Criação de uma biblioteca para Redes Neurais Perceptron com Múltiplas Camadas na linguagem de programação C

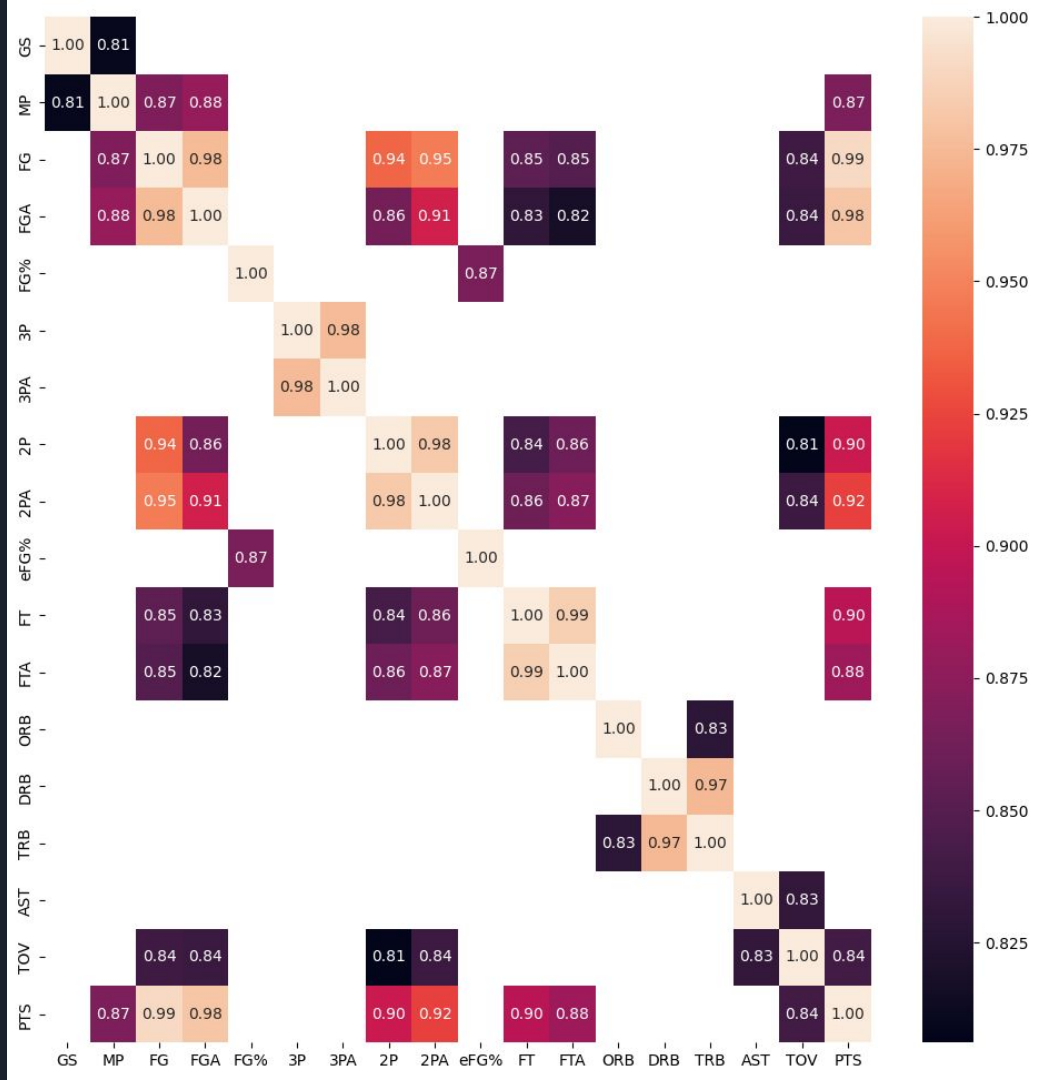
Eduardo Eberhardt Pereira

1. Preparação dos dados

Operações feitas

- Remoção de valores nulos
- Remoção da coluna 'Player' por conter valores únicos
- Remoção da coluna 'FG%.1' por ser só uma variação de 'FG%'
- Remoção das colunas 'GS' (games started tem alta relação com minutes played), 'FG', 'FGA', '2PA', '3PA', 'FTA' (já há a quantidade de acertos e a porcentagem de acerto), 'ORB' e 'DRB' (por serem agrupadas em Total Rebounds)
- Remapeamento da coluna 'Performance': 'Good': 1, 'Bad': 0
- Remapeamento das colunas 'Pos' e 'Tm' com LabelEncoder

Verificação das correlações:



Colunas finais:

```
Pos,Age,Tm,G,MP,FG%,3P,3P%,2P,2P%,eFG%,FT,FT%,TRB,AST,STL,BLK,TOV,PF,PTS,Performance
```

```
df.head()
```

	Pos	Age	Tm	G	MP	FG%	3P	3P%	2P	2P%	...	FT	FT%	TRB	AST	STL	BLK	TOV	PF	PTS	Performance
0	0	23.0	27	38.0	23.3	0.482	0.5	0.260	3.4	0.557	...	1.9	0.689	6.6	1.0	0.6	0.7	1.2	2.1	10.2	0.0
1	0	29.0	14	42.0	27.0	0.597	0.0	0.000	3.7	0.599	...	1.1	0.364	11.5	2.3	0.9	1.1	1.9	2.3	8.6	0.0
2	0	25.0	15	57.0	35.0	0.536	0.0	0.083	8.4	0.543	...	4.4	0.800	9.8	3.2	1.2	0.8	2.5	2.8	21.2	1.0
3	6	22.0	29	39.0	15.6	0.483	0.9	0.396	0.9	0.621	...	0.4	0.682	1.8	0.6	0.2	0.1	0.3	1.4	5.0	0.0
4	1	22.0	14	56.0	22.0	0.474	1.4	0.364	2.0	0.601	...	1.4	0.729	4.6	1.2	0.7	0.7	0.7	1.9	9.4	0.0

5 rows × 21 columns

Funções Utilizadas

- Em seguida, os dados passaram pela codificação Dummy
- Depois, foram divididos em conjuntos de treino e teste, com 70% das instâncias destinadas ao treino e 30% ao teste. A biblioteca oferece dois algoritmos para o embaralhamento, FISCHER-YATES e um próprio do autor, sendo o primeiro mais veloz mas mais simples, e o segundo lento mas eficaz. Neste trabalho, o optado foi o algoritmo próprio do autor.
- Cada conjunto (treino e teste) passou pelo Escalonamento MinMax
- Por fim, cada conjunto (treino e teste) foi separado em x e y

Operações iniciais disponíveis em:

- https://colab.research.google.com/drive/1jkDGuIZ8s95btk99eUv_SBNEIq6tOXVr?authuser=2#scrollTo=06o_peH0UgKN

2. Implementação do modelo

Arquitetura usada

- Visto que a biblioteca criada permite definir n camadas intermediárias e quaisquer valores para números de neurônios em cada camada intermediária/oculta, testes foram feito com diversas arquiteturas. Não foi percebida grande variação nos resultados, mas a arquitetura escolhida foi:
  - 2 camadas intermediárias
    - 20 neurônios
    - 10 neurônios
  - Camada de entrada com 59 neurônios
  - Camada de saída com 2 neurônios (jogador performa bem; jogador performa mal)
- A biblioteca permite também a escolha de diferentes funções de ativação, mas a escolhida foi a sigmoide, e portanto, a função de erro é a derivada da sigmoide.

3. Treinamento do modelo

Processo do Treinamento

- Os algoritmos de feedforward e backpropagation foram utilizados, mas mascarados ao usuário da biblioteca para facilitar seu uso. Para treinar a rede, basta o usuário usar a função train ou train_with_early_stopping (ou as macros fit e fit_with_early_stopping), e a função se encarrega de passar por cada as instância n vezes (de acordo com o número de épocas), rodando os carregando os dados daquela instância na camada de entrada, aplicando o feedforward, e corrigindo os pesos com algoritmo de backpropagation.
- A biblioteca usa também o algoritmo de MSE (Mean Squared Error) para definir como o treinamento da rede está progredindo, e assim, aplicar e verificar a descida gradiente.
- Todas essas funções podem ser encontradas no código anexado e são explicadas em detalhes no arquivo “readme.md”.

Processo do Treinamento

- Após testes com diferentes hiperparâmetros, os escolhidos foram:
  - Momentum 1
  - Learning rate foi 0.2
  - 1000 épocas mas a função usada (fit_with_early_stopping) a encerra sempre que a diferença entre dez épocas (paciência escolhida) não passa de 0.0001.

4. Avaliação e ajustes

Métricas

- As métricas de avaliação implementadas para cada classe foram: Precisão, Recall e F-Score.
- A acurácia também é calculada, mas para o modelo todo.
- A biblioteca fornece uma struct chamada metrics que guarda a matriz de confusão e outras informações sobre as métricas alcançadas após os testes.
- Devido a questões da implementação da biblioteca, hiperparâmetros como random_state fazem com que haja uma pequena variação nas métricas em cada treino. Como não há um random_state que seja o melhor para todos os conjuntos de dados, para os testes apresentados a seguir, foram usados random_states aleatórios (time(NULL)) e o resultado apresentado é o de maior acurácia encontrado (levando boas precisões em consideração).
- Como mencionado anteriormente, todos os conjuntos passaram por tratamento dos dados (escalonamento, codificação dummy, etc) antes dos testes serem realizados.

Métricas

- O primeiro dataset testado foi o apresentado anteriormente, com as seguintes colunas:

```
Pos,Age,Tm,G,MP,FG%,3P,3P%,2P,2P%,eFG%,FT,FT%,TRB,AST,STL,BLK,TOV,PF,PTS,Performance
```

- As métricas obtidas foram (para a classe 0):
- Precisao: 98.67
- Sensitividade: 99.33
- Especificidade: 87.5
- Acurácia: 98.19
- Erro: 1.81
- TPR = 99.33%; FPR = 12.5%

```
Resultados obtidos:
+-----------------------------------------------------
                    Classe 0
| Previsoes esperadas para classe 0: 150
| Previsoes para 0: 149
| Previsoes para 1: 1
| Precisao: 98.675497
| Recall: 99.333333
| F-1 Score: 99.003322
+-----------------------------------------------------

+-----------------------------------------------------
                    Classe 1
| Previsoes esperadas para classe 1: 16
| Previsoes para 0: 2
| Previsoes para 1: 14
| Precisao: 93.333333
| Recall: 87.500000
| F-1 Score: 90.322581
+-----------------------------------------------------

| Acuracia: 163 / 166 = 98.192771
```

Métricas

- O segundo dataset testado foi uma versão sem as colunas 2PT e 3PT, deixando apenas suas versões em porcentagem para testar se elas bastariam:

```
Pos,Age,Tm,G,MP,FG%,3P%,2P%,eFG%,FT,FT%,TRB,AST,STL,BLK,TOV,PF,PTS,Performance
```

- As métricas obtidas foram (para a classe 0):
- Precisao: 98.01
- Sensitividade: 98.67
- Especificidade: 81.25
- Acurácia: 96.98
- Erro: 3.02
- TPR = 98.67%; FPR = 18.75%

```
Resultados obtidos:
+-----------------------------------------------------
                    Classe 0
| Previsoes esperadas para classe 0: 150
| Previsoes para 0: 148
| Previsoes para 1: 2
| Precisao: 98.013245
| Recall: 98.666667
| F-1 Score: 98.338870
+-----------------------------------------------------

+-----------------------------------------------------
                    Classe 1
| Previsoes esperadas para classe 1: 16
| Previsoes para 0: 3
| Previsoes para 1: 13
| Precisao: 86.666667
| Recall: 81.250000
| F-1 Score: 83.870968
+-----------------------------------------------------

| Acuracia: 161 / 166 = 96.987952
```

Métricas

- O terceiro dataset testado não continha a coluna G, que representa o número de jogos jogados. Como jogadores bons que se contundem jogam menos partidas, é possível que o modelo confunda esses jogadores como ruins. Para esse teste, as colunas usadas foram:

```
Pos,Age,Tm,MP,FG%,3P,3P%,2P,2P%,eFG%,FT,FT%,TRB,AST,STL,BLK,TOV,PF,PTS,Performance
```

- As métricas obtidas foram (para a classe 0):
- Precisao: 99.33
- Sensitividade: 99.33
- Especificidade: 93.75
- Acurácia: 98.79
- Erro: 1.21
- TPR = 99.33%; FPR = 6.25%

```
Resultados obtidos:
+----------------------------------------------------------
                    Classe 0
| Previsoes esperadas para classe 0: 150
| Previsoes para 0: 149
| Previsoes para 1: 1
| Precisao: 99.333333
| Recall: 99.333333
| F-1 Score: 99.333333
+----------------------------------------------------------

+----------------------------------------------------------
                    Classe 1
| Previsoes esperadas para classe 1: 16
| Previsoes para 0: 1
| Previsoes para 1: 15
| Precisao: 93.750000
| Recall: 93.750000
| F-1 Score: 93.750000
+----------------------------------------------------------

| Acuracia: 164 / 166 = 98.795181
```

Métricas

- O último dataset testado não utilizava as colunas Tm (time) e Pos (posição), visando reduzir o número de colunas dummy geradas e ver se, em uma estrutura mais simples, conseguia resultados melhores:

```
Pos,Age,Tm,MP,FG%,3P,3P%,2P,2P%,eFG%,FT,FT%,TRB,AST,STL,BLK,TOV,PF,PTS,Performance
```

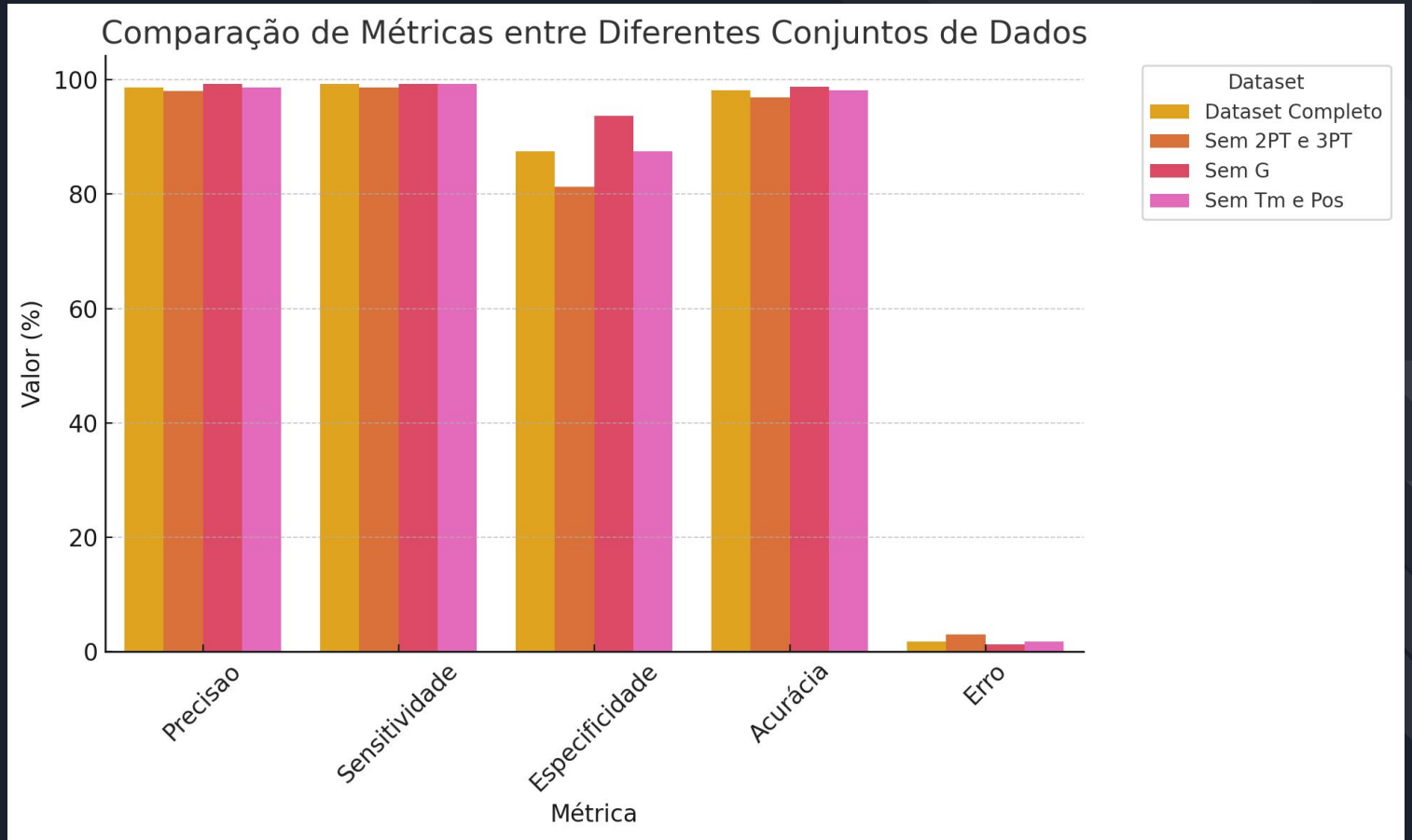
- As métricas obtidas foram (para a classe 0):
- Precisao: 98.67%
- Sensitividade: 99.33%
- Especificidade: 87.5%
- Acurácia: 98.19%
- Erro: 1.81%
- TPR = 99.33%; FPR = 12.5%

```
Resultados obtidos:
+-----------------------------------------------------
                    Classe 0
| Previsoes esperadas para classe 0: 150
| Previsoes para 0: 149
| Previsoes para 1: 1
| Precisao: 98.675497
| Recall: 99.333333
| F-1 Score: 99.003322
+-----------------------------------------------------

+-----------------------------------------------------
                    Classe 1
| Previsoes esperadas para classe 1: 16
| Previsoes para 0: 2
| Previsoes para 1: 14
| Precisao: 93.333333
| Recall: 87.500000
| F-1 Score: 90.322581
+-----------------------------------------------------

| Acuracia: 163 / 166 = 98.192771
```

Tabela Comparativa dos Resultados

	Precisao	Sensitividade	Especificidade	Acurácia	Erro
Dataset Completo	98.67	99.33	87.5	98.19	1.81
Sem 2PT e 3PT	98.01	98.67	81.25	96.98	3.02
Sem G	99.33	99.33	93.75	98.79	1.21
Sem Tm e Pos	98.67	99.33	87.5	98.19	1.81

Gráfico Comparativo



5. Interpretação e Conclusão

Conclusões

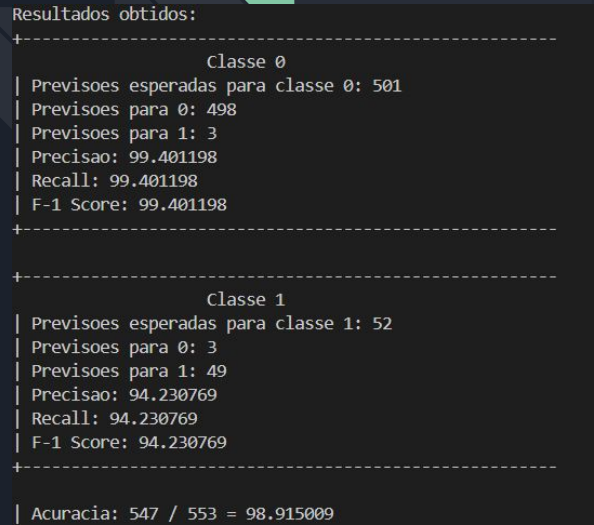
- Como foi possível ver pelas métricas obtidas, os modelos todos obtiveram resultados excelentes e muito satisfatórios. O modelo aprendeu bem a diferenciar jogadores com bom e mal desempenho mesmo com o desbalanceamento do dataset. Ele também se demonstrou capaz de manter excelente qualidade com diferentes variações, demonstrando que o aprendizado foi ótimo.
- Como as métricas utilizadas são do melhor caso, não é demonstrado, por exemplo, que o conjunto original e sem os jogos possuem a média das taxas de acerto muito próximas ao máximo, enquanto os demais datasets possuem variância de até 3% na taxa de acerto (acurácia).
- O melhor dataset foi o sem a coluna de quantidade de partidas jogadas, o que comprovou a teoria de que bons jogadores, quando lesionados, perdem partidas e atrapalham, mesmo que minimamente, a predição do modelo.

Conclusões

- Ainda, o núcleo principal de colunas, que envolve a quantidade de pontos marcados (PTS), quantas cestas de cada tipo são marcadas (FG%, 3P, 3P%, 2P, 2P%, eFG%, FT, FT%), a precisão no acerto de cada tipo de cesta, posição, time (Tm), minutos jogados (MP), faltas (PF), perdas de bola (TO), assistências (AST), rebotes (TRB) tocos (BLK), roubos (STL), idade (Age), posição (Pos) e a coluna alvo (Performance), se demonstrou muito eficaz em verificar a performance dos jogadores.
- Dada a incrível performance do modelo, é difícil de pensar em melhorias para o dataset atual. A biblioteca, no entanto, poderia buscar usar algoritmo mais determinísticos para evitar tanta variação, além de disponibilizar mais ferramentas genéricas para treinamento da rede e visualização dos dados. Além disso, poderia fornecer mais opções de algoritmos para serem usados e testados.

Conclusões

- Como a biblioteca permite a exportação da rede neural - isto é, da arquitetura e da matriz de pesos, é possível carregar o conjunto de dados completo em sequência e testar os resultados juntando os conjuntos de treino e teste, verificando se a precisão se mantém. Esse teste foi realizado apenas para o dataset sem a coluna G, visto que ele obteve a melhor performance. Os resultados obtidos então foram:
  - Precisao: 98.91%
  - Sensitividade: 99.4%
  - Especificidade: 94.23%
  - Acurácia: 98.91%
  - Erro: 1.09%
  - TPR = 99.4%; FPR = 5.37%



Matriz de Confusão Para Todos os dados