representa um determinado atributo e S_v um subconjunto de S, no qual o atributo A possui valor V, o ganho de informação de cada atributo é calculado como:

$$Ganho(S, A) = E(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{S} E(S_v)$$

O atributo que possui maior ganho é inserido no nó mais alto que esteja livre na árvore (se estiver vazia é inserido na raiz) e seus possíveis valores criam novos ramos na árvore, se a entropia do ramo for zero, ou não possui atributos fora da árvore ele vira um nó folha com o valor de uma das saídas, senão o ramo refaz o processo de entropia para as instâncias restantes e recalcula o ganho de informação dos atributos restantes até que só exista nós folha no final de cada ramo.

O pseudocódigo em Algoritmo 1 para o ID3 foi retirado de [Mitchell, 1997].

Algorithm 1 ID3

Input: Exemplos, atributo Destino, Atributos
Output: Árvore

Crie um nó raiz para a árvore
if todos os exemplos são positivos **then**
    Retorne a raiz de árvore de nó único, com o rótulo = +.
end if
if todos os exemplos são negativos **then**
    Retorne a raiz de árvore de nó único, com o rótulo = -.
end if
if o número de atributos de previsão está vazio **then**
    Retorne a raiz da árvore de nó único, com label = valor mais comum do atributo de destino nos exemplos.
else
    A = O atributo que melhor classifica os exemplos.
    Atributo Árvore de Decisão para Raiz = A.
    for cada valor possível, v_i, de A **do**
        Adicione um novo galho de árvore abaixo de Root, correspondente ao teste A = v_i.
        Seja Exemplos (v_i) o subconjunto de exemplos que têm o valor vi para A.
        if Exemplos (v_i) está vazio **then**
            Abaixo dessa nova ramificação, adicione um nó folha com o valor alvo = label mais comum nos exemplos.
        else
            Abaixo desta nova ramificação, adicione a subárvore **ID3(Exemplos(v_i), atributo Destino, Atributos – {A})**.
        end if
    end for
end if
Retorna Raiz.

Algoritmo CART (Classification and Regression Trees)
O algoritmo CART é resultado de um trabalho publicado em [Leo Breiman, 1984].

O algoritmo sempre constroí uma árvore binária que é construída de forma similar a árvore em ID3, porém respondendo questões simples de sim/não. Diferente do ID3 o algoritmo CART usa o índice de Gini, criado por Conrado Gini em 1912, que mede a heterogeneidade das instâncias. Dados p_i a frequência relativa de cada classe em cada nó e C o número de classes, o cálculo do índice de Gini é dado por:

$$G = 1 - \sum_{i=1}^{C} p_i^2$$

Se o valor de G é igual a zero, então o nó é puro. Por outro lado, quando ele se aproxima de um, o nó é impuro (aumenta o número de classes uniformemente distribuídas neste nó). Quando, o critério de Gini é utilizado tende-se a isolar num ramo os registros que representam a classe mais frequente.

O algoritmo sempre faz uma divisão binária. Suponha que um determinado atributo é composto por M distintas categorias, o conjunto S de possíveis desdobramentos desse nó é dado por $2^{M-1} - 1$. A melhor divisão S do nó é dada pela redução da impuridade em dividir o nó em filhos direito e esquerdo. A melhor divisão para um atributo, é a que tem a menor impuridade de todas as divisões possíveis $2^{M-1} - 1$ para o atributo. Para o critério de parada o algoritmo cresce a árvore até a saturação. Depois de dividir o nó raiz, repetimos o mesmo processo para os nós filhos nesse caso 2, pois o CART só realiza partições binárias. Este processo de divisão sequencial de construir uma árvore de camada por camada, é chamado de particionamento recursivo. A árvore binária é dividida até que nenhum dos nós possa ser dividido mais, até que os nós sejam os mais puros possíveis, ou podemos definir um número mínimo de exemplos para que o nó pare, mas é preferível deixar crescer até a saturação. Mas esses dados podem estar super ajustados ao conjunto de treino, e conter nós com baixa significância estatística, por isso depois de gerada a árvore é necessário podá-la.O algoritmo usa a chamada regra da pluralidade, onde a classe associada ao nó terminal é a classe que pertence a maioria dos exemplos atribuídos aquele nó. Para o método de podar a árvore o algoritmo usa uma medida chamada taxa de erro ajustada. Esta medida incrementa cada taxa de classificação errada de cada nó para o conjunto de treinamento pela imposição de uma penalidade baseada no número de folhas da árvore. O objetivo é podar primeiro os ramos que possuem um menor poder preditivo por folha. A fórmula da taxa de erro ajustada é:

$$EA(T) = E(T) + \alpha ContadorFolhas(T)$$

Onde α é um fator de ajuste incrementado gradualmente para criar novas subárvores. Para encontrar a primeira subárvore, as taxas de erro ajustadas para todas as possíveis subárvores contidas no nó raiz são avaliadas com α sendo gradualmente aumentado. Quando a taxa de erro ajustada para alguma subárvore for menor do que a taxa da árvore completa, então tem-se a primeira subárvore candidata. Todos os ramos que não fazem parte desta subárvore são eliminados e então o processo se reinicia a partir desta subárvore para se obter uma segunda. O processo termina quando todos os caminhos até o nó raiz forem podados. Cada uma das subárvores são candidatas a fazerem parte do modelo final. O próximo passo é escolher dentre as subárvores, aquela que classifica melhor

novos dados por meio de um conjunto de validação. Isto é, a árvore que classificar os novos registros com a menor taxa de erro é a escolhida[Juliana M. Barbosa, 2015].

2.2 Método de Aprendizado de Rede Neural

Redes Neurais Artificiais (RNA) são sistemas paralelos distribuídos compostos por unidades de processamento simples (nodos) que calculam determinadas funções matemáticas (normalmente não-lineares). Tais unidades são dispostas em uma ou mais camadas e interligadas por um grande número de conexões, geralmente unidirecionais. Na maioria dos modelos estas conexões estão associadas a pesos, os quais armazenam o conhecimento representado no modelo e servem para ponderar a entrada recebida por cada neurônio da rede. O funcionamento destas redes é inspirado no cérebro humano.

A arquitetura das RNAs criam a possibilidade de um desempenho superior ao dos modelos convencionais. A capacidade de aprender através de exemplos e de generalizar a informação aprendida é, sem dúvida, o atrativo principal da solução de problemas através de RNAs. A generalização, que está associada à capacidade de a rede aprender através de um conjunto reduzido de exemplos e posteriormente dar respostas coerentes para dados não conhecidos, é uma demonstração de que a capacidade das RNAs vai muito além do que simplesmente mapear relações de entrada e saída. As RNAs são capazes de extrair informações não apresentadas de forma explícita através dos exemplos. Não obstante, as RNAs são capazes de atuar como mapeadores universais de funções multivariáveis, com custo computacional que cresce apenas linearmente com o número de variáveis. Outra característica importante é a capacidade de auto-organização e de processamento temporal, que, aliada àquelas citadas anteriormente, faz das RNAs uma ferramenta computacional extremamente poderosa e atrativa para a solução de problemas complexos. A figura 2 ilustra a representação de uma rede MLP genérica.

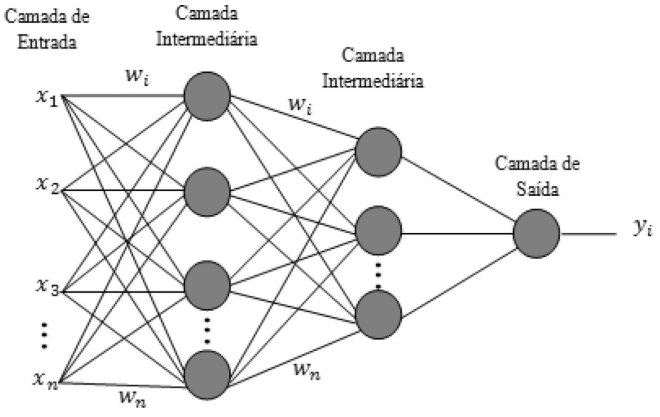

Figura 2: Exemplo de Rede Neural.

Algoritmo MLP

A rede perceptron de múltiplas camadas (multilayer perceptron) é a mais popular e simples dentre todas as redes neurais de camada intermediária, seu treinamento baseado no gradiente descendente possibilita a solução de problemas não linearmente separáveis. Este método de treinamento ficou conhecido como algoritmo de **Backpropagation** (retropropagação do erro).

Durante a inicialização da rede o valor dos pesos de cada camada são gerados aleatoriamente e as camadas intermediárias são geradas como produto da multiplicação entre a matriz de pesos e matriz da camada, ambos anteriores a ela. O processo se repete até a camada de saída. Durante o processo de cálculo das camadas é ainda calculado por cima do valor resultante da multiplicação das matrizes a função de ativação. Neste caso varia muito da função de ativação que está sendo utilizada, podendo ser ela, por exemplo a sigmoidal, a reLU ou a tangente hiperbólica.

Após o cálculo das camadas o algoritmo faz o processo de backpropagation calculando o erro gerado em cada camada. Primeiramente é calculado do erro na última camada, que é simplesmente a diferença entre a saída desejada y_{real} e a saída atual da rede y_{atual} no tempo (época) t, em forma de equação, pode ser visto como:

$$erro(t) = y_{real} - y_{atual}(t)$$

Para o cálculo dos erros das camadas anteriores é feita uma multiplicação entre os pesos da camada atual, o erro da camada anterior e a derivada da função de ativação g' dado a camada atual $a^{(i)}$, portanto a fórmula é:

$$erro^{(i)}(t) = (pesos^{(i)})^T * erro^{(i+1)}(t). * g'(a^{(i)})$$

Esse processo é feito de forma iterativa por um número de vezes popularmente chamado de época (**epochs**). A atualização dos pesos é dado pela fórmula:

$$w(t+1) = w(t) + \eta\delta_j\chi_i$$

O valor de w(t) corresponde a matriz de peso, η corresponde a taxa de aprendizado, δ_j corresponde ao erro e χ_i corresponde a camada intermediária de índice i.

3 Desenvolvimento do Trabalho

A utilização de uma base de dados composta por muitos atributos e instâncias é muito bom para adicionar mais informações ao algoritmo e fazê-lo ter uma experiência mais sólida, porém somente adicionar dados sem a análise prévia do mesmo não implica na melhora do algoritmo. É importante sempre verificar se existe ganho real em um dado atributo ou instância. Para isso é feito uma análise da base de dados.

3.1 Análise da Base de Dados

A base de dados utilizada consiste em uma tabela em formato **Comma-separated values** (CSV) contendo 718 instâncias que descrevem cada espécie de Pokémon divididas dentre os 18 possíveis tipos e seus respectivos atributos. Todas as informações foram tiradas de [BUL, 2002] e [Chris Tapsell, 2019].

Os atributos utilizados estão relacionados aos dados oficiais divulgados em jogos da franquia Pokémon. Nesta base de dados foram coletados sete tipos de características:

1. **Habitat**: Referente ao local onde o Pokémon é encontrado. A hipótese é que certos tipos de Pokémons tem tendência a se localizarem em um ambiente específico como, por exemplo Pokémons de água são encontrados em oceanos e lagos.

2. **Cor**: Referente a cor predominante do Pokémon. A hipótese é que certos tipos terem uma tendência a possuir a mesma cor como, por exemplo Pokémons de fogo tendem a ser vermelhos.
3. **Formato do Corpo**: Referente a estrutura corporal do Pokémon. O fato de certos tipos de Pokémon terem a mesma estrutura corporal como, por exemplo Pokémons voadores possuem apenas um par de asas.
4. **Resistente a**: Referente a resistência de um tipo de Pokémon a ataques de outro. Um exemplo é a resistência do tipo pedra a ataques do tipo elétrico.
5. **Vulnerável a**: Referente a vulnerabilidade de um tipo de Pokémon a ataques de outro. Um exemplo é a vulnerabilidade do tipo pedra a ataques do tipo água.
6. **Forte contra**: Referente a alta efetividade de um ataque de um tipo de Pokémon a outro.
7. **Fraco contra**: Referente a baixa efetividade de um ataque de um tipo de Pokémon a outro.

Para verificar o relacionamento e a correlação entre os atributos foi construído uma matriz de correlação. A figura 3 mostra o resultado obtido.

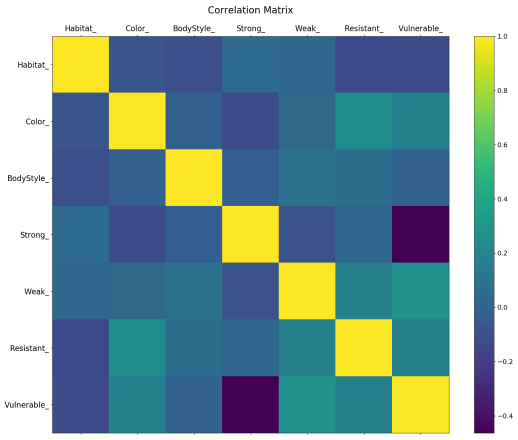

Figura 3: Matriz de correlação entre os atributos.

A matriz de correlação mostra valores entre -1.0 e 1.0, onde valores próximos a 1.0 mostram atributos que apresentam a mesma informação, enquanto valores próximos a -1.0 mostram atributos com informações contrárias. A partir da matriz é possível concluir que não há atributos fortemente correlacionados, ou seja cada atributo fornece informações diferentes e contribuem positivamente para as técnicas de aprendizado gerarem bons resultados.

3.2 Metodologia Utilizada

Para a realização dos experimentos e análise os algoritmos CART, ID3 e MLP foi usado algoritmos já implementados da biblioteca Scikit-Learn e decision-tree-id3 em Python, além disso, o algoritmo MLP foi implementado em Python.

A utilização do conjunto de dados é feita com a separação entre dados de treinamento e dados de teste de maneira **holdout**, ou seja, o conjunto de treinamento e teste é o mesmo durante todo o processo de validação do erro. Os dados são separados na forma de sorteio randômico.

A utilização dos algoritmos se dá de forma bem padrão, no caso do algoritmo de CART é utilizado o índice de Gini, para verificar a impureza dos dados e a criação de uma árvore binária, assim como a literatura indica. O ID3 também é implementado como a literatura indica, utilizando a entropia e o ganho de um atributo dado os outros.

A rede MLP possui uma arquitetura totalmente conectada, ou seja, todos os nodos de uma camada esão conectados com os de outra. Foi construída tendo como base alguns parâmetros que permanecem fixos durante o processo de treinamento do algoritmo, o nome do parâmetro e o valor respectivo estão sendo mostrados na tabela 1.

Parâmetro	Valor
função de ativação	reLU
taxa de aprendizado	0.001
qtd. camadas ocultas	1

Tabela 1: Tabela de parâmetros fixos.

4 Experimentos Iniciais e Análise dos Resultados

Nesta seção será abordado os experimentos realizados e a respectiva análise dos resultados de cada um. É importante destacar que para simplificar o problema será considerado que cada Pokémon possui apenas um tipo. O conjunto de dados é dividido da mesma forma para todos os algoritmos, dessa forma os dados são divididos entre treinamento e teste até o final do treinamento.

4.1 Algoritmo CART

A análise deste algoritmo foi feita com base na comparação do erro gerado pela imprecisão da saída em relação ao crescimento da proporção da base de treinamento em relação ao tamanho total da base de dados. A ideia é que quanto maior for a proporção dos dados de teste, então menor será os dados para treinamento, limitando a capacidade de generalização do algoritmo. A figura 4 mostra o resultado do experimento, apresentando tanto a linha dos dados já treinados (Treinamento) quanto a linha de dados "novos"(Teste).

A análise do gráfico mostra que o aumento dos dados de teste o algoritmo perde essa capacidade de generalizar, e com isso sua aprendizagem fica obsoleta.

É interessante observar que a rede teve um resultado excelente, com um erro de 0%, tanto na validação dos dados de treinamento quanto os de teste com proporção 0.2 de teste e 0.8 de treinamento. Além disso, a linha de treinamento fica com um erro de 0% constante, dado isso é possível afirmar que a estrutura da árvore aprendida está se ajustando muito bem aos dados.

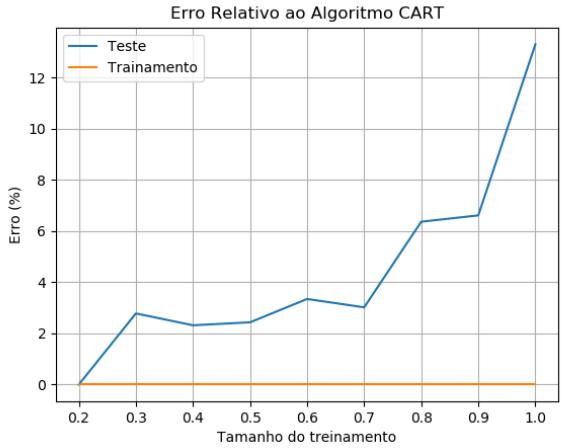

Figura 4: Erro relativo ao Algoritmo CART.

4.2 Algoritmo ID3

Assim como o algoritmo CART, os experimentos feitos buscam a redução da taxa de erro. A figura 5 mostra os resultados dos testes.

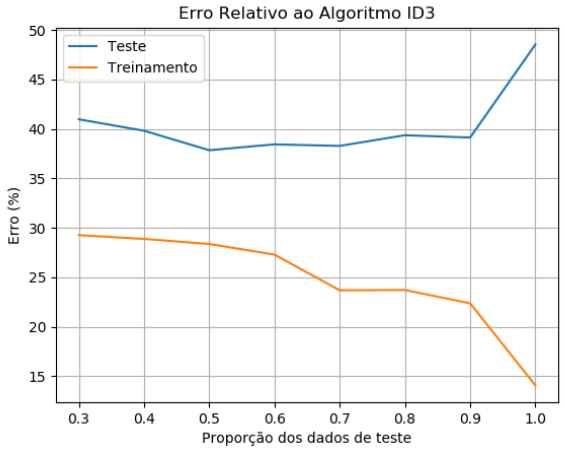

Figura 5: Erro relativo ao Algoritmo ID3.

Os resultados obtidos estão bem abaixo do esperado, estando em um intervalo de 40% a 50% para os dados de teste e 15% a 30% para os dados de treinamento. Os resultados apontam que o algoritmo não conseguiu aprender uma estrutura de árvore adequada para os dados.

4.3 Algoritmo MLP

O algoritmo de redes de múltiplas camadas será analisado de forma que o objetivo é diminuir a taxa de erro. A verificação da redução da taxa de erro é dada pelo aumento da quantidade de épocas (iterações) necessárias para que o método convirja a um ponto de erro mínimo. O treinamento excessivo da rede gerado por muitas iterações geram um problema de "overfitting", fazendo com que a rede perca a capacidade de generalização, aumentando a taxa de erro para o teste.

A figura 6 mostra o resultado do experimento, apresentando duas linhas que representam o erro gerado pelo treinamento e pelo teste.

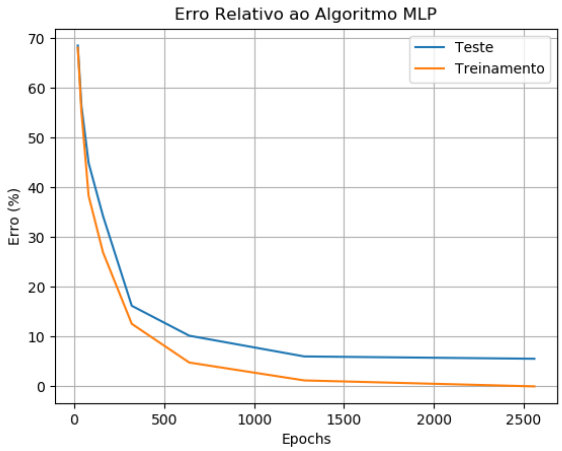

Figura 6: Erro relativo ao Algoritmo MLP.

De acordo com o gráfico é possível observar o declínio da taxa de erro com o aumento das épocas nas duas linhas. O valor do erro se estabiliza em menos de 10% para o teste por volta de 2000 épocas, o que permite concluir que o método convergiu com este valor. O possível aumento a partir do valor de convergência resultaria em um "overfitting", que não é desejado.

5 Conclusão

A utilização dos algoritmos de aprendizado de máquina para classificação de Pokémons, pela base de dados que foi apresentada, mostrou resultados satisfatórios.

Os resultados obtidos pela Árvore de Decisão CART mostram um modelo mais simples que mesmo assim, consegue atuar sobre o problema e exaltar a característica de robustez a dados ruidosos já que consegue manter a taxa de erro em 0% para treinamento mesmo com a diminuição no tamanho dos dados para treinamento. Além disso, apresenta uma taxa de erro baixa para os dados de teste. De fato, possui os melhores resultados dentre as técnicas apresentadas.

Os resultados obtidos pela Árvore de Decisão ID3 não foram satisfatórios. O modelo apresenta uma taxa de erro muito grande quando comparado aos outros. Apesar de ter uma implementação simples e robusta, a estrutura da árvore não se ajustou bem ao conjunto de dados.

Os resultados obtidos pelo algoritmo MLP estão satisfatórios. O erro fica em torno de 5% a 10%, e isso pode ser explicado pela pouca quantidade de instâncias. Por ser uma técnica mais robusta, com um poder computacional muito maior que a Árvore de Decisão, e pelo fato dos resultados estarem piores que os da CART, faz com que a utilização de redes neurais não seja uma técnica tão atrativa para o problema em questão.

Referências

[BUL, 2002] https://bulbapedia.bulbagarden.net/wiki/Main_Page, 2002. [Online; accessed 17-November-2019].

[Chris Tapsell, 2019] Staff Writer Chris Tapsell. https://www.eurogamer.net/articles/2018-12-21-pokemon-go-type-chart-effectiveness-weaknesses, 2019. [Online; accessed 17-November-2019].

[Juliana M. Barbosa, 2015] Andrea I. Tavares Juliana M. Barbosa, Tiago G. de S. C. Métodos de classificação por Árvores de decisão disciplina de projeto e análise de algoritmos. 2015.

[Leo Breiman, 1984] Charles J. Stone R.A. Olshen Leo Breiman, Jerome Friedman. *Classification and Regression Trees*. Taylor Francis, 1984.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Quinlan, 1986] J. Quinlan. *Machine Learning*. Springer US, 1986.