ANÁLISE DE MÉTODOS DE CLASSIFICAÇÃO PARA TAREFAS DE MINERAÇÃO DE DADOS

Júlio César Machado Álvares [(1)], Marcus Vinícius Rodrigues Campos [(1)], Marcos Roberto Ribeiro [(2)]

RESUMO

O processo de extração de conhecimento de bases de dados é um processo um tanto quanto complexo e é uma área de estudo crescente da computação atual. O processo é conhecido como KDD (*Knowledge discovery database*) e é composto por várias fases, sendo o seu conjunto, o procedimento de extração de conhecimento. A fase que será abordada no presente artigo trata-se da mineração dos dados, mais precisamente, a classificação dos mesmos. Serão abordados três diferentes classificadores, sendo esses uma Rede Neural Artificial, KNN e *Naive Bayes*. Os mesmos serão submetidos a duas bases de dados e a diferentes parâmetros, buscando diferentes níveis de precisão.

Palavras-chave: *Data Mining*. Classificação. *Neural Network*. *Naive Bayes*. KNN

1 INTRODUÇÃO

Com diversos problemas surgindo a partir do avanço desenfreado da computação nos dias de hoje, várias técnicas são criadas todos os dias para contornar tais problemas.

Um desses é o crescimento absurdo da quantidade de dados que são gerados todos os dias. O problema criado com tal crescimento é o fato de que nem sempre as bases de dados trazem conhecimentos explícitos consigo. Assim, é necessário aplicar técnicas para extrair o conhecimento das bases *in natura*.

Para a solução de tal problema, foi proposta uma metodologia, chamada de KDD (*Knowledge discovery database*), onde a mesma trata-se de um conjunto de técnicas e ações para extrair da melhor forma possível uma quantidade de conhecimento das bases de dados.

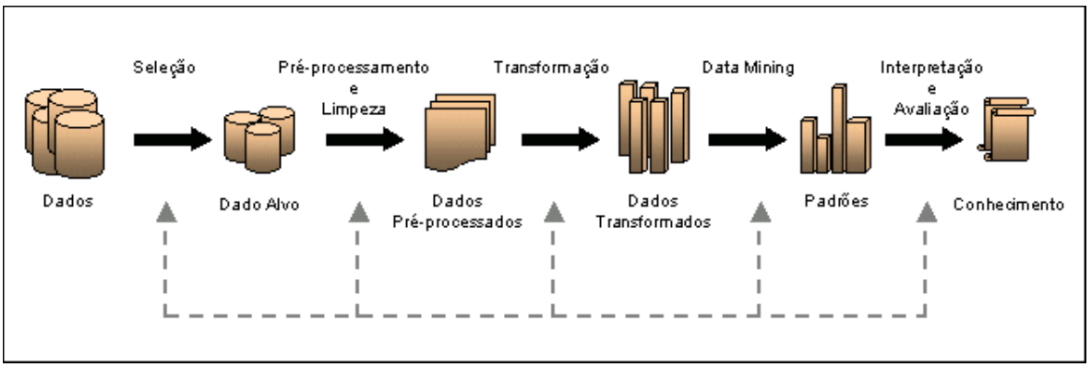

Figura 1 – Processo de *KDD* (PRASS, 2016).

O KDD trata-se de um processo não trivial de extração de informações implícitas, previamente desconhecidas e potencialmente úteis, a partir de dados armazenados em um banco de dados (PRASS, 2016). A Figura 1 demonstra o processo completo. Pode-se notar que o início é dado a partir de uma base sem nenhum tratamento (*in natura*) e o seu resultado é o conhecimento implícito na mesma.

A área de interesse do presente trabalho está na quarta parte do processo de KDD, a mineração dos dados.

1.1 Bases de dados

Para este trabalho, foram utilizadas duas bases de dados. A primeira delas foi o *Iris Flower Dataset*, que é muito utilizada para testes de técnicas de classificação estatística e *machine learning* (como SVM, dentre outros). Ela consiste de diversas amostras de três espécies de íris (*Iris setosa, Iris virginica and Iris versicolor)*, e quatro *features*, largura, altura, sépalas e pétalas (LEARNING; SYSTEMS, 2018).

A segunda base de dados, denominada *Hill-Valley dataset*, trata-se de um conjunto com 100 (cem) *features* para cada amostra, e as amostras representam picos e vales. Cada *feature* representa um ponto em um plano cartesiano de duas dimensões. Foi criada com intuito principal de classificação, apenas e contém apenas valores numéricos (LEARNING; SYSTEMS, 2018).

1.2 *Machine Learning*

Aprendizado de máquina (do Inglês, Machine Learning) é um método de análise de dados que automatiza a construção de modelos analíticos. É um ramo da Inteligência Artificial baseado na ideia de que sistemas podem aprender com dados e tomar decisões com o mínimo de intervenção humana. Dentre os métodos usados para Machine Learning, temos:

Algoritmos de Aprendizado Supervisionado: Estes são treinados por meio de exemplos rotulados como uma entrada na qual a saída desejada é conhecida. Um bom exemplo é um conjunto de dados de frases, marcadas como positivas, ofensivas ou neutras e treinar um classificador de análise de sentimento.

Algoritmos de Aprendizado não-Supervisionado: É quando os algoritmos têm que predizer uma resposta correta, sem ter um conhecimento prévio sobre os dados a serem analisados.

Aprendizado por Reforço: É mais utilizado em robótica e jogos. O algoritmo procura através de testes como "tentativa e erro"que ações rendem mais recompensas. (SAS, 2018)

1.2.1 Redes Neurais Artificiais

As redes neurais surgiram, primeiramente, com a reprodução de um neurônio humano. Trabalho desenvolvido por *Warren McCulloch* e *Walter Pitts*, em 1943. O intuito disso era desenvolver um modelo matemático que descrevesse o comportamento dos neurônios do cérebro humano, assim, criando uma rede de neurônios, reproduziria-se o cérebro.

A motivação por trás desse e dos trabalhos desenvolvidos posteriores ao mesmo, vem do poder incalculável do cérebro humano. Portando cerca de 10^{11} neurônios que processam e se comunicam com milhares de outros continuamente e de forma paralela. A estrutura de tais neurônios dá aos pesquisadores a base para o desenvolvimento das redes neurais.

O resultado da pesquisa de *McCulloc e Pitts* resultou no modelo de neurônio MP43. Tal modelo tem n terminais de entrada (dendritos) que recebem valores $x_1, x_2, ..., x_n$ e apenas um terminal de saída y, representando o axiônio. Juntamente com vetores de pesos e multiplicadores como épocas e taxa de aprendizado, o neurônio computa a entrada e gera uma saída. (BRAGA; CARVALHO; LUDERMIR, 2000)

1.2.1.1 Função *Sigmoid*

A função de ativação *Sigmoid* trata-se de uma função não linear com curva suave. Sua maior vantagem perante as outras é que a mesma não é linear. Outra vantagem interessante da função *sigmoid*, é que os valores convergem para os extremos, inferiores ou superiores, qualidade desejável quando se deseja classificar alguma amostra específica.

Porém, essa função apresenta um problema. Quando os gradientes da rede se tornam muito pequenos, ou seja, se aproximando de 0, a rede não está realmente aprendendo. Além disso, seu domínio é apenas positivo, característica que nem sempre é desejada (ACADEMY, 2018).

$$f(x) = \frac{1}{1 + e^{-x}}$$

1.2.1.2 Função *ReLU*

A função *ReLU* é amplamente utilizada atualmente, em redes neurais complexas.

Uma das principais vantagens da *ReLU* é que, os valores negativos serão convergidos a 0, fazendo com que nem todos os neurônios da rede sejam ativados, diminuindo a complexidade da rede. Porém, ela apresenta o mesmo tipo de problema da função *Sigmoid* quando os gradientes se aproximam de 0 (ACADEMY, 2018).

$$f(x) = max(0, x)$$

1.2.1.3 Função *TanH*

A função *TanH* trata-se de um escalonamento da função *sigmoid*, mas simétrica à origem e variando de -1 a 1.

Basicamente, essa função soluciona os problemas de valores da *sigmoid*, também é contínua, diferenciavel e não linear (ACADEMY, 2018).

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

1.2.1.4 Função *Degrau*

A função Degrau, também conhecida como *Binary Step Function* trata-se de uma função extremamente simples, que é usada principalmente quando os dados são binários, e que consiste em verificar o valor da saída da rede com o *threshold*, se a saída for maior, será a classe 1, se não, a classe 0. Tal função é tão simples que é usada, normalmente, apenas na academia, além de que ela só consegue classificar dados de 2 (duas) classes (ACADEMY, 2018).

$$f(x) = 0, if x < 0$$

$$f(x) = 1, if x >= 0$$

1.2.2 *Naive Bayes*

Classificadores bayesianos são uma família de "classificadores probabilísticos"simples baseados em aplicação do teorema de Bayes com fortes suposições independentes entre *features*. Por exemplo, uma fruta pode ser considerada uma maçã, se for vermelha, redonda e com um diâmetro de 6 centímetros. Um classificador bayesiano considera cada uma dessas "*features*"contribuindo independentemente com a probabilidade de que a fruta é uma maçã, sem importar com quaisquer relação entre as *features* (ZAKI; JR; MEIRA, 2014).

$$P(\frac{C_k}{x}) = \frac{p(C_k)p(\frac{x}{C_k})}{p(x)}$$

1.2.3 *K Nearest Neighbors*

K Nearest Neighbors é um algoritmo simples que prevê a classe de um determinado ponto X pela maioria de classes em uma distância K do próprio ponto, ou seja, seus "vizinhos". As medidas de similaridade podem ser várias, como a distância euclidiana Manhattan, e outras. (ZAKI; JR; MEIRA, 2014) A probabilidade condicional é dada por:

$$P(ci|x) = \frac{\frac{Ki}{nV}}{\sum_{j=1}^{k} \frac{Ki}{nV}} = \frac{Ki}{nV}$$

E finalmente, a classe prevista para x é :

$$y = argmax P(ci|x) = argmax \frac{Ki}{K}$$

1.3 Métodos de validação para modelos de classificação

O método de validação utilizado foi o de validação cruzada. Nesta abordagem, cada registro é usado o mesmo número de vezes para treinamento e exatamente uma vez para teste. Para melhor visualização, imagine o seguinte exemplo: Existem dois subconjuntos de tamanho igual. Primeiro, escolhe-se um dos subconjuntos para treinamento, e o outro para teste. Depois, trocam-se os papeis dos subconjuntos. Esta abordagem é chamada de validação cruzada de duas partes. O erro total é obtido pela soma dos erros de ambas execuções. (TAN et al., 2007)

O algoritmo para *cross validation* é apresentado abaixo:

$nFolds \leftarrow input()$
$fold \leftarrow 0$
while $fold < nFolds$ **do**
    train, test $\leftarrow data.split(1/nFolds)$
    model.train(train)
    result $\leftarrow result + model.predict(test)/lengthTest$
    fold $\leftarrow fold + 1$
end
acurracy $\leftarrow result/lengthResult$

Algorithm 1: Algoritmo para realização de *cross validation*.

2 DESENVOLVIMENTO

Para o desenvolvimento do presente trabalho, após a escolha das bases de dados, foi implementado toda uma Rede Neural *Perceptron*, utilizando a linguagem de programação

Python, com auxílio da biblioteca *numpy* (NUMPY, 2018). Também foi implementado um *script* para controle dos dados, leitura e escrita de arquivos.

A implementação dos outros classificadores (KNN e *Naive Bayes*) foi feita utilizando a biblioteca *Scikit-Learning*, com a linguagem de programação *Python* (SKLEARN, 2018).

Por fim, a plotagem dos gráficos foi feito utilizando o pacote *Matplotlib* do *Python*, juntamente com o pacote *Seaborn*, para estilização dos gráficos (MATPLOTLIB, 2018).

Para gerar os resultados do presente trabalho, foram montados 4 (quatro) experimentos, sendo esses, 2 (dois) envolvendo a Rede Neural, 1 (um) envolvendo o KNN e 1 (um) envolvendo o *Naive Bayes*. Tais experimentos foram numerados de 1 (um) a 4 (quatro), respectivamente e serão apresentados a seguir.

1) O objetivo desse experimento é observar a convergência da acurácia da rede neural variando a quantidade de épocas de treino e fazendo um teste. O experimento consistiu em variar de $[0, 300]$, deixando o valor de *threshold* padrão em -1 e testar o modelo em todas e para todas as funções de ativação, também capturando seu tempo de execução.

2) O objetivo desse experimento é observar a convergência da acurácia da rede neural variando o *threshold* da rede, utilizando um valor padrão de épocas de treino. Tal valor foi de 50 (cinquenta) épocas e a variação do *threshold* foi feita de $[-2, 2]$ com intervalos de 0.1, também capturando seu tempo de execução.

3) O objetivo desse experimento é observar a convergência da acurácia do KNN variando a quantidade de K-vizinhos. O experimento consistiu em variar a quantidade de K-vizinhos de $[0, 500]$ para a base de dados Hill e $[0, 140]$ para a base de dados Íris e testar o modelo para todos os cenários, também capturando seu tempo de execução.

4) O objetivo desse experimento é observar a convergência da acurácia do *Naive Bayes* variando o *alpha*. O experimento consistiu em variar o *alpha* de $[0, 20]$ com intervalos de 0.1 e testar o modelo em todas os cenários, também capturando seu tempo de execução.

Para os experimentos, todos os testes foram feitos diretamente no classificador, ou seja, não houve utilização da metodologia de validação cruzada. Tal decisão deve-se ao fato que a validação é muito custosa computacionalmente e que os testes diretos mostram o valor concreto de acurácia para o momento. Todas os dados de treino e teste foram iguais para todos os experimentos. A captura dos tempos de execução foi feita utilizando a biblioteca *time* do *Python* (PYTHON, 2018).

Algoritmos que foram implementados para a realização dos experimentos:

$ActualEpoch \leftarrow 0$
$epochs \leftarrow 300$
while $ActualEpoch < epochs$ **do**
    model $\leftarrow Perceptron(ActualEpoch, -1)$
    model.train(data)
    result $\leftarrow result + model.predict(test)/lengthTest$
    ActualEpoch $\leftarrow ActualEpoch + 1$
end
acurracy $\leftarrow result/lengthResult$

Algorithm 2: Algoritmo para realização do experimento 1.

$threshold \leftarrow -2$
while $threshold < 2$ **do**
    model $\leftarrow Perceptron(50, threshold)$
    model.train(data)
    result $\leftarrow result + model.predict(test)/lengthTest$
    threshold $\leftarrow threshold + 0.1$
end
acurracy $\leftarrow result/lengthResult$

Algorithm 3: Algoritmo para realização do experimento 2.

$neighbors \leftarrow 1$
$MaxNeighbors \leftarrow lengthDados$
while $neighbors < MaxNeighbors$ **do**
    model $\leftarrow KNN(neighbors)$
    result $\leftarrow result + model.predict(test)/lengthTest$
    neighbors $\leftarrow neighbors + 1$
end
acurracy $\leftarrow result/lengthResult$

Algorithm 4: Algoritmo para realização do experimento 3.

3 RESULTADOS

A partir dos experimentos, foram gerados os dados e os mesmos foram plotados em gráficos e apresentadaos em tabelas, dispostos a seguir.

$alpha \leftarrow 1$
while $alpha < 20$ **do**
    model $\leftarrow NaiveBayes(alpha)$
    result $\leftarrow result + model.predict(test)/lengthTest$
    alpha $\leftarrow alpha + 0.1$
end
acurracy $\leftarrow result/lengthResult$

Algorithm 5: Algoritmo para realização do experimento 4.

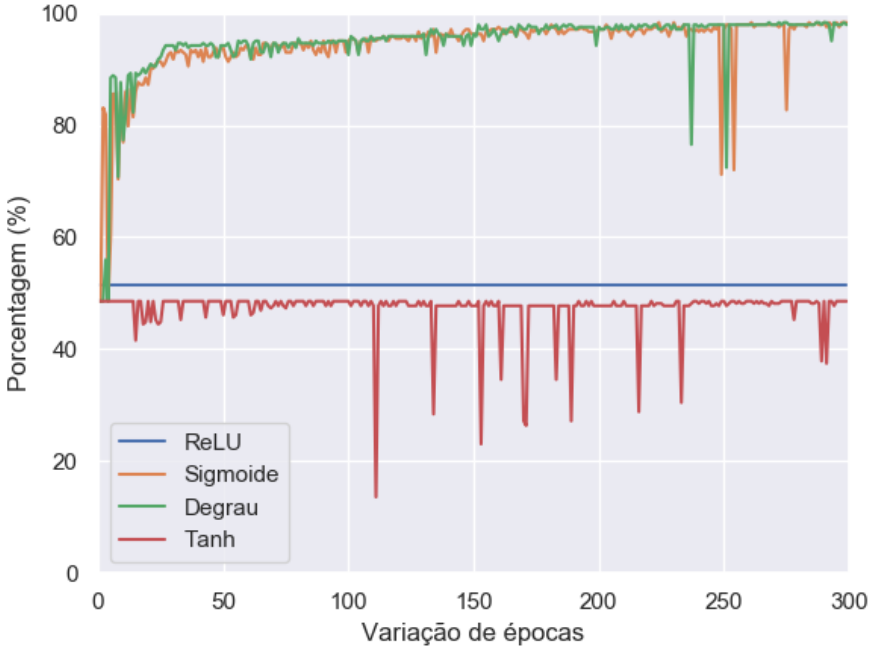

Figura 2 – Variação da acurácia do modelo Rede Neural pela variação da quantidade de épocas de treino para a base de dados Hill.

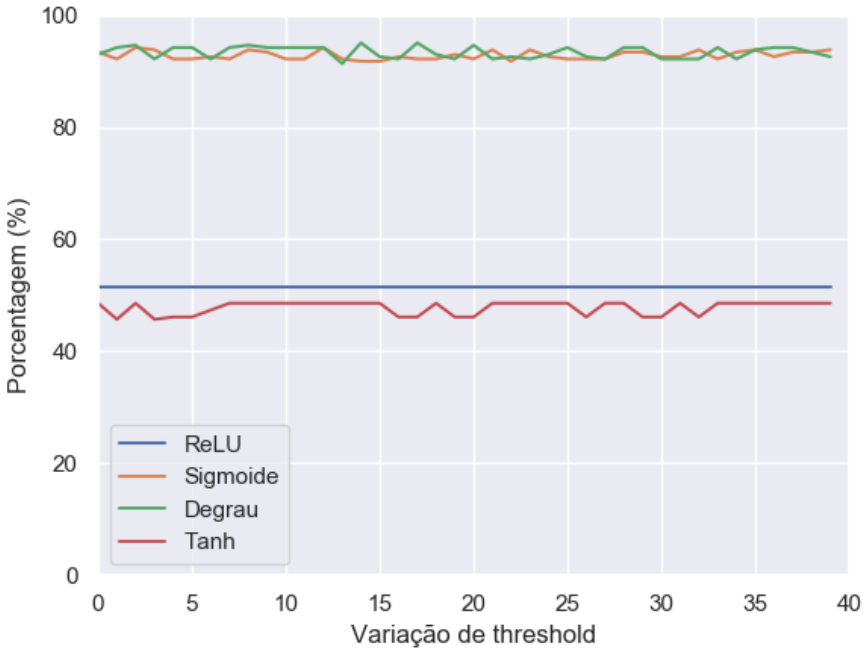

Figura 3 – Variação da acurácia do modelo Rede Neural pela variação do *threshold* de treino para a base de dados Hill.

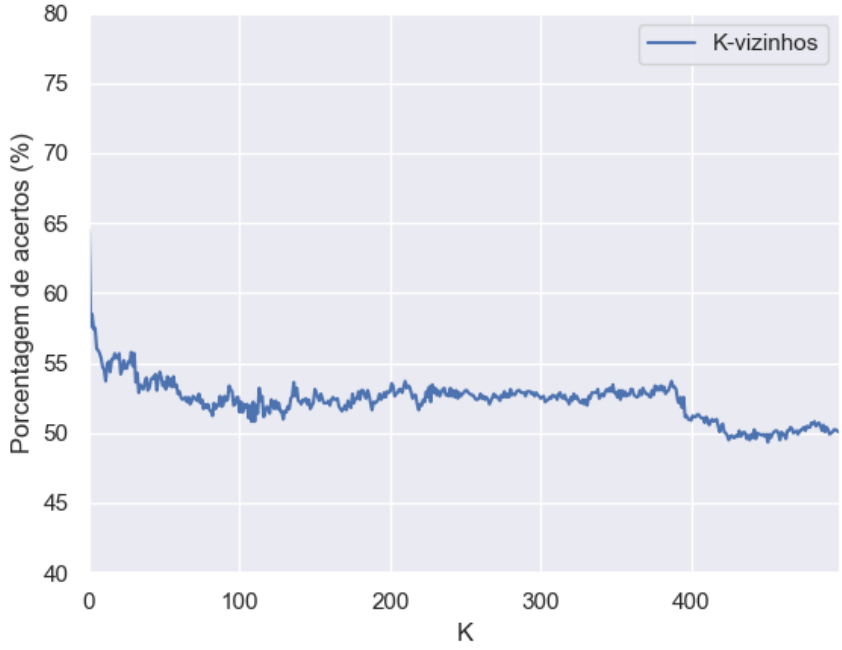

Figura 4 – Variação da acurácia do modelo KNN pela variação dos K-vizinhos para a base de dados Hill.

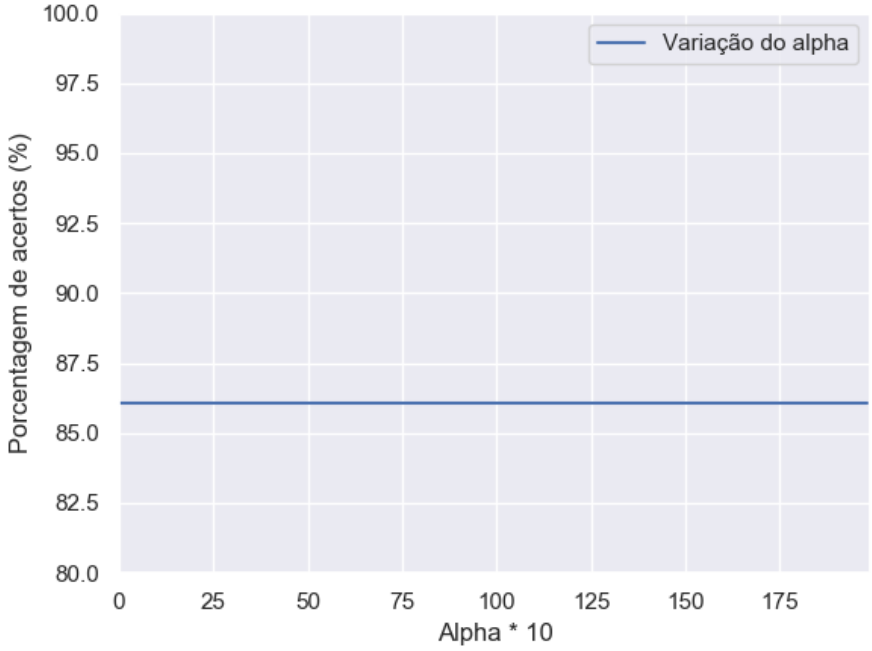

Figura 5 – Variação da acurácia do modelo *Naive Bayes* pela variação do *alpha* para a base de dados Hill.

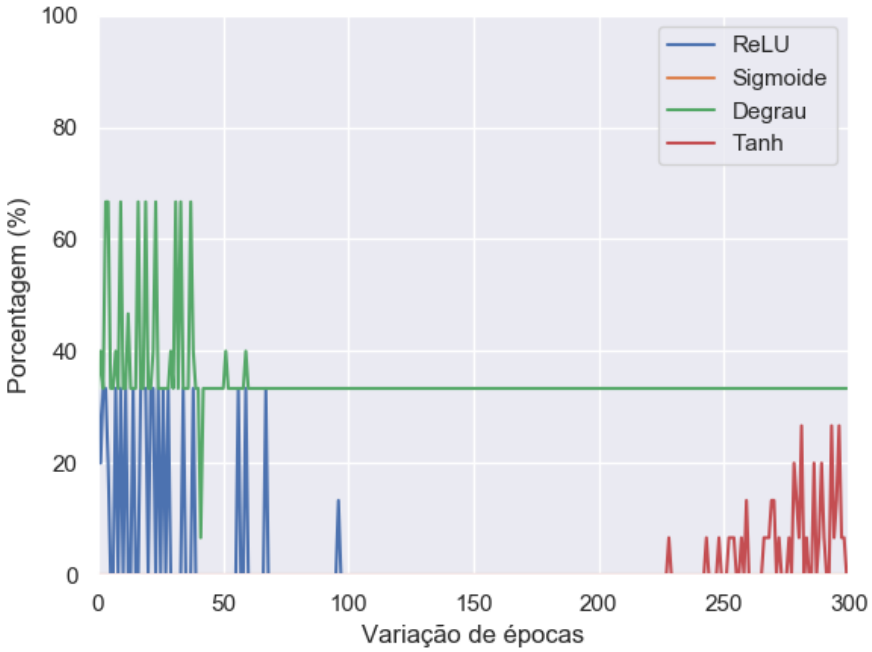

Figura 6 – Variação da acurácia do modelo Rede Neural pela variação da quantidade de épocas de treino para a base de dados Íris.

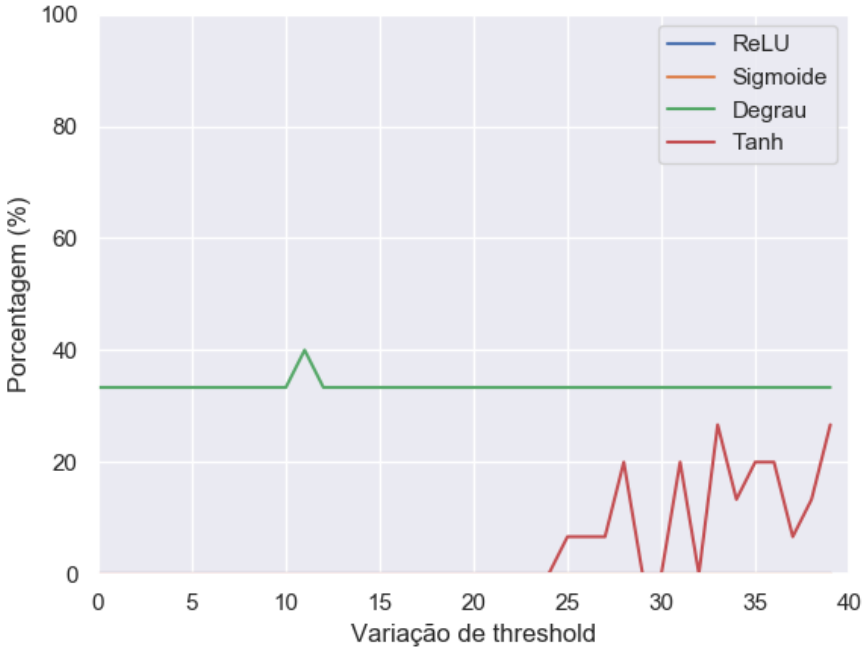

Figura 7 – Variação da acurácia do modelo Rede Neural pela variação do *threshold* de treino para a base de dados Hill.

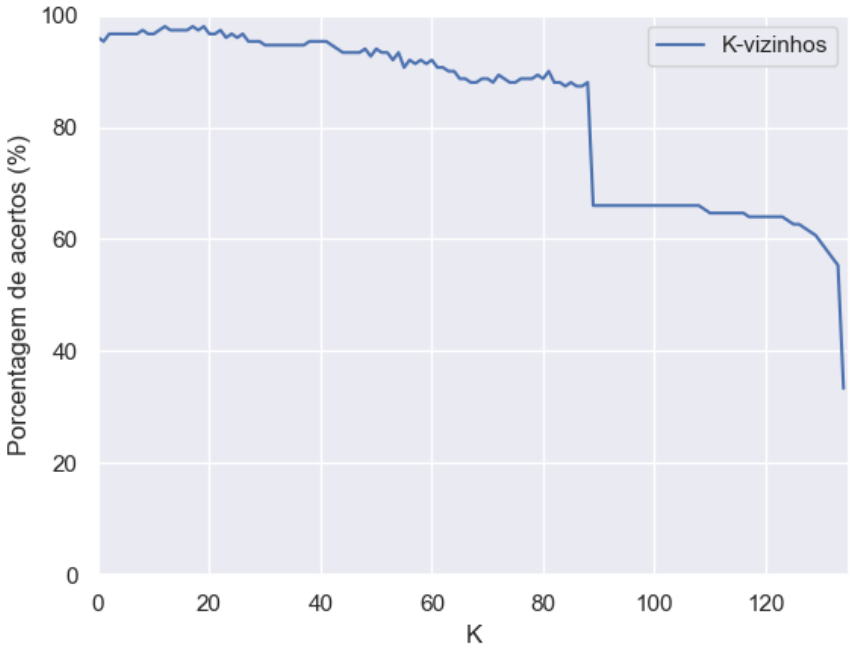

Figura 8 – Variação da acurácia do modelo KNN pela variação dos K-vizinhos para a base de dados Íris.

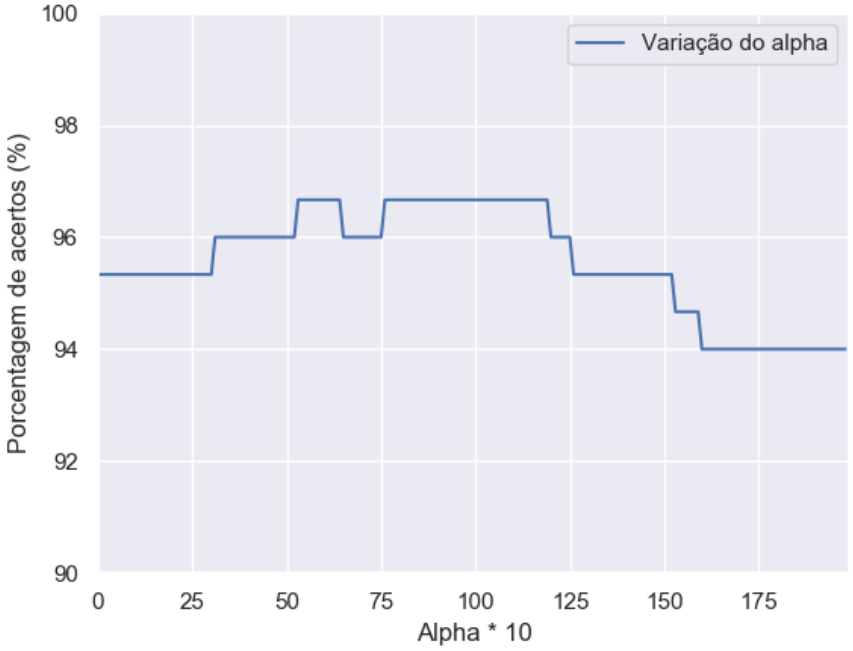

Figura 9 – Variação da acurácia do modelo *Naive Bayes* pela variação do *alpha* para a base de dados Íris.

Função de Ativação	**Acurácia (%)**
Degrau	74.0204
Sigmoid	73.8954
ReLU	50.9947
TanH	49.0052

Tabela 1 – Acurácias utilizando os melhores resultados de *Threshold* e épocas e a metodologia de validação cruzada para o classificador *Perceptron* na base de dados Hill.

Classificador	**Acurácia (%)**
Naive Bayes	86.0601
KNN	64.5190

Tabela 2 – Acurácias utilizando os melhores resultados dos experimentos a metodologia de validação cruzada na base de dados Hill.

Função de Ativação	**Acurácia (%)**
Degrau	33.3333
Sigmoid	0.0
ReLU	33.3333
TanH	22.6666

Tabela 3 – Acurácias utilizando os melhores resultados de *Threshold* e épocas e a metodologia de validação cruzada para o classificador *Perceptron* na base de dados Íris.

Classificador	**Acurácia (%)**
Naive Bayes	86.0601
KNN	98.0001

Tabela 4 – Acurácias utilizando os melhores resultados dos experimentos e a metodologia de validação cruzada na base de dados Íris.

Classificador	Tempo de execução (s)
RNA + Degrau	26.1658
RNA + Sigmoid	125.7785
RNA+ ReLU	78.5546
RNA + TanH	73.7237
KNN	0.0610
Naive Bayes	0.0148

Tabela 5 – Tempos de execução de cada classificador para a base de dados Hill.

Classificador	Tempo de execução (s)
RNA + Degrau	0.0535
RNA + Sigmoid	0.1923
RNA+ ReLU	0.2156
RNA + TanH	3.4318
KNN	0.0091
Naive Bayes	0.0075

Tabela 6 – Tempos de execução de cada classificador para a base de dados Íris.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Analisando o a Figura 2, podemos observar o crescimento da acurácia da classificação a partir de mais ou menos 25 (vinte e cinco) épocas de treino, com alguns pontos de queda na acurácia perto de 250 (duzentos e cinquenta) épocas. O maior valor alcançado a partir de testes diretos foi de 98.35%, alcançado pelas funções Sigmoid e Degrau, como observado na mesma. Ainda é possível observar que a função ReLU e TanH tem valores iniciais altos, por volta dos 50%, mas que tais valores não variam com o aumento das épocas, como nas outras funções.

Na Figura 3, podemos observar mais ou menos o mesmo comportamento da Figura 2, onde as funções Sigmoid e Degrau tem valores altos na acurácia por testes diretos e as funções Tanh e ReLU, apesar de já iniciarem com valores razoáveis, não apresentam melhorias durante a variação do *threshold*. Os maiores valores alcançados a partir de testes diretos foram 95.06% para a função Degrau, 94.23% para a função Sigmoid, 48.55% para a função TanH e 51.44% para a função ReLU.

A partir desses valores, foram feitos então novos testes na rede neural em cada função, dessa vez utilizando a metodologia de validação cruzada com 10 (dez) camadas. Tais resultados estão dispostos na Tabela 1 e nota-se que a função que deu melhores resultados foi a Degrau.

Os mesmos experimentos foram feitos utilizando as mesmas regras e os resultados dispostos nas Figuras 6 e 7 mostram comportamentos extremamente diferentes da base de dados Hill.

A rede neural implementada apresentou um comportamento medíocre quando sub-

metida à base de dados Íris, deixando muito a desejar na sua acurácia. Além de sua acurácia baixa, os tempos de execução da rede neural, em comparação com os outros algoritmos, foi muito elevado. Tais elevações no tempo de execução devem-se pela própria natureza do algoritmo, além da forma de implementação. Quando comparadas, as implementações da biblioteca de apoio e as próprias, as implementações próprias perdem e muito.

Quanto a baixa acurácia, espera-se que tal comportamento foi demonstrado por causa da convergência dos gradientes da rede para 0, fazendo com que a rede aprenda de forma enviesada, gerando falso-positivos. Tais problemas podem ser contornados utilizando técnicas de *Back-Propagation*, presente em redes neurais mais complexas.

Observando as Figuras 6 e 7, é possível observar comportamentos não similares às figuras anteriores, pois a única função que obteve ganho com o crescer da quantidade de épocas foi a função TanH. As demais funções se mantiveram estáveis ao crescer das épocas, e a função sigmoid e ReLU apresentaram 0.0% de acurácia sobre todo o experimento.

Na Figura 7 é possível observar um comportamento interessante na função TanH, novamente. A partir da 23^a variação do *threshold*, houve um crescimento acentuado na acurácia do modelo, sugerindo que valores acima disso tenham maiores acurácias e levando a conclusão de que a mesma necessita de uma quantidade alta de épocas de treinamento.

Partindo para os outros classificadores, as Figuras 4 e 8 mostram a variação da acurácia do classificador KNN a partir da variação dos K-vizinhos, parâmetro do classificador. Pode-se observar na Figura 4 que a acurácia é localmente sensível mas que ela se mantem em uma média de 53% até perto de 400 (quatrocentos), onde apresenta uma queda mais considerável na acurácia, e a mesma se mantem por ali. O maior valor foi alcançado com 1 (um) vizinho, alcançando 64.00% de acurácia. A baixa acurácia do modelo é esperada. As amostras da base de dados, por se tratarem de curvas em um plano cartesiano, são muito parecidas, ou seja, a distância entre uma instância e outra é consideravelmente baixa, deixando o KNN em desvantagem quando comparado aos outros.

Já na Figura 8, observamos um comportamento bem parecido, onde o crescimento da quantidade de vizinhos deixa a acurácia do classificador desfalcada. Isso é bem notável a partir de 80 (oitenta) vizinhos, quando a acurácia sofre quedas bruscas. O melhor ponto foi alcançado com 12 (doze) vizinhos e acurácia de 98.00%.

Finalmente, o *Naive Bayes* apresentou baixas variações de acurácia quanto a variação do *alpha*. É observável na Figura 5 que a acurácia do modelo se manteve constante mesmo variando o *alpha* bastante. Já na Figura 9, houve leves variações, alcançando até um valor máximo de 96.80% nos testes diretos. A acurácia na base de dados Íris sofreu uma queda brusca a partir da 10^a variação, sugerindo que o modelo tenha piores resultados a partir disso.

A partir dessas afirmações, é possível chegar ao consenso de que o *Naive Bayes* apresentou melhores resultados gerais na base de dados Hill e o KNN apresentou melhores

resultados na base de dados Íris. Infelizmente, a rede neural não atendeu às expectativas, comparada aos outros modelos. A partir disso, também é possível afirmar que as Redes Neurais são úteis, mas somente se forem mais trabalhadas, com algoritmos de correção de erros e otimizações consideráveis.

5 CONCLUSÃO

Podemos dizer que os objetivos foram alcançados com sucesso. Acompanhamos os comportamentos dos algoritmos de classificação em ambas bases de dados, analisamos criticamente suas acurácias, ou a falta delas, em alguns casos.

Em relação aos resultados, observando a análise, pode-se concluir que a rede neural *Perceptron*, apesar de ser um classificador renomado, é bastante limitado e só funciona bem para casos específicos. Já os outros classificadores tem um maior poder de generalização, apresentando bons resultados até quando submetidos em bases de dados multi-classes.

Por fim, com base nas comparações feitas entre os classificadores, podemos concluir que, em geral, o classificador *Naive Bayes* apresentou melhores resultados do que seus concorrentes, tanto em acurácia quanto em tempo de execução, ou seja, apresenta um melhor custo benefício.

REFERÊNCIAS

ACADEMY, D. S. **Deep Learning Book.** 2018. <http://deeplearningbook.com.br/funcao-de-ativacao/>. Accessed: 2018-10-23.

BRAGA, A. d. P.; CARVALHO, A.; LUDERMIR, T. B. **Redes neurais artificiais: teoria e aplicações**. [S.l.]: Livros Técnicos e Científicos Rio de Janeiro, 2000.

LEARNING, C. for M.; SYSTEMS, U. o. C. I. **UCI Machine Learning Repository**. 2018. <http://archive.ics.uci.edu/ml/index.php>. Accessed: 2018-09-15.

MATPLOTLIB. **Matplotlib, a package for data visualization.** 2018. <https://matplotlib.org/>. Accessed: 2018-10-23.

NUMPY. **Numpy - a package for scientific computing**. 2018. <http://www.numpy.org/>. Accessed: 2018-10-23.

PRASS, F. S. Kdd–uma visão geral do processo. **Recuperado em**, v. 15, 2016.

PYTHON. **Time - Time access and conversions**. 2018. <https://docs.python.org/3/library/time.html>. Accessed: 2018-10-23.

SAS. **Machine Learning o que é e qual sua importância SAS**. 2018. <https://www.sas.com/pt_br/insights/analytics/machine-learning.html>. Acessed: 2018-10-30.

SKLEARN. **Scikit-Learning, a package for machine learning in Python**. 2018. <http://scikit-learn.org/stable/>. Accessed: 2018-10-23.

TAN, P.-N. et al. **Introduction to data mining**. [S.l.]: Pearson Education India, 2007.

ZAKI, M. J.; JR, W. M.; MEIRA, W. **Data mining and analysis: fundamental concepts and algorithms**. [S.l.]: Cambridge University Press, 2014.

ANALYSIS OF CLASSIFICATION METHODS FOR DATA MINING TASKS

ABSTRACT

The process of extracting knowledge from databases is a rather complex process and is a growing area of study of current computing. The process is known as KDD (Knowledge Discovery database) and is composed of several phases, the whole being the procedure of knowledge extraction. The phase that will be approached in this article is the mining of the data, more precisely, the classification of the same. Three different classifiers will be addressed, such as an Artificial Neural Network, KNN and Naive Bayes. They will be submitted to two databases and to different parameters, seeking different levels of precision.

Keywords: *Data Mining. Classification. Neural Network. Naive Bayes. KNN.*