INSTITUTO FEDERAL DE MINAS GERAIS - CÂMPUS BAMBUÍ
DEPARTAMENTO DE ENGENHARIAS E COMPUTAÇÃO
Bacharelado em Engenharia de Computação

HARLEY VITOR SANTANA DE BRITO
HEITOR RAMOS ALVES DA SILVA
MARCO ANTÔNIO GOMES
SAMUEL TRINDADE SILVA

CLASSIFICAÇÃO DE COMENTÁRIOS DO IMDB USANDO ALGORITMOS DE MACHINE LEARNING

BAMBUÍ
2017

INTRODUÇÃO

Para o trabalho final da disciplina de Inteligência Artificial, foi proposto que os alunos implementassem 4 algoritmos de machine learning, supervisionados. Destes algoritmos, dois deveriam ser implementados do zero e dois implementados usando bibliotecas.

Após a implementação dos mesmos foi proposta a validação cruzada destes algoritmos para mostrar sua acurácia de previsão, para modelos desconhecidos.

Com isto escolhemos implementar, a rede neural Adaline, o KNN, e para as bibliotecas foram utilizadas o Naive Bayes, e o KNN.

1 NAIVE BAYES

Naive Bayes é um algoritmo que é utilizado no campo de aprendizagem de máquina para problemas de classificação. É principalmente empregado para classificação de texto, que requer uma grande quantidade de amostras para treinamento. Pode ser utilizado para filtro de spam, análise de sentimento etc... O algoritmo recebe o nome "naive" pois sua premissa assume que a frequência das características de uma população independe da frequência de quaisquer outras características. O algoritmo baseia-se no teorema de Bayes, desenvolvido por Thomas Bayes, um estatístico e filósofo.

O teorema afirma que, a probabilidade de um determinado evento B acontecer sob uma determinada condição A é igual à probabilidade do evento A acontecer sob a condição de B multiplicada pela divisão da probabilidade de A pela probabilidade de B, isso é:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Para classificação de texto, para uma dada amostra y e um conjunto de características x_n, o algoritmo de Naive Bayes calcula:

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

O algoritmo possui três variações:

- Naive Bayes Multinomial;
- Naive Bayes de Bernoulli;
- Naive Bayes Gaussiano.

No trabalho foi utilizado o algoritmo Multinomial da biblioteca scikit-learn. Considerando um conjunto de probabilidades $P(x_i|y) = \theta_y = (\theta_{y1}, \dots, \theta_{yn})$ para cada amostra y e uma característica i, com $n = número\ de\ características$:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Onde $N_{yi} = \sum x \in T^{x_i}$, é a quantidade de vezes que uma característica i ocorre em uma amostra y em um conjunto de treinamento T

E $N_y = \sum_{i=1}^{|T|} N_{yi}$, é a contagem total de todas as características para uma amostra y.

Para a implementação do trabalho foi utilizada uma base de dados contendo 25000 comentários do imdb para o conjunto de treinamento do algoritmo. Foram utilizadas também as bibliotecas $sklearn.datasets$, $sklearn.feature\_extraction.text$, $sklearn.naive\_bayes$ e $sklearn.model\_selection$.

Inicialmente os 25000 comentários são carregados com suas respectivas classes:

```
categorias = ["positivo", "negativo"]
dados = load_files("H:\\downloads\\ComentariosIMDB", description = None,categories =
categorias,
load_content = True, shuffle = False, encoding="utf-8", decode_error = 'strict',
random_state = 0)
```

Em seguida, foi instanciado um objeto da classe CountVectorizer(), cujos atributos são um dicionário contendo as palavras presentes nos comentários e palavras que foram ignoradas devido à ocorrência máxima ou mínima atingida. Foi instanciado também um objeto da classe TfidfTransformer(). Em seguida utilizamos o método fit_transform(), do objeto CountVectorizer instanciado, para armazenar a frequência de ocorrência de cada palavra contida nos comentários em cada comentário e o método fit_transform(), do objeto TfidfTransformer(), para inverter a importância da ocorrência de palavras muito constantes:

```
vetorizador = CountVectorizer()
vetorizador_invertido = TfidfTransformer()
dados_vetorizados = vetorizador.fit_transform(dados.data)
dados_invertidos = vetorizador_invertido.fit_transform(dados_vetorizados)
```

Com os dados quase prontos para serem utilizados, armazenamos uma instância da classe train_test_split nas variáveis dados_treinamento, dados_teste, categorias_treinamento e categorias_teste para posterior uso no treinamento e na validação cruzada dos dados.

```
dados_treinamento, dados_teste, categorias_treinamento, categorias_teste =
train_test_split(dados_invertidos, dados.target, test_size=0.1, random_state=0)
```

O parâmetro test_size define a porcentagem da população que será utilizada para validação cruzada, utilizamos 10% dos comentários para validação cruzada. Em seguida utilizamos o método fit() de uma instância de classe MultinomialNB para treinar o estimador. Em seguida, foi chamado o método predict() da instância de MultinomialNB para realizar a previsão dos 10% dos comentários reservados, e, por último, imprimimos a chamada de método score() da instância MultinomialNB para ver o resultado da validação cruzada, que no nosso caso, foi de 93,44%.

```
naive_bayes = MultinomialNB().fit(dados_invertidos, dados.target)
previsao = naive_bayes.predict(dados_teste)
print(naive_bayes.score(dados_teste, categorias_teste))
```

2 KNN

2.1 O que é o KNN

O algoritmo KNN foi proposto por Fukunaga e Narendra em 1975. É um dos classificadores mais simples de ser implementado, de fácil compreensão e ainda hoje pode obter bons resultados dependendo de sua aplicação.

2.2 Como o KNN funciona

A ideia principal do KNN é determinar o rótulo de classificação de uma amostra baseado nas amostras vizinhas advindas de um conjunto de treinamento. O algoritmo KNN pertence à família de algoritmos baseados em instâncias, competitivos e de aprendizado preguiçoso.

Algoritmos baseados em instâncias são os algoritmos que modelam o problema usando instâncias de dados (ou linhas) para tomar decisões preditivas. O algoritmo KNN é uma forma extrema de métodos baseados em instâncias porque todas as observações de treinamento são mantidas como parte do modelo.

É um algoritmo de aprendizagem competitivo, porque utiliza internamente a competição entre elementos do modelo (instâncias de dados) para tomar uma decisão preditiva. A medida de similaridade objetiva entre instâncias de dados faz com que cada instância de dados compita para "ganhar" ou ser mais semelhante a uma determinada instância de dados não vistos e contribuir para uma previsão.

O aprendizado preguiçoso refere-se ao fato de que o algoritmo não constrói um modelo até o momento em que é necessária uma previsão. É preguiçoso porque só funciona no último momento de sua execução. Isso tem o benefício de incluir apenas dados relevantes para os dados não vistos, chamados de modelo localizado, uma desvantagem é que pode ser computacionalmente caro repetir a mesma pesquisa ou pesquisas semelhantes em conjuntos de dados de treinamento maiores.

Finalmente, o KNN é poderoso porque não assume nada sobre os dados, além de uma medida de distância, pode ser calculado consistentemente entre duas instâncias. Como tal, é chamado de não-paramétrico ou não linear, pois não assume uma forma funcional.

2.3 Aplicação do KNN usando biblioteca

Em nosso trabalho, aplicamos o algoritmo KNN com o uso da biblioteca Scikit-learn, que é focada no aprendizado de máquina, de código aberto disponível para a linguagem python. Foi feito todas as importações das bibliotecas internas que seriam necessárias para o funcionamento do código.

2.4 Construindo o KNN

Para construir o KNN, foi utilizado apenas uma dimensão, para o problema, no caso a distância a k que este deveria procurar, seria referente as frequências de treino e teste, onde este seria o novo comentário.

Com isso foi pensado em uma função de predição que receberia 2 matrizes esparsas] uma de treino e uma de teste, o número de vizinho, os vetores de frequência, e o comentário.

Então o algoritmo pegava os comentários, o quebrava em várias strings, e procurava no vetor de frequência de treino para saber qual o índice deveria procurar na matriz esparsa. Posteriormente deveria pegar os valores dentro dos índices, gerar as distâncias euclidianas e devolver os menores k, classificando o comentário.

Contudo o vetor das palavras do treino era muito grande, sendo assim o cálculo fica sendo (palavras ^ palavras do vetor treino) o que tornou nosso algoritmo inviável, visto que para 400 comentários tínhamos em torno de 10800 palavras no vetor, e um comentário pequeno, mesmo trabalhado para diminuir suas strings tinha em torno de 100 caracteres, gerando assim um cálculo muito dispendioso, o qual não nos permitiu conseguir terminar o algoritmo, pois não conseguimos compilar em tempo hábil.

Uma observação foi que em uma hora e meia de compilação, conseguimos ler em torno de 10 palavras para um vetor com 10800 palavras no vetor de frequência.

2.5 Resultados utilizando bibliotecas

A implementação do algoritmo foi bem clara e objetiva, pois a biblioteca é responsável por todo o tratamento dos dados. Primeiramente foi feita a leitura de todos os comentários da base de dados para a memória, para que ficassem disponíveis para o devido tratamento. Em seguida, treinamos os dados através da função train_test_split, passando como parâmetro os dados dos comentários já lidos, as categorias em que eles se encaixam (positivos ou negativos) e a quantidade de comentários para se fazer o teste no algoritmo. Com o treinamento realizado, passamos alguns comentários do próprio banco para fazer a predição, e contabilizamos a porcentagens de acerto através da função score, que retorna à probabilidade de acerto dos comentários testados.

O algoritmo se mostrou bastante eficiente, com uma média de acertos de 67% com a constante K=5 da quantidade de vizinhos a ser analisado a partir de um certo comentário. O impacto desta constante foi médio no final da execução, uma variação no valor de K de 2 até 12, apresentava uma variação de aproximadamente 6% no acerto total.

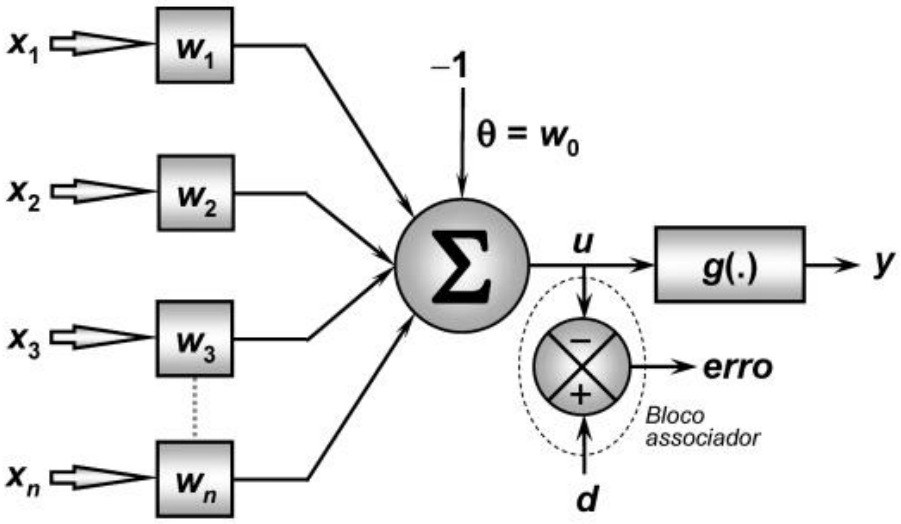
3 REDE NEURAL ADALINE

3.1 Adaline e estrutura

Para o problema de classificação de comentários do IMDB escolheu se o algoritmo de Rede Neural, Adaline, idealizado por Widrow e Hoff em 1960, o algoritmo foi primeiramente utilizado para chaveamento de circuitos telefônicos e em suas primeiras aplicações industriais.

A estrutura da Adaline é parecida com a configuração da Perceptron, constituído de apenas uma camada neural e por um único neurônio artificial. De forma geral a estrutura da Adaline é representada conforme a figura 1 a seguir:

Figura 1 - Estrutura da RNA Adaline



Fonte: Redes Neurais Artificiais Para Engenharia e Ciências Aplicada: Curso Prático, Pag. 74

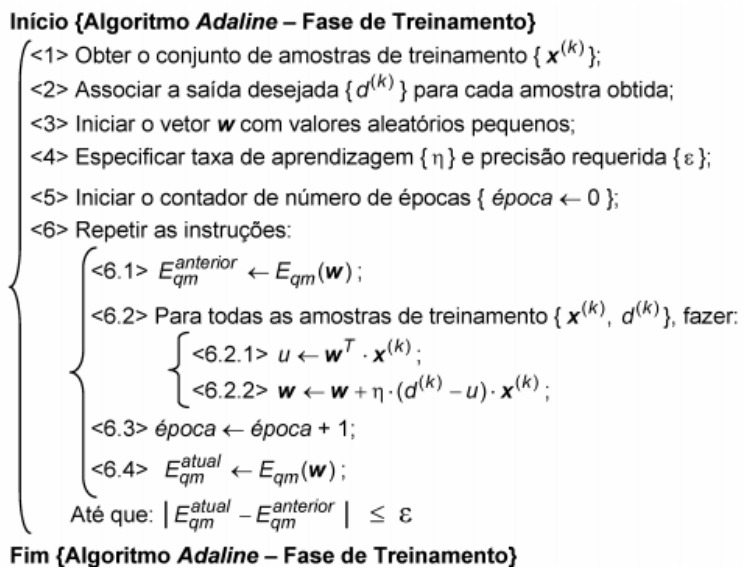
A escolha de implementar o algoritmo do Adaline se deu devido a característica do problema, que é uma classificação de padrões, a simplicidade do modelo é ideal para tal problema, visto ainda que é um problema que envolve somente duas classes de separação diferentes, segundo Silva, Spatti e Flauzino (2010).

O potencial de ativação da rede vista na figura 1 é calculado baseado no respectivo somatório dos valores de entrada multiplicados pelos seus pesos incluindo também o limiar de ativação. A saída produzida vem da utilização da função de ativação g(u) que no caso do problema será utilizado uma função degrau bipolar.

O bloco associador da figura 1 visa auxiliar no processo de treinamento da rede através do cálculo do erro no qual se subtrai o valor desejado o limiar de ativação e com isso os pesos são reajustados.

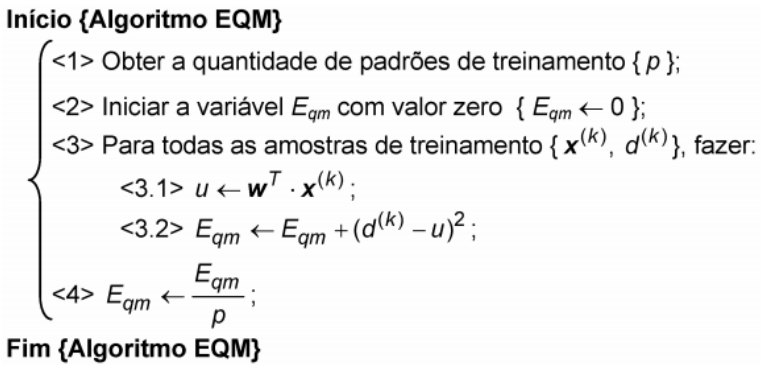
A vantagem do Adaline em relação a Perceptron é a utilização do algoritmo de aprendizado chamado de *Regra Delta*, conhecido também como método do Gradiente Descendente. A figura 2,3 e 4 mostram os pseudocódigos utilizados para a implementação dos algoritmos.

Figura 2 – Pseudocódigo Algoritmo Adaline de Treinamento



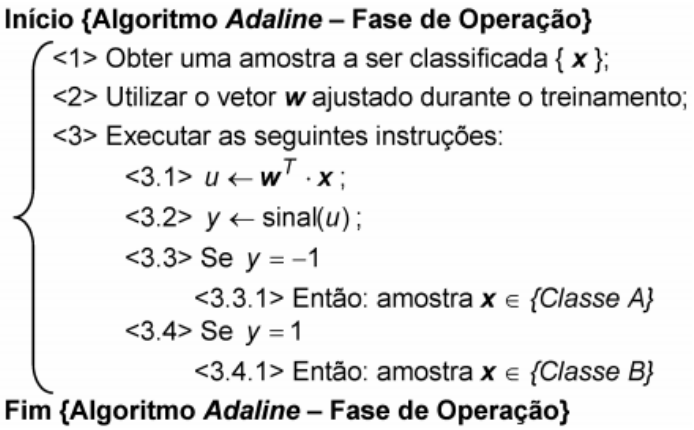
Fonte: Redes Neurais Artificiais Para Engenharia e Ciências Aplicada: Curso Prático, Pag. 81

Figura 3 – Algoritmo Adaline de cálculo do erro quadrático médio



Fonte: Redes Neurais Artificiais Para Engenharia e Ciências Aplicada: Curso Prático, Pag. 81

Figura 4 – Algoritmo Adaline classificação das amostras



Fonte: Redes Neurais Artificiais Para Engenharia e Ciências Aplicada: Curso Prático, Pag. 82

O código do Adaline foi implementado em python conforme se segue abaixo:

```
1.  from random import  uniform
2.  import random
3.
4.
5.
6.  def degreeBipole(u):
7.      if( u >= 0):
8.          return 1
9.      return -1
10.
11.
12.
13.
14. def calcActivationThreshold(element):
15.     u = 0
16.     for j in range(len(element)):
17.         u =  u + W[j]*element[j]
18.     return u
19.
20.
21. def calcDelta(error,element):
22.     delta = list()
23.     for k in range(len(W)):
24.         delta.append(error * eta * element[k])
25.     return  delta
26.
27.
28. def updateWeights(delta):
29.     for j in range(len(delta)):
30.         W[j] = W[j] + delta[j]
31.
32.
33.
34. def setWeigths():
```

```
    for i in range(3):
        W.append(uniform(-0.3,0.3))


def extractElement(index, data):
    element = []
    for m in data:
      element.append(m[index])
    return element


def trainingAndLearning(eseason,maxseason,tol,averageErrorSeasonToSeason,N,data,yEsperado):
    nseason = 0  # Numero de Épocas
    index = list(range(N))
    random.shuffle(index)

    while (( nseason < maxseason) and (eseason > tol)):

        convergenceError = 0

        for i in range(N):

            currentIndex = index[i]
            element = extractElement(currentIndex,data)
            u  = calcActivationThreshold(element)
            error = yEsperado[currentIndex]-u
            delta = calcDelta(error,element)
            updateWeights(delta)
            convergenceError += (error**2)
            #print(W)

        averageErrorSeasonToSeason.append((convergenceError/N))
        eseason = averageErrorSeasonToSeason[nseason]
        nseason += 1



def classification (element):
    u = calcActivationThreshold(element)
    yhat = degreeBipole(u)
    return yhat
```

3.2 Treinamento e tratamento dos dados

Para realizar o treinamento, primeiramente realizou se o tratamento dos dados para a utilização na rede, sendo que nesta parte, adotou se a convenção de que para todos os experimentos, os comentários que estavam em um pasta com 2 diretórios, sendo um com 12 500 comentários negativos e o outro 12 500 comentários positivos, ficariam separados em 5 intervalos sendo, que cada intervalo consistiu em 5 pastas, cada uma com 2 diretórios contendo em um 2500 comentários negativos e na outra 2500 comentários positivos.

No tratamento de dados para uso no algoritmo de treinamento da RNA e nos outros algoritmos utilizou se da biblioteca do sklearn para tratamento e leitura de datasets no python, no algoritmo da rede. Dos 5 intervalos, primeiramente carregou se os comentários do 1º intervalo e depois eles foram separados em arrays de comentários positivos e negativos. Cada posição de um dos arrays, positivo ou negativo era um comentário lido dos arquivos nos diretórios, logo após, juntou se todos os comentários, negativo ou positivo, em duas strings, uma de comentários negativos e outra de comentários positivos e com a função split do python separou se cada palavra presente na string.

Com a utilização da biblioteca Counter, realizou se a contagem da frequência de cada palavra presente, tanto na string de comentários positivos, como na string de comentários negativos, para isso criou se dois objetos do tipo Counter para que se realizasse a contagem. Depois disso, foi preciso fazer um tratamento para a separação no conjunto de treinamento, de palavras que não influenciaram no treinamento, tais palavras são, artigos, substantivos e alguns adjetivos e caracteres que são como outliers para o modelo, podendo gerar underfitting e overfitting no momento de classificação.

O modo de entrada dos dados para treinamento na rede, são duas entradas de dados, uma para a frequência de comentários positivos e outras para a frequência de comentários negativos, para que a RNA aprendesse utilizou se dá heurística criada no qual se associava na saída o valor -1 como esperado para quando a frequência da palavra negativa fosse maior que a frequência da palavra positiva e o valor 1 para o contrário. Sendo assim espera se que RNA conseguisse classificar um comentário em negativo ou positivo baseado na frequência das palavras.

Para efetuar o treinamento o algoritmo utiliza a frequência das palavras dos comentários negativos e positivos padronizados num intervalo entre 0 e 1, um detalhe importante é o de que se utilizou como parâmetro a retirada de 10 000 palavras das strings de cada comentário, tal valor foi escolhido empiricamente pois na análise das strings percebeu-se que havia muitas palavras que não se repetiam ou que se repetiam apenas uma vez, por isso não compensava pegar todas palavras para se fazer o treinamento, depois do treino, realizou a se classificação do outros comentários carregando os outros intervalos, 2º,3º,4º e 5º, e realizando a classificação.

3.3 Classificação dos comentários

Para a classificação, o algoritmo trata os comentários de uma forma diferente, os comentários são lidos um a um do array de comentários de classificação que é carregado separadamente. A leitura ocorre em um laço do tamanho da quantidade de comentários presentes no intervalo, o comentário é splitado e dele são extraídos as palavras e frequências para serem passadas para o algoritmo de classificação do Adaline. Ao retirar as palavras, realiza se aqui também, o tratamento para retirada das palavras insignificantes para a classificação.

A biblioteca do sklearn possui a característica de ao carregar os dados criar um array chamado de target, este é um array binário no qual cada posição dele afirma se o comentário é negativo (0) ou positivo (1), com o uso do target é possível verificar se a RNA realizou a classificação corretamente.

Ao realizar classificação o algoritmo verifica se a mesma foi correta e incrementa uma variável que diz respeito a taxa de acerto do modelo implementado, no final essa variável é dividida pela quantidade total dos comentários de classificação para se saber o percentual de acerto da RNA implementada.

O script para tratamento e classificação dos dados utilizando python foi implementado como se segue abaixo:

```
1.  from sklearn.datasets import *
2.  from collections import Counter
3.  import time
4.  from Adaline import*
5.
6.
7.  def padroniza(frequencia,frequenciaPadronizada):
8.      for i in range(len(frequencia)):
9.          frequenciaPadronizada.append((frequencia[i]-
    min(frequencia))/(max(frequencia)-min(frequencia)))
10.
11. def padronizaClassiflicacao(frequencia, frequeciaRef):
12.     elementosPadronizados = []
13.     for i in range(len(frequencia)):
14.         elementosPadronizados.append((frequencia[i]-
    min(frequeciaRef))/(max(frequeciaRef)-min(frequeciaRef)))
15.     return elementosPadronizados
16.
17. if __name__ == "__main__":
18.
19.     print("Lendo dados de classificação...")
20.     categorias = ["positivo", "negativo"]
21.
22.
23.
24.     dados = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteligencia Artific
    ial\\Divisão Comentarios\\1º Intervalo", description = None, categories = categoria
    s, load_content = True, shuffle = False, encoding="utf-
    8", decode_error = 'strict', random_state = 0)
25.
26.     #dados = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteligencia Artifi
    cial\\Divisão Comentarios\\2º Intervalo",description=None, categories=categorias, l
    oad_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
27.
28.     #dados = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteligencia Artifi
    cial\\Divisão Comentarios\\3º Intervalo",description=None, categories=categorias, l
    oad_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
29.
30.     #dados = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteligencia Artifi
    cial\\Divisão Comentarios\\4º Intervalo",description=None, categories=categorias, l
    oad_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
31.
32.     #dados = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteligencia Artifi
    cial\\Divisão Comentarios\\5º Intervalo",description=None, categories=categorias, l
    oad_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
33.
```

```
34.     #dadosClassificacao = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteli
    gencia Artificial\\Divisão Comentarios\\1º Intervalo",description=None, categories=
    categorias, load_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
35.
36.     #dadosClassificacao = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteli
    gencia Artificial\\Divisão Comentarios\\2º Intervalo",description=None, categories=
    categorias, load_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
37.
38.     #dadosClassificacao = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteli
    gencia Artificial\\Divisão Comentarios\\3º Intervalo",description=None, categories=
    categorias, load_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
39.
40.     #dadosClassificacao = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Inteli
    gencia Artificial\\Divisão Comentarios\\4º Intervalo",description=None, categories=
    categorias, load_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
41.
42.     dadosClassificacao = load_files("C:\\Users\\Marco\\Desktop\\6º Periodo\\Intelig
    encia Artificial\\Divisão Comentarios\\5º Intervalo",description=None, categories=c
    ategorias, load_content=True, shuffle=False, encoding="utf-
    8", decode_error='strict', random_state=0)
43.
44.
45.     print("Ajustando os dados de classificação... ")
46.     time.sleep(100)
47.
48.     comentarios = dados.data
49.     #print(comentarios[1])
50.     #print(dados.target)
51.
52.     comentariosClassificacao = dadosClassificacao.data
53.     tamanho = len(comentarios)
54.     tamanhoClassificacao = len(comentariosClassificacao)
55.     QtdPalavras = 10000
56.
57.     palavrasIgnoradas = ['herself.<br','/>If','saw','"Duchess"','sound','So','there
    ','into','/','the','and','of','a','to','is','in','that','/><br','I','this','as','it
    ', 'for','with','was','but','The','film','by','on','are','his','her','movie','not',
    'you','have','be','at','from','who','an','one','he','all','she','has','or','very','
    their','they','so','when','some','more','-','were','what','its','my','/>The',]
58.
59.     comentariosNegativos = []
60.     comentariosPositivos = []
61.
62.     palavrasPositivasTreino = []
63.     palavrasNegativasTreino = []
64.
65.     freqPositivasTreino = []
66.     freqNegativasTreino = []
67.
68.     frequenciaPositivaPadronizada = []
69.     frequenciaNegativaPadronizada = []
70.
71.     stringPositivos = ""
72.     stringNegativos = ""
73.     stringComentariosClassificacao = ""
74.     palavrasClassificacao = []
75.     freqPalavrasClassificacao = []
76.     elementoClassificacao = []
77.     taxaDeAcerto = 0
78.     y = []
79.
80.     for i in range(int((tamanho/2)-1)):
```

```
        comentariosPositivos.append(comentarios[i])


    for i in range((int(tamanho/2)),(tamanho)):
        comentariosNegativos.append(comentarios[i])









    for i in range ((int(tamanho/2)-1)):
        stringPositivos = comentariosPositivos[i] +  stringPositivos
        stringNegativos = comentariosNegativos[i] + stringNegativos


    stringPositivos = stringPositivos.split(' ')
           stringNegativos = stringNegativos.split(' ')









           FreqNegativos = Counter(stringNegativos)
           FreqPositivos = Counter(stringPositivos)

           #print(FreqNegativos)
           #print(FreqPositivos)

           palavrasNegativas = FreqNegativos.most_common(QtdPalavras)
           palavrasPositivas = FreqPositivos.most_common(QtdPalavras)



           for i in range(QtdPalavras):
               #print(palavrasNegativas[i][0])
               if palavrasNegativas[i][0] not in palavrasIgnoradas :
                   palavrasNegativasTreino.append(palavrasNegativas[i][0])
                   freqNegativasTreino.append(palavrasNegativas[i][1])

               if palavrasPositivas[i][0] not in palavrasIgnoradas:
                   palavrasPositivasTreino.append(palavrasPositivas[i][0])
                   freqPositivasTreino.append(palavrasPositivas[i][1])


           #print( palavrasNegativasTreino)
           #print(freqNegativasTreino)
           #print( palavrasPositivasTreino)
           #print(freqPositivasTreino)

           for i in range(len(freqNegativasTreino)):

               if freqNegativasTreino[i] > freqPositivasTreino[i]:
                   y.append(-1)
               else:
                   y.append(1)


           padroniza(freqNegativasTreino, frequenciaNegativaPadronizada)
           padroniza(freqPositivasTreino, frequenciaPositivaPadronizada)

           #print(freqPositivasTreino)

```

```
146.           matriz = [frequenciaPositivaPadronizada, frequenciaNegativaPadronizada,
    y]
147.
148.
149.           print("Treinando a Rede Neural Adaline ... ")
150.           time.sleep(100)
151.
152.           data = matriz
153.           yEsperado = y
154.           N =  len(freqPositivasTreino)
155.
156.           setWeigths()
157.           trainingAndLearning(eseason,maxseason,tol,averageErrorSeasonToSeason,N,d
    ata,yEsperado)
158.
159.           print("Classiflicando Comentários ... ")
160.           time.sleep(100)
161.           for i in range(tamanhoClassificacao):
162.               stringComentariosClassificacao = comentariosClassificacao[i]
163.               stringComentariosClassificacao = stringComentariosClassificacao.spli
    t(' ')
164.               FreqPalavrasDosComentariosClassificacao = Counter(stringComentariosC
    lassificacao)
165.               FreqPalavrasDosComentariosClassificacao = FreqPalavrasDosComentarios
    Classificacao.most_common(len(FreqPalavrasDosComentariosClassificacao))
166.
167.
168.
169.               for j in range(len(FreqPalavrasDosComentariosClassificacao)):
170.                   if FreqPalavrasDosComentariosClassificacao[j][0] not in palavras
    Ignoradas:
171.
172.                       palavrasClassificacao.append(FreqPalavrasDosComentariosClass
    ificacao[j][0])
173.                       freqPalavrasClassificacao.append(FreqPalavrasDosComentariosC
    lassificacao[j][1])
174.                       #print(palavrasClassiflicacao)
175.                       #print(freqPalavrasClassiflicacao)
176.
177.               maxElement1 = max(freqPalavrasClassificacao)
178.               elementoClassificacao.append(maxElement1)
179.               palavra1 = palavrasClassificacao[freqPalavrasClassificacao.index(max
    Element1)]
180.               print("1º Palavra  com maior frequencia:   ", palavra1, " Frequencia
    : ", maxElement1)
181.               freqPalavrasClassificacao.remove(maxElement1)
182.               palavrasClassificacao.remove(palavra1)
183.               maxElement2 = max(freqPalavrasClassificacao)
184.               elementoClassificacao.append(maxElement2)
185.               palavra2 = palavrasClassificacao[freqPalavrasClassificacao.index(max
    Element2)]
186.               print("2º Palavra  com maior frequencia:   ", palavra2, " Frequencia
    : ", maxElement2)
187.               elementoClassificacao = padronizaClassiflicacao(elementoClassificaca
    o, freqPalavrasClassificacao)
188.               elementoClassificacao.append(bias)
189.               print("Elementos Padronizados: ", elementoClassificacao)
190.
191.               if classification(elementoClassificacao) == -1:
192.                   print("Comentário: Negativo ")
193.                   if (dadosClassificacao.target[i] == 1):
194.                       print("Classiflicou correto...")
195.                       taxaDeAcerto += 1
196.                   else:
197.                       print("Classiflicou errado...")
198.               if classification(elementoClassificacao) == 1:
```

```
199.                     print("Comentário: Positivo ")
200.                     if (dadosClassificacao.target[i] == 0):
201.                         print("Classiflicou correto...")
202.                         taxaDeAcerto += 1
203.                     else:
204.                         print("Classiflicou errado...")
205.                 elementoClassificacao = list()
206.
207.             print("Taxa de Acerto: ",((taxaDeAcerto/tamanhoClassificacao)*100),"%")
```

3.4 Parâmetros utilizados no Adaline

A RNA implementada possuía os seguintes parâmetros da tabela 1 definidos empiricamente. Em relação aos parâmetros não houve variação dos mesmos na busca de melhores resultados, foram utilizados sempre estes parâmetros.

Tabela 1 - Parâmetros usados na RNA Adaline

Parâmetros	Descrição	Valor Utilizado
eta	Taxa de aprendizagem	0.001
maxseason	Máximo de épocas	3000
tol	Tolerância ao erro	0.05
N	Quantidade de dados	Quantidade de dados do intervalo
averageErrorSeasonToSeason	Erro Médio Época a Época	Calculado em tempo de execução
data	Dados de treino	Comentários carregados
yEsperado	Lista de valores esperados na saída da RNA (1 ou -1)	Calculado em tempo de execução

W	Lista de pesos da RNA	Começa com números aleatórios, reais e pequenos
bias	Peso do neurônio	-1

3.5 Resultados

Como resultado dos experimentos realizados através da validação cruzada, esperava - se que a RNA Adaline conseguiria separar bem os comentários, mas não ocorreu, a RNA sempre informava como resultado que um comentário era negativo, independente de ele ser positivo ou não, e com isso em todos os cruzamentos ela sempre acertava 50 % dos comentários, visto que 50% era negativo.

REFERÊNCIAS

da Silva, I.N., Spatti D., e Flauzino, R. (2010). Redes neurais artificiais para engenharia e ciências aplicadas: curso prático, Artliber Editora Ltda, São Paulo, SP, Brasil.