UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
TÓPICOS ESPECIAIS EM INTELIGÊNCIA ARTIFICIAL - ELE0606

Docente:

José Alfredo Ferreira Costa

Discente:

Douglas Wilian Lima Silva

Semestre 2023.2 - Turma 01

Implementação de Redes Neurais Multilayer Perceptron

Natal - RN

Outubro de 2023

Sumário

1 Introdução

As redes neurais Multilayer Perceptron (MLP) são componentes fundamentais no ramo de aprendizagem de máquina (Machine Learning), aprendizado profundo (Deep Learning) e processamento de dados. Essas estruturas visam copiar os neurônios biológicos, através de diversas camadas de neurônios artificiais interconectados entre si.

As MLPs são notáveis por sua capacidade de aprender e generalizar a partir de dados complexos, tornando-se uma ferramenta versátil em uma variedade de aplicações, desde reconhecimento de padrões e processamento de linguagem natural até previsão de séries temporais e classificação de dados.

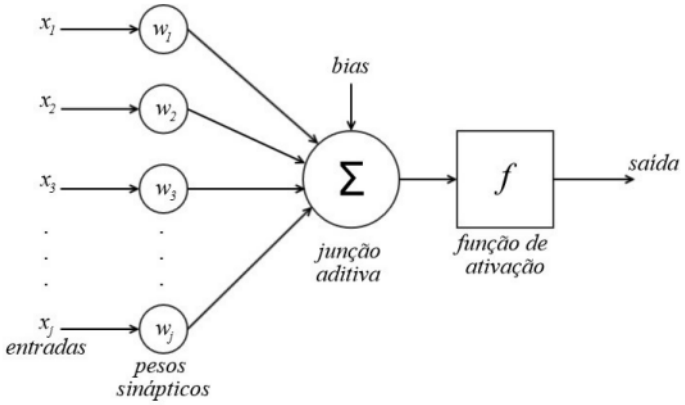

Figura 1: Esquema - Rede Neural.

Neste contexto, o presente estudo tem por objetivo a implementação de redes neurais MLPs utilizando as ferramentas disponíveis na biblioteca da Scikit-Learn através da análise das bases de dados trabalhadas Wine e Heart Desease. Através dessa implementação, será possível analisar o desempenho do modelo criado em comparação aos demais métodos de classificação já estudados.

2 Desenvolvimento

Como apresentado, a implementação das MLPs foi realizada para as bases de dados Wine, disponível na Scikit-Learn, e a base Heart Desease. Em ambos os casos, o objetivo é a classificação do conjunto em cada uma das bases desejadas para cada conjunto.

Para tal, as ferramentas disponíveis na Scikit-Learn são fundamentais. Elas permitem a aplicação de redes neurais artificiais apenas com a identificação de certos parâmetros de acordo com a necessidade de implementação. Com base nisso, serão apresentadas as aplicações dessas estruturas em cada uma das bases trabalhadas.

2.1 Implementações

```
#Bibliotecas utilizadas

import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Listing 1: Bibliotecas Utilizadas.

2.1.1 Base de dados Wine

O primeiro passo para a classificação correta dos dados é a análise e normalização dos mesmos. A normalização é recomendada para dados que possuem grandes variabilidades, o que ocorre nessa base de dados. Assim, através das funções presentes na MinMaxScaler(), é possível fazer com que todas as informações possam variar entre 0 e 1.

```
wine = load_wine()

df = pd.DataFrame(data = wine['data'], columns = wine['feature_names'])
Zscore = MinMaxScaler()
dfn = pd.DataFrame(Zscore.fit_transform(df), columns = df.columns)
dfn['target'] = wine.target

dfn.head() # Visualizacao do DataFrame normalizado
```

Listing 2: Carregamento e normalização dos dados.

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols
0	0.842105	0.191700	0.572193	0.257732	0.619565	0.627586
1	0.571053	0.205534	0.417112	0.030928	0.326087	0.575862
2	0.560526	0.320158	0.700535	0.412371	0.336957	0.627586
3	0.878947	0.239130	0.609626	0.319588	0.467391	0.989655
4	0.581579	0.365613	0.807487	0.536082	0.521739	0.627586

Figura 2: Dados normalizados.

De forma sequencial, foram separadas as classes de treinamento e teste usando a NumPy, para a aplicação do modelo. Nesse caso, foram utilizados 40% dos treinamento e 60% para testes.

```
i = np.random.permutation(dfn.shape[0])
d = int(0.4*len(i)) # 40% dos termos

train, test = i[:d], i[d:]

ctrain, ctest = dfn.loc[train,:], dfn.loc[test,:]

xtrain = ctrain.drop('target', axis=1)
ytrain = ctrain.target

xtest = ctest.drop('target', axis=1)
ytest = ctest.target
```

Listing 3: Separação das classes

Com a separação necessária, é possível utilizar a implementação da scikit-learn para condicionar o classificador da MLP. Nele, basicamente definimos o número de neurônios presentes nas camadas internas (hidden layers), a função de ativação utilizada e o solver.

```
classifier = MLPClassifier(hidden_layer_sizes=(64, 32), activation = '
    relu', solver = 'lbfgs')

classifier.fit(xtrain, ytrain)

ypred = classifier.predict(xtest)

acuracia = accuracy_score(ytest, ypred)

print(f'A acuracia do modelo e {acuracia*100:.3f} %')

cm = confusion_matrix(ytest, ypred)
```

Listing 4: Implementação da rede.

Neste caso, foram utilizadas duas camadas intermediárias com 64 e 32 neurônios, respectivamente e uma função de ativação relu, apresentada na figura 3 abaixo. Além disso, o solver padrão "adam"foi substituído pelo "lbfgs"que possibilitou uma melhor convergência para esse banco de dados.

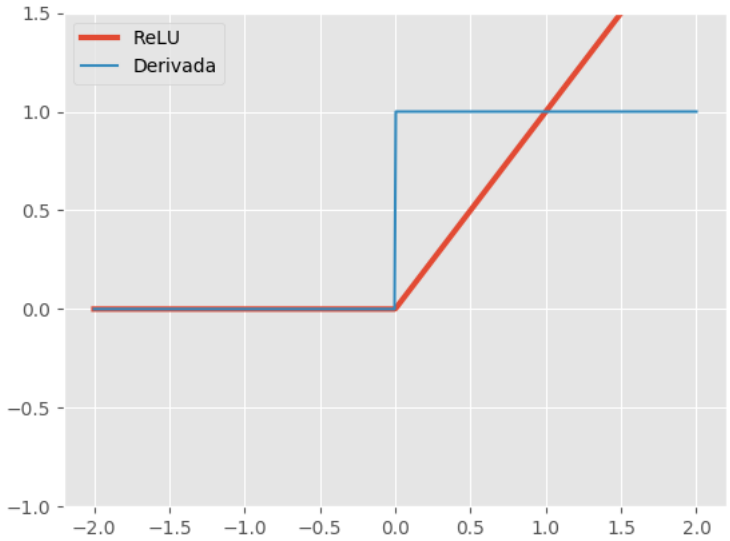

Figura 3: Função de ativação.

Para essa simulação, a acurácia do modelo foi de 95,33%, se equivocando em apenas 5 itens durante a classificação. O resultado pode ser visualizado na matriz de confusão abaixo.

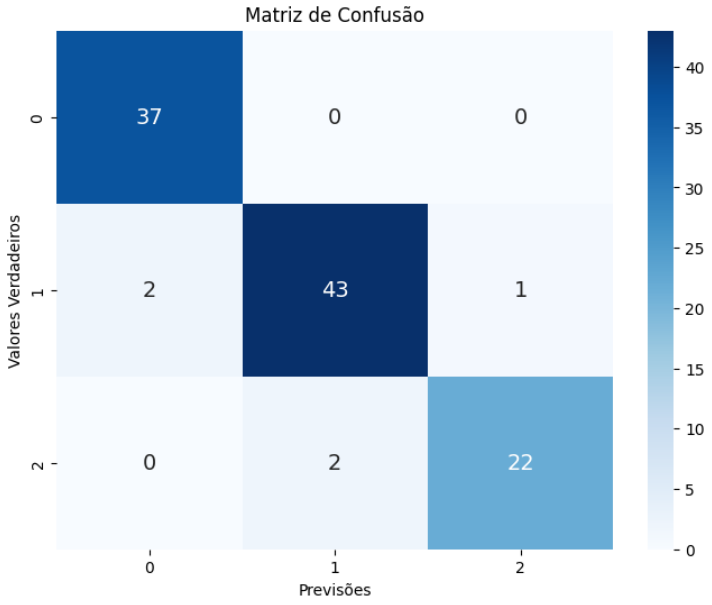

Figura 4: Matriz de confusão.

2.1.2 Base de dados Heart Desease

A implementação da rede neural utilizando a base de dados heart desease foi feita exatamente da mesma forma do caso anterior: Os dados foram importados, normalizados e então separados para a classificação.

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
0	0.479167	1.0	0.0	0.292453	0.196347	0.0	0.5	0.740458
1	0.500000	1.0	0.0	0.433962	0.175799	1.0	0.0	0.641221
2	0.854167	1.0	0.0	0.481132	0.109589	0.0	0.5	0.412214
3	0.666667	1.0	0.0	0.509434	0.175799	0.0	0.5	0.687023
4	0.687500	0.0	0.0	0.415094	0.383562	1.0	0.5	0.267176

Figura 5: Dados normalizados.

De forma sequencial, foi aplicado o classificador MLP, utilizando os mesmos parâmetros do caso anterior.

```
clf = MLPClassifier(hidden_layer_sizes=(64,32), activation='relu',
    solver='lbfgs')
clf.fit(xtrain1, ytrain1)

ypred1 = clf.predict(xtest1)

acuracia1 = accuracy_score(ytest1, ypred1)

print(f'A acuracia do modelo e {acuracia1*100:.3f} %')

cm1 = confusion_matrix(ytest1, ypred1)
```

Listing 5: Implementação da rede.

Neste caso a acurácia do modelo foi de 93,17%. Algo relativamente esperado já que a base trabalha com muito mais valores que a classe wine. A matriz de confusão também foi visualizada, para uma análise visual da eficiência do modelo testado.

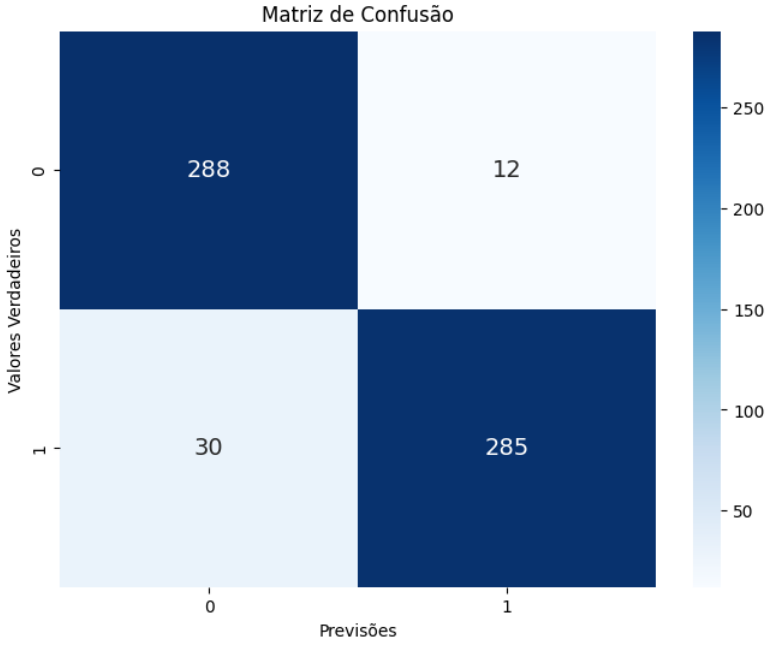

Figura 6: Matriz de confusão.

2.2 Comparação com os demais classificadores

Aqui, serão comparados os resultados obtidos na análise apresentada em relação ao classificadores já utilizados. Todos os casos serão referentes à base de dados wine, já que esta foi mais trabalhada durante o processo de construção dos algoritmos.

2.2.1 Naive Bayes

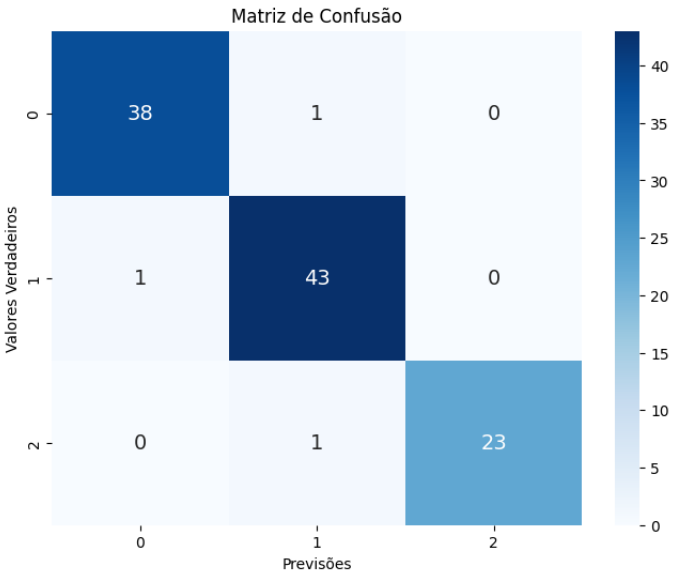

Figura 7: Resultados - Naive Bayes.

2.2.2 Support Vector Machine

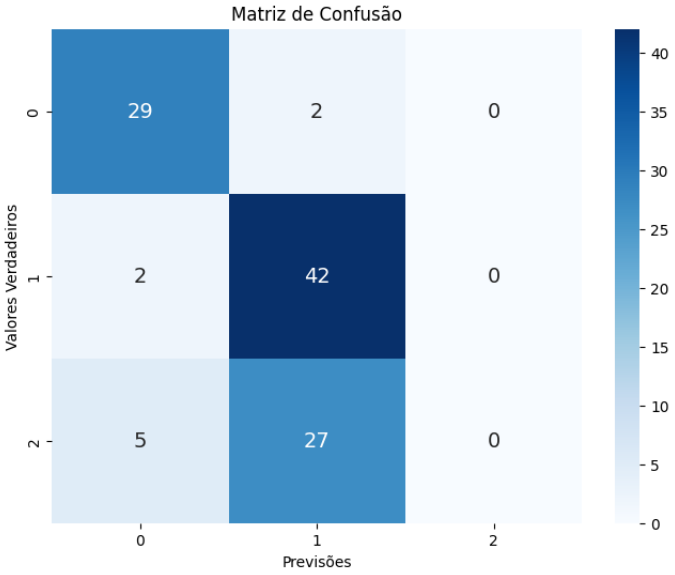

Figura 8: Resultados - SVM.

Observa-se que os classificadores apresentados, possuem resultados bem variados, mesmo utilizando a mesma base de dados e a mesma porcentagem para treinamento (40%). Enquanto o algoritmo de Naive Bayes, implementado usando a scikit-learn possui uma acurácia de 97,19%, o algoritmo SVM possui uma acurácia de 68%.

Essa distinção dos resultados pode ser fruto de diferentes motivos: leves mudanças na implementação, falta de otimização na mesma. Porém, um dos pontos cruciais que vale ressaltar é a escolha do conjunto de dados que é diretamente ligada à eficiência do modelo.

Ao se utilzar a função random permutation da NumPy, a cada novo teste, os valores de treinamento e testes são alterados. Dessa forma, os dados escolhidos puderam possibilitar um melhor desempenho, até semelhante, entre a implementação da MLP e do algoritmo de Bayes, em detrimento do Support Vector Machine.

3 Considerações Finais

No decorrer deste estudo, exploramos a implementação de redes neurais do tipo Multilayer Perceptron (MLP) usando a biblioteca Scikit-Learn e as comparamos com algoritmos estabelecidos, como Naive Bayes e Support Vector Machine (SVM). Uma das observações cruciais foi a versatilidade das redes neurais MLP. Elas se destacam em tarefas complexas devido à sua capacidade de aprender padrões intricados nos dados. No entanto, essa complexidade também exige um ajuste cuidadoso de hiperparâmetros, o que pode ser uma tarefa desafiadora e intensiva em recursos computacionais.

De forma geral, as redes neurais MLP superam Naive Bayes em cenários onde as relações nos dados são intrincadas e não lineares. Isso é especialmente evidente em conjuntos de dados grandes e complexos. Por outro lado, Naive Bayes, devido à sua simplicidade, oferece resultados aceitáveis em tarefas menos complexas e tem a vantagem da interpretabilidade imediata, o que pode ser crucial em algumas aplicações.

Em vista disso, é essencial notar que este estudo não é conclusivo para todas as situações. O desempenho dos algoritmos pode variar significativamente com base na natureza específica do problema e nas características dos dados. Recomenda-se uma análise cuidadosa das características do problema em questão antes de tomar uma decisão sobre qual algoritmo utilizar.

4 Referências

[1] RABELO, Rafael Tolentino. **Arquitetura de hardware dedicada de uma rede neural perceptron para reconhecimento de terreno aplicado a robótica móvel**. 2014. Trabalho de Conclusão de Curso (Bacharel) - Universidade de Brasília - UnB, [S. l.], 2014

[2] WSMALTA. **MLPClassifier - Wine - 1**. Disponível em: <https://www.kaggle.com/code/wsmalta/mlpclassifier-wine-1>. Acesso em: 30 out. 2023.

[3] **Funções de Ativação**. Disponível em: <https://matheusfacure.github.io/2017/07/12/activ-func/>.