

Avaliação I - Computação Evolutiva

Daniel Alves Costa¹

¹Curso de Engenharia de Computação
Universidade Estadual de Feira de Santana (UEFS)
CEP: 44036-900 – Avenida Transnordestina – S/N – Novo Horizonte
Feira de Santana – Bahia – Brasil

dancostafsa@hotmail.com

Resumo. *Inteligência artificial está, cada vez mais, conquistando espaço em todas as áreas de estudo. Uma de suas vertentes é a Computação Evolutiva. Como forma de iniciar o ensino desse grande ramo, foi proposto pelo professor Moab De Jesus da disciplina EXA867 - COMPUTAÇÃO EVOLUTIVA, um projeto, que utilizando de uma base de dados referente a pacientes com ou sem câncer, deve-se aplicar o que foi aprendido para realizar o processo de classificação e regressão de dados.*

1. Introdução

Inteligência artificial(IA) é o futuro para muitas áreas de pesquisa. Com sua capacidade de simular os pensamentos humano, com foque em resolução de problemas, levando uma máquina a ser "inteligente", sendo de grande ajuda para solução de diversas dificuldades atuais que exigem grande presença humana.

A IA possui diversos ramos dentro de si, desde aprendizado de máquina até computação evolucionária. Sendo computação evolucionária o ponto de estudo desta disciplina.

O objetivo desse relatório é a exibição de técnicas utilizadas para resolução de um problema proposto pelo professor da disciplina EXA867 - COMPUTAÇÃO EVOLUTIVA, assim como os resultados obtidos e as análises observadas.

A atividade proposta consiste em duas atividades, utilizando de uma base de dados contendo dados de pacientes e informando se eles possuem, ou não, câncer, a partir disso fazer:

- **Atividade 1 - Classificação** - Criar um classificador e assim identificar se, determinado paciente, possui ou não câncer.
- **Atividade 2 - Regressão** - Utilizando da função cardioide disponibilizada, deve-se utilizar de uma rede neural para aproxima-la. Assim, utilizando da função, gerar um dataset, e utilizar para treinar e testar a rede.

Além de ser necessário a utilização de redes neurais do tipo Multilayer Perceptron, ou seja, uma rede neural semelhante a rede perceptron porém com mais de uma camada de neurônios oculta, contendo um número indeterminado de neurônios.

2. Metodologia

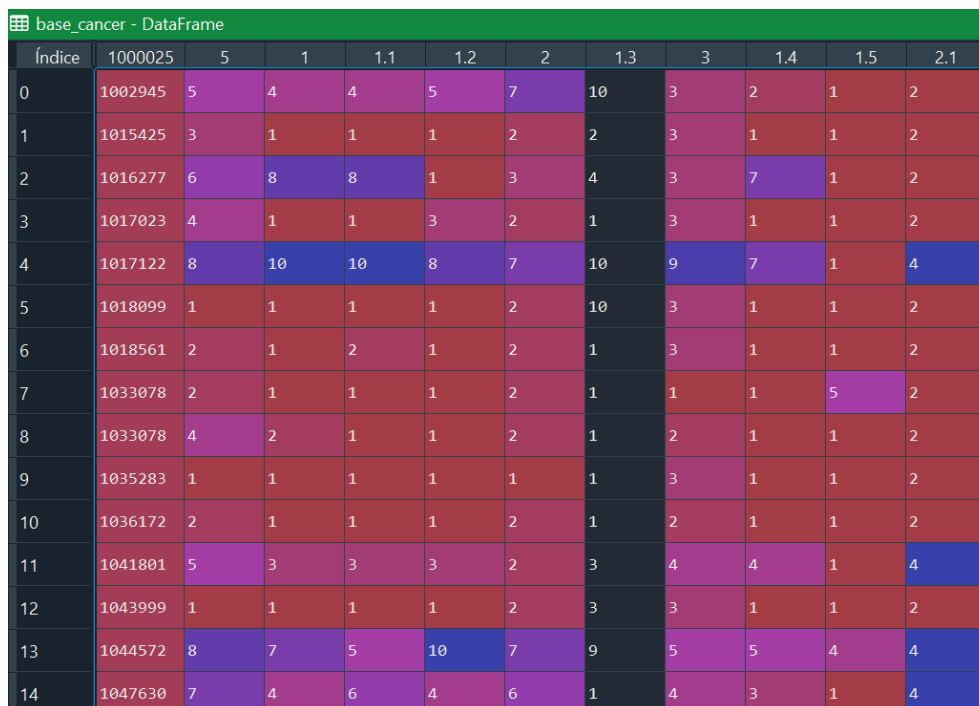
Existem diversos procedimentos que podem ser utilizados para confecção de uma rede neural, desde o tratamento dos dados até o treino da rede. Para a realização das atividades foram seguidos os seguintes passos e ferramentas:

2.1. Atividade 1: Classificação

Para toda a realização foi utilizado a linguagem de programação Python[Foundation a] em sua versão 3.7, juntamente com a IDE Spyder na versão 4.0.1.

2.1.1. Leitura de Dados

Foi disponibilizado um dataset chamado "breast-cancer.csv" contendo 11 colunas e 699 linhas. Foi utilizado a biblioteca **Pandas** [Team b] para realizar a leitura desse arquivo csv e assim poder manipular os dados como um DataFrame.



Índice	1000025	5	1	1.1	1.2	2	1.3	3	1.4	1.5	2.1
0	1002945	5	4	4	5	7	10	3	2	1	2
1	1015425	3	1	1	1	2	2	3	1	1	2
2	1016277	6	8	8	1	3	4	3	7	1	2
3	1017023	4	1	1	3	2	1	3	1	1	2
4	1017122	8	10	10	8	7	10	9	7	1	4
5	1018099	1	1	1	1	2	10	3	1	1	2
6	1018561	2	1	2	1	2	1	3	1	1	2
7	1033078	2	1	1	1	2	1	1	1	5	2
8	1033078	4	2	1	1	2	1	2	1	1	2
9	1035283	1	1	1	1	1	1	3	1	1	2
10	1036172	2	1	1	1	2	1	2	1	1	2
11	1041801	5	3	3	3	2	3	4	4	1	4
12	1043999	1	1	1	1	2	3	3	1	1	2
13	1044572	8	7	5	10	7	9	5	5	4	4
14	1047630	7	4	6	4	6	1	4	3	1	4

Figure 1. Base de dados breast-cancer.csv

Sem os nomes dos atributos, esses tiveram que ser adicionados via código, ainda sendo utilizado funções dos próprio pandas.

```

8 import pandas as pd
9 import random
10
11 ##### LEITURA DATASET #####
12 #leitura do arquivo .csv
13 base_cancer = pd.read_csv('./data/breast-cancer.csv')
14
15 #nomeando as colunas
16 base_cancer.rename( columns={'1000025' : 'id'}, inplace=True )
17 base_cancer.rename( columns={'5' : 'Clump_Thickness'}, inplace=True )
18 base_cancer.rename( columns={'1' : 'Cell_Size'}, inplace=True )
19 base_cancer.rename( columns={'1.1' : 'Cell_Shape'}, inplace=True )
20 base_cancer.rename( columns={'1.2' : 'Marginal_Adhesion'}, inplace=True )
21 base_cancer.rename( columns={'2' : 'Epithelial_Cell_Size'}, inplace=True )
22 base_cancer.rename( columns={'1.3' : 'Bare_Nuclei'}, inplace=True )
23 base_cancer.rename( columns={'3' : 'Bland_Chromatin'}, inplace=True )
24 base_cancer.rename( columns={'1.4' : 'Normal_Nucleoli'}, inplace=True )
25 base_cancer.rename( columns={'1.5' : 'Mitoses'}, inplace=True )
26 base_cancer.rename( columns={'2.1' : 'Class'}, inplace=True )
27
28

```

Figure 2. Nome das colunas. Atributos

2.1.2. Tratamento de Dados

Ainda utilizando funções da biblioteca pandas, primeiramente nessa parte do projeto foi verificado a existência de valores nulos, vendo que não existia nenhum na base foi realizado exibição em tela(print) de todos os valores, e suas quantidades, presentes em cada coluna de atributo.

```

Clump_Thickness - {1: 145, 5: 130, 3: 108, 4: 80, 10: 69, 2: 50, 8: 46, 6: 34, 7: 23, 9: 14}
Cell_Size - {1: 384, 10: 67, 3: 52, 2: 45, 4: 40, 5: 30, 8: 29, 6: 27, 7: 19, 9: 6}
Cell_Shape - {1: 353, 2: 59, 10: 58, 3: 56, 4: 44, 5: 34, 7: 30, 6: 30, 8: 28, 9: 7}
Marginal_Adhesion - {1: 407, 3: 58, 2: 58, 10: 55, 4: 33, 8: 25, 5: 23, 6: 22, 7: 13, 9: 5}
Epithelial_Cell_Size - {2: 386, 3: 72, 4: 48, 1: 47, 6: 41, 5: 39, 10: 31, 8: 21, 7: 12, 9: 2}
Bare_Nuclei - {'1': 402, '10': 132, '2': 30, '5': 30, '3': 28, '8': 21, '4': 19, '?': 16, '9': 9, '7': 8, '6': 4}
Bland_Chromatin - {2: 166, 3: 165, 1: 152, 7: 73, 4: 40, 5: 34, 8: 28, 10: 20, 9: 11, 6: 10}
Normal_Nucleoli - {1: 443, 10: 61, 3: 44, 2: 36, 8: 24, 6: 22, 5: 19, 4: 18, 9: 16, 7: 16}
Mitoses - {1: 579, 2: 35, 3: 33, 10: 14, 4: 12, 7: 9, 8: 8, 5: 6, 6: 3}
Class - {2: 458, 4: 241}

```

Figure 3. Valores presentes nos atributos, e suas quantidades

A partir disso foi possível a detecção do valor '?' em Bare_Nuclei, assim foi realizado o tratamento atribuindo um número randomico, utilizando a biblioteca **Random**[Foundation b], de 1 a 10, para as determinadas células que continham o valor '?'.

Logo depois convertendo os valores presentes em `Bare_Nuclei` de strings para números.

Após isso, a classe que será prevista '**Class**' foi removida da base de dados original e armazenada em uma outra variável. Além de transformar seus valores 2(o paciente não possui câncer) e 4(o paciente possui câncer) para, respectivamente 0 e 1.

2.1.3. Classificação dos Dados

Na parte da classificação foi utilizado a biblioteca **Scikit-Learn**[Developers e] para realizar todos os procedimentos.

Inicialmente foi utilizado a função '`train_test_split`' [Developers f] para dividir a base de dados. Obtendo quatro variáveis, duas contendo valores para treinamento e outras duas para teste. Foi separado 25% da base direcionada para o teste. Essa separação é ideal caso se queira saber o valor de acurácia do modelo e o quanto se pode melhorar a classificação sem que haja perda na capacidade de generalização. E com seu '`random_state`' igual a 0, visto que esse parâmetro, quando não é atribuído um valor, os valores de teste, a cada vez que a linha de código é executada, as variáveis adquirem um valor diferente.

previsao_teste	Array of int64 (175,)	[0 0 0 ... 1 0 1]
previsao_treinamento	Array of int64 (524,)	[1 0 1 ... 0 1 1]
previsores	Array of int64 (699, 9)	<div><div>[5 4 3 ... 2 3 1]</div><div>[9 1 2 ... 7 7 2]</div></div>
previsores_teste	Array of int64 (175, 9)	<div><div>[4 1 1 ... 2 1 1]</div><div>[3 3 2 ... 3 5 1]</div></div>
previsores_treinamento	Array of int64 (524, 9)	<div><div>[10 10 7 ... 10 10 3]</div><div>[1 1 1 ... 1 1 1]</div></div>

Figure 4. Variáveis resultantes do `train_test_split`

Na imagem 4, as variáveis destacadas são as resultantes da separação. As nomeadas '`previsores`' são os atributos utilizados para realizar a previsão e as '`previsão`' são os valores que devem ser previstos, o atributo '`Class`'.

Após a separação foi iniciada a parte de montagem da rede. Utilizando a função **MLPClassifier** [Developers a] para toda a rede. Com ela foi possível realizar as operações cumprindo a requisição do projeto de se utilizar um rede do tipo Multilayer Perceptron.

Com o **MLPClassifier** é possível alterar algumas variáveis de como a rede será montada, buscando a melhor parametrização de acordo com os dados que o usuário disponibiliza. Com a realização de teste buscando os melhores resultados, foi decidido que alguns parâmetros iriam ser alterados:

- **random_state** - foi atribuído o valor igual a 0, para que sempre que o código for executado, o valor da variável permaneça o mesmo.
- **max_iter** - atribuindo o valor de 300, esse parâmetro é referente ao número máximo de interações de interações que a rede pode realizar.

Todos os outros parâmetros foram mantidos em seus valores padrões.

Com a variável da rede criada foi realizado seu treinamento e, logo após, armazenado os valores de previsão para a variável 'previsores_teste', que serão utilizados posteriormente.

2.1.4. Análise dos Dados

Foram utilizados também as funções de 'score' da própria MLPClassifier, e utilizado a função 'confusion_matrix' [Developers c] da mesma biblioteca. Essa função tem como objetivo mostra a frequência de classificação para cada classe resultante do modelo, separando-as em verdadeiro positivo(TP), falso positivo(FP), verdadeiro negativo(TN) e falso negativo(FN).

- **Verdadeiro Positivo(TP)** - Quando a classe que se deseja foi prevista corretamente.
- **Falso Positivo(FP)** - Quando a classe que se deseja foi prevista incorretamente.
- **Verdadeiro Negativo(TN)** - Quando a classe que não se deseja foi prevista corretamente.
- **Falso Negativo(FN)** - Quando a classe que não se deseja foi prevista incorretamente.

Além de utilizar a função 'mean_absolute_error'[Developers b] para calcular a taxa de erro da rede neural.

2.2. Atividade 2: Regressão

Para a segunda atividade sendo um processo de Regressão, foi utilizado, também, a linguagem de programação Python[Foundation a] em sua versão 3.7, juntamente com a IDE Spyder na versão 4.0.1.

2.2.1. Criação Dataframe

Primeiramente para a realização dessa atividade foi utilizada uma equação do cardioide, representada pela equação 1.

$$y = \sqrt{2\sqrt{a^3(a+2x)} + 2a^2 + 2ax - x^2} \quad (1)$$

Foi utilizado a biblioteca **Random** [Foundation b] para gerar os valores referentes a 'x', 100 números randomicos definidos entre 1 e 100, e com o valor de 'a' definido como 2, após alguns teste de verificação na função esse foi o melhor valor encontrado.

Utilizando dos valores de 'x', do valor de 'a' e da equação 1, foram definidos 100 valores para 'y'. Depois esses valores obtidos foram salvos em um arquivo csv para ser lido posteriormente.

2.2.2. Regressão dos Dados

Após todo o processo de construção do DataFrame e da sua leitura, foi iniciado o processo de regressão da rede neural.

Utilizando da biblioteca **Scikit-Learn** [Developers e] foi utilizado a função `'train_test_split'` [Developers f] para a separação dos dados entre treinamento e teste.

Após a divisão foi iniciado a fase de treino da rede. Utilizando da função **ML-PRegressor** [Developers d] foi possível realizar a proposta da atividade, assim como, cumprindo o requisito de ser uma rede neural multilayer perceptron.

Os parâmetros utilizados para gerar a rede foram:

- **random_state** - com valor igual a 0, para toda execução do código os valores não serem alterados,
- **max_iter** - com valor igual a 200(seu valor padrão), sendo o número máximo de interação que a rede pode realizar.
- **solver** - sendo `'lbfgs'`, referente ao tamanho da base, foi escolhido esse parâmetro por ele ser mais eficiente para DataFrames pequenos, como foi o caso.

Foi realizado o treinamento da rede, e salvo para posterioridade os valores de precisão da rede referente aos valores de teste.

2.2.3. Análise dos Dados

Foi medido seu score e a exibição de alguns atributos da rede utilizado como: o número de interação realizadas, o números de camadas e a função de ativação de saída, sendo suas chamadas, respectivamente: `'n_iter_'`, `'n_layers_'` e `'out_activation_'`, além da taxa de erro da rede.

Para a exibição de dados foi utilizado uma biblioteca para construção de gráficos. Neste projeto foi utilizado a **Matplotlib** [Team a] para a montagem de um gráfico comparando os valores obtidos com a rede neural com os dados reais.

3. Resultados e Discussão

Após todo o processo de criação, tratamento e treino da rede neural, foi obtido os resultados referente a cada atividade.

3.0.1. Atividade 1: Classificador

Na primeira atividade, consistindo da classificação dos dados, prevendo se determinado paciente têm ou não têm câncer, foram calculados o score da rede, a taxa de erro e gerado uma matriz de confusão para melhor visualização dos dados:

- **Score:** Através do próprio classificador é possível medir os score da rede, sendo assim o score calculado foi de, aproximadamente: **0.96**.

```

In [6]: print( 'Score da rede: {}'.format(score))
Score da rede: 0.9657142857142857

In [7]: classificador.score(previsores_teste, previsao_teste)
Out[7]: 0.9657142857142857

In [8]:

```

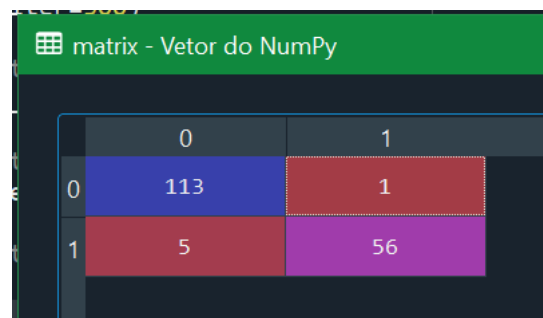
Figure 5. Score Classificador

- **Taxa de erro:** Utilizando da função para calcular taxa de erro absoluto foi encontrado o valor de, aproximadamente: **0.04**.

```
Taxa de erro: 0.04
```

Figure 6. Taxa de Erro Classificador

- **Matriz de Confusão:** Com a função de criação da matriz do sklearn, utilizando os valores finais de teste e os resultados obtidos foi gerada a tabela abaixo:



	0	1
0	113	1
1	5	56

Figure 7. Matriz de Confusão do Classificador

Com essa imagem 7 é possível concluir que foi classificado, erroneamente, 6 pacientes. 1 sendo previsto que possui câncer sendo que não possui e 5 paciente tendo câncer ele previu que não.

3.0.2. Atividade 2: Regressor

Na segunda atividade, sendo a regressão utilizando uma equação do cardioide, foram calculados, também, o score e a taxa de erro da rede, além da construção de um gráfico mostrando o desempenho da rede em relação aos dados reais.

- **Score:** O score calculado pela própria rede foi com o valor de, aproximadamente: **0.99**.

```

In [54]: regressor.score(previsores_teste, previsao_teste)
Out[54]: 0.9999999995280346

```

Figure 8. Score Regressor

- **Taxa de erro:** A taxa de erro absoluto calculo a partir da rede foi de, aproximadamente: **0.0002**.

Taxa de erro: 0.00025988019256573125

Figure 9. Taxa de Erro Regressor

- **Gráfico:** O gráfico montado utilizando os valores reais do DataFrame e os valores que a rede encontrou está logo abaixo:

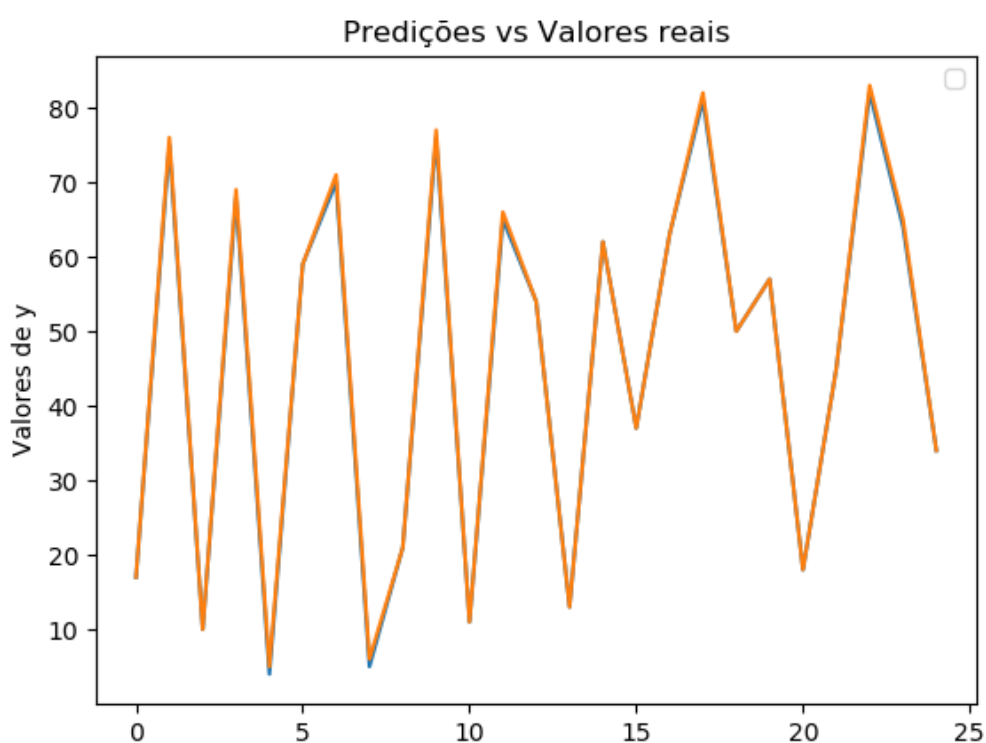


Figure 10. Gráfico de Regressão

O gráfico, figura 10, mostra a relação entre os valores reais de y e os valores previsto, sendo a linha laranja exibindo os valores reais e a linha azul os valores previsto.

Realizando diferentes combinações de parâmetros foi possível perceber quais ficaria melhor para cada base de dados utilizada.

Na **atividade 1** foi alternado a função de ativação entre 'identity' e 'relu' vendo que não teria diferenças com essa mudança o padrão foi mantido, ou seja, a função 'relu'. Também foi testado valores para o parâmetro solver, alternando entre 'adam' e 'lbfgs', com isso foi perceptível que o valor padrão 'adam' é mais aplicável devido até a quantidade de informações contida no DataFrame.


```
300
3
logistic
```

Figure 11. Número de interações, número de camadas e função de ativação de saída do classificador.

O número de interações que o classificador utilizou foram 300 interações, contendo 3 camadas ocultas e a função de ativação de saída sendo 'logistic', como mostra a imagem 11.

Já na **atividade 2** a função de ativação também foi testada com outras além da padrão, porém, seu valor padrão, 'relu', foi mantido devido a utilização das outras funções terem diminuído o valor de score. Outros tipos de solver também foram testados como o 'sgd', este gerando até um resultado negativo, e o 'adam' porém o 'lbfgs' foi escolhido devido ao melhor resultado encontrado.

```
66
3
identity
```

Figure 12. Número de interações, número de camadas e função de ativação de saída do regressor.

Como é possível identificar na imagem 12 o número de interações feitas pela rede é de 66 interações, assim como 3 camadas ocultas foram utilizadas, sendo a função de ativação de saída a 'identity'.

4. Conclusão

De modo geral, os requisitos de classificação da base de dados prevendo o diagnóstico do paciente, assim como os da regressão de aproximação da equação do cardiode foram devidamente atendido, juntamente com os requisitos do projeto, exibição e análise de score e taxa de erro da rede neural utilizada e também a obrigatoriedade de ser uma rede Multilayer Perceptron.

Os passos executados de pré-processamento, os testes e análises das melhores configurações, auxiliaram a melhorar os resultados obtidos.

References

- Developers, S.-L. Documentação classificador multi-layer perceptron. [Acessado em: 15 Setembro 2020.].
- Developers, S.-L. Documentação erro absoluto médio. [Acessado em: 16 Setembro 2020.].
- Developers, S.-L. Documentação matriz de confusão. [Acessado em: 16 Setembro 2020.].
- Developers, S.-L. Documentação regressor multi-layer perceptron. [Acessado em: 19 Setembro 2020.].
- Developers, S.-L. Documentação scikit-learn. [Acessado em: 14 Setembro 2020.].

Developers, S.-L. Documentação train test split. [Acessado em: 14 Setembro 2020.].

Foundation, P. S. Documentação python. [Acessado em: 9 Setembro 2020.].

Foundation, P. S. Documentação random. [Acessado em: 12 Setembro 2020.].

Team, M. D. Documentação matplotlib. [Acessado em: 19 Setembro 2020.].

Team, P. D. Documentação pandas. [Acessado em: 9 Setembro 2020.].