Computação Evolutiva - Redes Neurais Artificiais

Adlla Katarine Aragão Cruz Passos[1]

[1]Engenharia de Computação – Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, s/n – Novo Horizonte - 44036-900 – Feira de Santana – BA – Brasil

adllakatarine@hotmail.com

Resumo. *Este relatório apresenta a resolução de problema proposto na disciplina EXA867 – Computação Evolutiva 2019.2 E. O objetivo deste trabalho é o desenvolvimento de duas Redes Neurais Artificiais (RNA's), sendo uma classificação e uma regressão.*

1. Introdução

O aprendizado de máquina é um ramo da inteligência artificial, com a ideia de que máquinas podem detectar padrões, aprender com dados e tomar decisões sozinhas. Estes algoritmos são implementados com o objetivo de fazer previsões a partir de dados que recebem e aprender com seus erros, assim melhorando seus resultados, não sendo necessário programá-las com frequência para que elas *aprendam*.

Uma das abordagens do aprendizado de máquina é a **supervisionada**, que consiste em um treinamento com os dados pré-definidos. Resumidamente, dividi-se um *dataset*, **com sua classificação final já definida**, em treinamento e teste; treina-se uma parte desses dados e ao final os testa com o restante dos dados para saber o quanto este algoritmo aprendeu. A classificação e regressão são duas sub-categorias de aprendizagem supervisionada, sendo a primeira o processo de tomar algum tipo de entrada e atribuir um rótulo a ela, enquanto que a segunda é usada quando o valor que está sendo previsto difere de um "sim ou não" e que siga um espectro contínuo.

Um algoritmo de aprendizagem de máquina que vem sendo bastante utilizado são as Redes Neurais Artificiais. Ela usa camadas de neurônios matemáticos para processar dados, compreender a fala humana e reconhecer objetos visualmente. A informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. A primeira camada em uma rede é chamada de camada de entrada, enquanto a última é chamada de camada de saída. Todas as camadas entre as duas são referidas como camadas ocultas. Cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação [Academy 2019].

Sendo assim, foi proposto o desenvolvimento de duas RNA's, sendo eles: um classificador afim de identificar se um determinado paciente tem câncer ou não (a partir do dataset "Breast Cancer"disponibilizado pelo professor); e um regressor, com um dataset gerado a partir de uma entrada com valores aleatórios e uma saída dada a equação do cardioide (também disponibilizada pelo professor). Por conseguinte, foi necessário o uso de métodos como accuracy, matriz de confusão e outros para avaliação dos modelos preditivos.

2. Metodologia

Para o desenvolvimento do código das duas atividades, foi escolhida a linguagem de programação *Python* (na versão 3.7), a mais popular no meio da Ciência de Dados

[Python].

Na manipulação e criação das bases de dados, foi utilizada a biblioteca *Pandas*, que fornece ferramentas de análise e estruturas de dados. Também foi necessário o uso da biblioteca *Random* para preenchimento (aleatório dentro do especificado) de valores inconsistentes em um *dataset* e na criação dos valores de entrada do outro [Pandas] [Random].

Por fim, foi utilizado uma biblioteca de aprendizagem de máquina, *Sklearn*, para o uso dos algoritmos de classificação, regressão e avaliação dos modelos de predição [scikit learn].

2.1. Classificação

Como já citado antes, a classificação busca aproximar uma função de mapeamento (f) das variáveis de entrada (X) para variáveis de saída discretas (y). Para esta tarefa de classificação, foi disponibilizado o dataset "Breast Cancer", contendo informações que indicam se os pacientes tem ou não câncer. O objetivo é criar um classificador que faça essa identificação.

Inicialmente, foi necessário fazer o pré-processamento dos dados, uma etapa importante onde os dados são limpos, tratados e transformados. Primeiro, os atributos precisaram ser renomeados (pelos nomes disponibilizados pelo professor), pois estavam com números no lugar, não sendo possível sua correta identificação. Depois, foram feitas verificações se haviam valores nulos ou inconsistentes e assim foi encontrados valores com '?' no atributo "Bare Nuclei"; seu tratamento foi feito através da biblioteca random (retorna um valor aleatório), substituindo-os por um valor aleatório entre 1 e 10.

Com os dados pré-processados, já pode prepara-los para o treinamento. Os dados foram divididos em **previsores** (com 9 atributos) e **classe** (com 1 atributo). O próximo passo é dividir esses dados entre treinamento e teste, sendo ideal deixar uma parte menor de teste para validar o treinamento. O método **train_test_split** do pacote *Sklearn* foi usado e dividiu os dados em 70% para treinamento e 30% para teste.

A rede neural é criada, instanciando-se a classe **MLPClassifier**. Esta classe trabalha com *Multi-layer Perceptron*, um dos requisitos deste trabalho. A *MLPClassifier* contém muitos parâmetros para uma melhor configuração da rede neural e a maioria foram mantidos seus parâmetros default, contudo houve modificações em alguns que melhoraram o treinamento do classificador [MLPClassifier].

Parâmetros:

- **max_iter**: define o máximo de iterações. Seu valor foi alterado para **100** iterações, pois não houve diferença no resultado usando seu valor default 200.
- **learning_rate_init**: taxa de aprendizagem inicial usada, ele controla o tamanho do passo na atualização dos pesos. Foi alterado para 0.01.
- **activation**: função de ativação para a camada oculta. Foi definido o valor **logistic** que usa a função sigmóide logística, retorna f(x) = 1 / (1 + exp (-x)).

Após isso, a rede foi treinada e então testada com os dados que foram separados para teste, afim de encontrar sua taxa de acerto.

2.2. Regressão

Para a tarefa de regressão, foi dada uma equação do cardioide, como pode ser vista na Figura 1. Sendo **x** a variável de entrada, **a** uma constante e **y** a variável de saída, o propósito seria gerar um dataset com **x** e **y**, sendo x variáveis geradas aleatoriamente e obtendo y a partir da equação. E por fim, treinar e testar a RNA.

$$y = \pm\sqrt{2\sqrt{a^3\,(a+2\,x)}\; + 2\,a^2 + 2\,a\,x - x^2}$$

Figura 1. Equação do cardioide.

Para gerar os dados para o *dataset*, duas listas foram instanciadas, uma para guardar os valores de **x** e outra para os valores de **y**, bem como a constante **a** recebeu o valor **3**. Como os valores de x seriam aleatórios, foi usada a biblioteca *random*. No laço de repetição **for**, que foi definido 400 iterações, cada iteração definia um valor aleatório de x, o adicionava na **lista_X**, calculava o valor de y e o adicionava na **lista_Y**. Para o calculo de y, a Figura 1 foi implementada com o auxilio da biblioteca *math* para a raiz quadrada e o método *abs* que retorna o valor absoluto para não haver problemas com raiz quadrada negativa. Com os valores de *entrada* e *saída* prontos, foi criado e salvo o *DataFrame*.

Abrindo o dataset com o Pandas, estes dados foram divididos em previsor e classe, e usando o método **train_test_split** foi dividido 70% dos dados em treinamento e 30% em teste, assim como aconteceu com os dados da tarefa de classificação.

A rede neural é criada, instanciando-se a classe **MLPRegressor**, sendo este um regressor *Multi-layer Perceptron*, um dos requisitos deste trabalho. Dentre os muitos parâmetros importantes do *MLPRegressor*, houve modificações em alguns que melhoraram o treinamento da rede [MLPRegressor].

- **random_state**: determina a geração de número aleatório para pesos e inicialização de polarização, foi definido como **1**.
- **max_iter**: determina o número máximo de iterações, foi alterado para **300**.
- **solver**: otimizador de pesos, alterado para **lbfgs**.
- **activation**: função de ativação para a camada oculta. Foi definido o valor **identity**, ativação sem operação, que retorna f(x) = x.

Finalmente, a rede neural foi treinada e testada.

2.3. Métricas de Avaliação

É necessário validar e saber realmente se o classificador e o regressor criados atendem as suas necessidades. Foram usados os métodos de acurácia, score, taxa de erro e matriz de confusão.

- **Acurácia**: diz quanto o modelo acertou das previsões possíveis.
- **Score**: verifica o número que o modelo previu e o compara com o valor esperado do conjunto de treino. O resultado é um valor de 0.0 a 1.0, onde quanto mais próximo de 1.0, melhor.
- **Taxa de erro**: a mínima taxa de erro é obtida quando a taxa de acerto é máxima.

- **Matrix de confusão**: É uma métrica voltada para *modelos de classificação* e tem como objetivo calcular a quantidade de falso positivo e falso negativo; e de verdadeiro positivo e verdadeiro negativo:
  - **falso positivo**: ocorre quando no conjunto real, a classe buscada foi prevista incorretamente [Medium 2019].
  - **falso negativo**: ocorre quando no conjunto real, a classe que não foi buscada foi prevista incorretamente [Medium 2019].
  - **verdadeiro positivo**: ocorre quando no conjunto real, a classe buscada foi prevista corretamente [Medium 2019].
  - **verdadeiro negativo**: ocorre quando no conjunto real, a classe que não foi buscada foi prevista corretamente [Medium 2019].

3. Resultados e Discussões

Agora é o momento de verificar o quão bom foram os treinamentos de classificação e regressão das redes neurais (RNA's).

3.1. Classificação

Durante os testes do classificador, como já citado anteriormente, alguns parâmetros foram modificados ao instanciar a classe MLPClassifier afim de melhorar o treinamento da rede. Como o uso de Multi-layer Perceptron era importante, foram feitos testes com seu valor default que era uma camada oculta de 100 neurônios e depois com a quantidade de neurônios calculada usando o número médio entre o tamanho da camada de entrada e o da camada de saída.

O parâmetro solver, que é um otimizador de pesos, também foi testado com seus 3 valores disponíveis: *lbfgs*, *sgd*, *adam*. O max_iter, que define o máximo de iterações, também sofreu variação do seu valor para 50, 100 e depois 200, bem como o activation (função de ativação para a camada oculta) que foi testado com todos os seus valores disponíveis.

Após esses testes, notou-se que a rede neural se adaptou melhor com os seguintes parâmetros definidos: hidden_layer_sizes continuou com seu valor default de 100 neurônios na camada oculta; o solver também permaneceu com o valor default Adam, que é baseado em gradiente estocástico; max_iter foi definido com 100 iterações; e o activation com logistic. Assim, obteve-se os seguintes resultados nas métricas de avaliação:

- Acurácia de 97.14% (ou 0.9714)
- Taxa de erro de 5.71% (ou 0.0571)
- Matriz de confusão na Figura 2: resumidamente, na primeira linha 127 foram classificados corretamente, enquanto que houve 6 erros; já na segunda linha todos foram classificados corretamente.

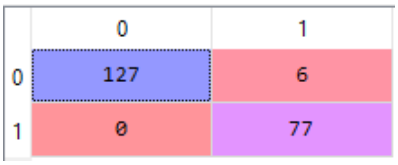

Figura 2. Matriz de confusão.

Os testes feitos com os outros parâmetros diferentes desses definidos, obtiveram uma menor taxa de aprendizagem: a acurácia ficou entre 91.42% e 95.71%; e taxa de erro entre 8.57% e 17.14%. Assim, pode-se notar uma grande diferença de resultados, principalmente na taxa de erro.

3.2. Regressão

Nos testes de regressão, também foi preciso alterar os valores de alguns parâmetros afim de melhorar o treinamento da RNA. Assim como aconteceu com o modelo de classificação, foram testados dois valores diferentes no parâmetro hidden_layer_sizes, com a quantidade de neurônios na camada oculta. O parâmetro max_iter teve seu valor alterado para 100, 200 e 300 durante os testes. O solver, random_state e activation também foram testados.

Por fim, foi notada uma melhor adaptação com estes valores de parâmetros definidos: o hidden_layer_sizes foi deixado com seu valor default; o max_iter melhorou o resultado quando foi definido como 300; o random_state, que determina a geração de número aleatório para pesos e inicialização de polarização foi ativado, ou seja, foi definido como 1; sover teve seu valor definido como lbfgs, pois quando o sgd foi usado retornava um valor negativo do método predict; e o activation alterado para identity. Com isso, os resultados obtidos das métricas de avaliação foram estes:

- Acurácia de 99.99% (ou 0.9999)
- Taxa de erro de 23.33% (ou 0.2333)

Pode-se notar que a taxa de erro, continuou alta mesmo após os testes, sendo este o mínimo de erro que a rede conseguiu alcançar.

Para os outros valores de parâmetros definidos, os resultados nas métricas de avaliação mostraram que a rede obteve uma menor taxa de aprendizagem: a acurácia variou de 36% a 99.99% e a taxa de erro variou de 24.56% a 61.68%.

4. Conclusão

Os produtos desenvolvidos atendem a todos os requisitos especificados. As Redes Neurais de classificação e regressão obtiveram melhores resultados após o testes feitos, demonstrando a importância do mesmo durante o treinamento, verificando sempre com quais parâmetros o classificador e regressor melhor se adaptam. Contudo, melhorias poderiam ser feitas no pré-processamento, bem como mais testes com os parâmetros e utilizando outras métricas de avaliação.

Referências

Academy, D. S. (2019). Deep learning book.

Medium (2019). Entendendo o que é matriz de confusão com python.

MLPClassifier, M. L. i. P. Disponível em: https://scikit-learn.org/stable/. Acesso em Setembro de 2020.

MLPRegressor, M. L. i. P. Disponível em: https://scikit-learn.org/stable/. Acesso em Setembro de 2020.

Pandas. Disponível em: https://pandas.pydata.org. Acesso em Setembro de 2020.

Python. Disponível em: https://docs.python.org/3/. Acesso em Setembro de 2020.

Random. Disponível em: https://docs.python.org/3/library/random.html. Acesso em Setembro de 2020.

scikit learn, M. L. i. P. Disponível em: https://scikit-learn.org/stable/. Acesso em Setembro de 220.