Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Relatório: Comparação entre Modelos Perceptron e MultiLayerPerceptron (MLP)

GitHub: https://github.com/luisggf/ai-tests
Aluno(s): Luis Guilherme (21.2.8007)
Gabriela Aranda (19.1.8136)
Tobias Ferraz (21.1.8016)

Fevereiro
2024

Conteúdo

1 Resumo

Este relatório descreve o segundo trabalho prático da disciplina de Inteligência Artificial, focado na resolução de um problema de classificação de dados por meio da implementação de dois modelos de redes neurais: um Perceptron Simples e uma Rede MLP com uma e duas camadas ocultas.

Iniciamos abordando a escolha da base de dados *Adult*, detalhando suas características e destacando aspectos como o desbalanceamento na distribuição da variável alvo *income*. Para lidar com esse desafio, aplicamos técnicas de tratamento de dados, removendo valores inválidos, convertendo colunas categóricas e normalizando os dados.

A implementação da Rede MLP com duas camadas ocultas foi discutida e documentada, destacando as modificações realizadas no código original. Além disso, detalhamos a etapa de treinamento do modelo, evidenciando a necessidade de ajustes nos métodos *fit* e *predicao*.

A seção de análise exploratória revelou observações importantes sobre a distribuição dos dados, evidenciando desequilíbrios em colunas como *race* e *workclass*. As análises também destacaram características específicas das variáveis, como a predominância de valores zero em *capital_gain* e *capital_loss*.

Por fim, a seção de estatísticas gerais proporcionou uma visão abrangente das colunas da base de dados, incluindo histogramas e observações específicas sobre características como raça, classe trabalhadora e país de origem.

O trabalho prossegue com a análise e ajustes necessários nos modelos implementados, abordando desafios identificados durante a exploração e tratamento dos dados.

2 Base de dados escolhida

A base de dados selecionada para este estudo é conhecida como *Adult*, também referenciada como *Census Income*. Esta base contém informações demográficas sobre indivíduos, tais como idade, educação, estado civil, ocupação, relação, raça, sexo, ganho de capital, perda de capital, horas trabalhadas por semana e país de origem. A variável alvo é o nível de renda, categorizado como »50K"ou «=50K,"indicando se um indivíduo ganha mais de $50,000.00 anualmente.

Tabela 1: Distribuição percentual dos valores da coluna 'Income'.

$\leqslant 50k$	$> 50K$
75.92%	24.08%

Tabela 2: Estatísticas gerais das colunas númericas

	Age	**Capital-Gain**	**Capital-Loss**	**Hours-Per-Week**	**Final_Weight**
Tipo	Inteiro	Inteiro	Inteiro	Inteiro	Inteiro
Variância	174.71	56346492.60	163989.43	144.18	11159639548.60
Valor Máximo	90	99999	4356	99	1490400
Valor Mínimo	17	0	0	1	13492
Média	38.54	1101.43	88.59	40.93	189734.73

2.1 Estatisticas gerais da base de dados

A base de dados utilizada neste estudo possui 48.842 registros e 15 colunas, sendo 10 categóricas [1] (tipo de trabalho, nível de educação, raça, etc.) e 5 nominais [2].

3.620 valores inválidos (NaNs[3], "?", NULL...) foram encontrados na base de dados. A coluna alvo ("y"), que indica a distribuição de renda, apresenta um desbalanceamento (detalhes na tabela 1).

2.2 Estatísticas Gerais das Colunas da Base de Dados

Nesta subseção, exploraremos o comportamento dos dados, apresentando estatísticas como médias, variâncias e histogramas. Na Tabela 1, observamos a distribuição dos valores binários da variável *income*, que exibe uma distribuição bastante desbalanceada. Esse desafio será tratado posteriormente na Seção 3, uma vez que o desbalanceamento pode gerar anomalias no aprendizado do modelo e afetar os resultados. Por fim, a análise das distribuições das colunas é ilustrada nos conjuntos de imagens abaixo.

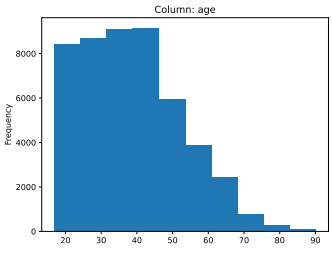
2.3 Observações sobre Distribuições

Ao analisar as distribuições das colunas, algumas observações relevantes foram identificadas. A distribuição da raça (*race*) revela uma quantidade sig-

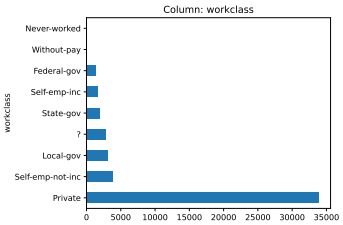
[1] **Colunas Categóricas:** Representam variáveis qualitativas, como *workclass*, *education* e *marital-status* que possuem categorias distintas.

[2] **Colunas Nominais:** São aquelas que representam dados sem uma ordem específica, como as colunas categóricas mencionadas acima.

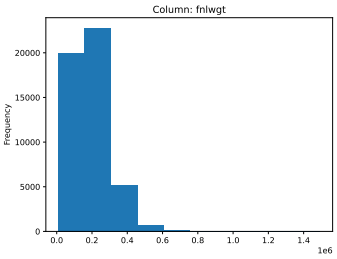
[3] **NaN:** Em computação, NaN é um valor ou símbolo usado nas linguagens de programação para representar um valor numérico indefinido ou irrepresentável



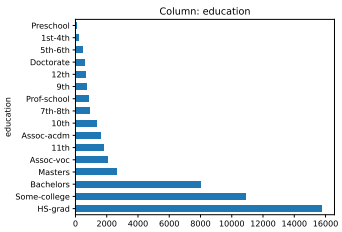
(a) Distribuição de "Age".



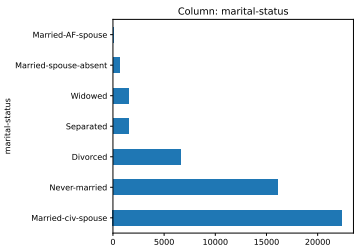
(b) Distribuição de "Workclass".



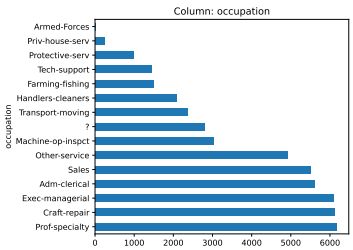
(c) Distribuição de "Final Weight".



(d) Distribuição de "Education".



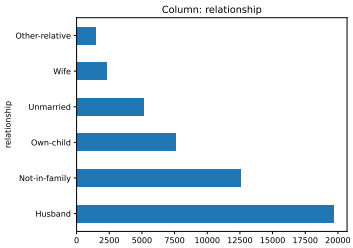
(e) Distribuição de "Marital Status".



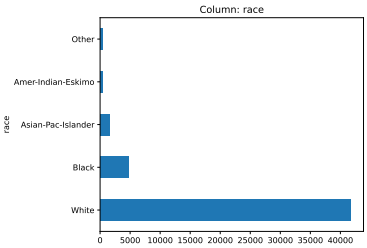
(f) Distribuição de "Occupation".

nificativa de valores associados a *White*, indicando um possível desequilíbrio racial na base de dados. Quanto à classe trabalhadora (*workclass*), há uma presença expressiva de valores atribuídos a *Private*, sugerindo que a maioria dos indivíduos pertence ao setor privado. Além disso, a variável referente ao país de origem (*native-country*) está fortemente inclinada para os Estados Unidos (*United-States*), sugerindo um viés geográfico. Uma alternativa seria converter essa variável em uma binária, como *from_united_states*, para reduzir esse viés.

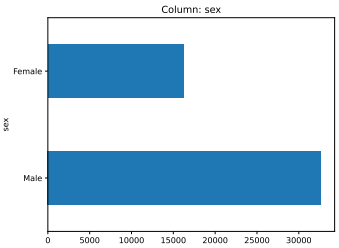
Observações mais específicas incluem a distribuição de horas trabalhadas por semana (*hours_per_week*), que apresenta um pico em 40 horas, refletindo a jornada de trabalho típica. As variáveis de ganhos e perdas de capital (*capital_gain* e *capital_loss*) mostram um pico em zero, indicando que a maioria dos indivíduos não possui ganhos ou perdas de capital adicionais. Considerando essa característica, uma abordagem eficiente seria analisar a distribuição excluindo os valores zero, pois podem ser considerados casos atípicos. Além disso, a variável relacionada ao número de anos de educação (*education_num*) exibe dois picos, sugerindo uma distribuição bimodal relacionada aos diferentes níveis educacionais na população.



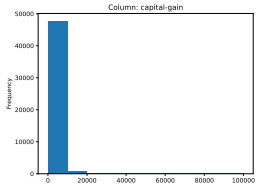
(g) Distribuição de "Relationship".



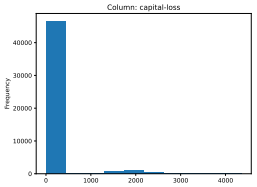
(h) Distribuição de "Race".



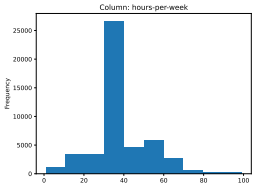
(i) Distribuição de "Sex".



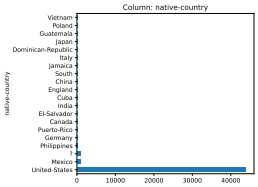
(j) Distribuição de "Capital Gain".



(k) Distribuição de "Capital Loss".



(l) Distribuição de "Hours per Week".



(m) Distribuição de "Native Country".

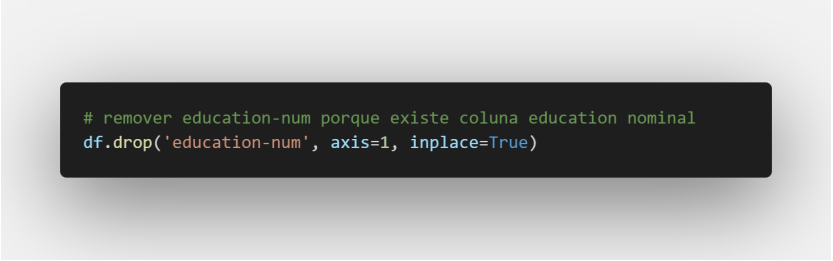
3 Tratamento de Dados

O tratamento de dados é uma etapa crucial na preparação do conjunto de dados para treinar os modelos de Machine Learning. Nesta seção, descreveremos detalhadamente as etapas realizadas para limpar e preparar o conjunto de dados *Adult*.

- **Remoção de dados inválidos:** Inicialmente, identificamos e removemos linhas que continham valores inválidos, representados por entradas marcadas como '?'. Essa abordagem de remoção é adotada literalmente, ou seja, as linhas inteiras com valores inválidos são descartadas do conjunto de dados.

```
# remover valores nans
df = df.replace('?', np.nan)
print("Quantidade de valores NULL: ", df[pd.isnull(df).any(axis=1
)].shape)
df.dropna(inplace=True)
```

- **Remoção da coluna *education-num*:** Observamos que a coluna *education-num* é equivalente à coluna *education* nominal. Portanto, optamos por remover a coluna *education-num*, mantendo apenas a informação nominal correspondente.



```
# remover education-num porque existe coluna education nominal
df.drop('education-num', axis=1, inplace=True)
```

- **Conversão de colunas categóricas para valores numéricos:** Utilizamos a função *convert_to_int* para mapear as colunas categóricas, como *workclass*, *education*, *marital-status*, *occupation*, *relationship*, *race*, *sex* e *native-country*, para valores numéricos. Isso é crucial para permitir que os modelos compreendam e processem essas informações.

- **Normalização das colunas entre 0 e 1:** Normalizamos [4] as colunas *age*, *fnlwgt*, *capital-gain*, *capital-loss* e *hours-per-week* para a faixa entre 0 e 1. Essa normalização facilita o cálculo de pesos durante o treinamento das redes neurais, contribuindo para a convergência eficiente do modelo.

[4] **Normalização:** Processo que visa ajustar as escalas das colunas, garantindo que elas estejam na mesma faixa de valores. Isso facilita o treinamento eficiente dos modelos de Machine Learning.

```
def normalize(columns):
    scaler = preprocessing.StandardScaler()
    df[columns] = scaler.fit_transform(df[columns])

def show_values(columns):
    for column in columns:
        max_val = df[column].max()
        min_val = df[column].min()
        mean_val = df[column].mean()
        var_val = df[column].var()
        print(column + ': values=['+str(min_val)+','+str(max_val
) +
              '] , mean='+str(mean_val)+' , var='+str(var_val))

# valores antes e pos normalização
show_values(normalize_columns)
normalize(normalize_columns)
# pos normalização
show_values(normalize_columns)
```

- **Conversão de booleanos para inteiros:** Algumas colunas contêm valores booleanos, e para garantir consistência, convertemos esses valores para inteiros, mantendo a integridade das informações.

```
def convert_to_int(columns):
    for column in columns:
        unique_values = df[column].unique().tolist()
        dic = {}
        for indx, val in enumerate(unique_values):
            dic[val] = indx
        df[column] = df[column].map(dic).astype(int)
        print(column + " done!")

# como função convert_to_int alterou coluna salario
print("Coluna salário antes: ", df['salary'])
# converter coluna salario em valores binarios
convert_to_int(label_column)
print("Coluna salário depois: ", df['salary'])
```

- **One-Hot Encoding:** Utilizamos a técnica de *One-Hot Encoding*[5] para

[5]**One-Hot Encoding:** Técnica utilizada para converter colunas categóricas em

transformar colunas categóricas em múltiplas colunas binárias, mantendo assim a representação adequada das informações.

```python
def convert_to_onehot(data, columns):
    dummies = pd.get_dummies(data[columns])
    data = data.drop(columns, axis=1)
    data = pd.concat([data, dummies], axis=1)
    return data
# converter colunas categoricas em valores numericos
df = convert_to_onehot(df, categorical_columns)
```

- **Visualização de estatísticas antes e após a normalização:** Utilizamos a função *show_values* para exibir estatísticas descritivas das colunas antes e após a normalização. Essa etapa é crucial para verificar se as colunas estão na mesma escala e se a normalização foi eficiente. No geral, tal análise é feita em nosso tratamento sempre após realizar algum tipo de processamento sobre a base, com a finalidade de observar se resultados são condizentes com o que se espera da filtragem.

- **Separar o conjunto de dados em classes bem definidas:** Dividimos o conjunto de dados em características (X) e rótulos (y), facilitando a posterior divisão entre conjuntos de treinamento e teste.

```python
# separar dataframe em 2 classes bem definidas
y_labels = df['salary']
x_data = df.drop('salary', axis=1)
```

múltiplas colunas binárias, onde cada categoria possui sua própria coluna com valores binários indicando a presença ou ausência da categoria.

4 Implementação de MLP com Duas Camadas Ocultas

Nesta seção, discutimos os detalhes de implementação do modelo Perceptron Multicamadas (MLP) com duas camadas ocultas, baseando-se na arquitetura original com uma única camada oculta. As modificações incluem a adição de uma segunda camada oculta, aumentando a capacidade do modelo de capturar padrões complexos nos dados de entrada.

4.1 Inicialização (função __init__)

```
class MLP_2LAYERS:
    def __init__(self, dim_entrada, dim_ocultos1, dim_ocultos2,
dim_saida, taxa_aprendizado=0.01):
        self.dim_entrada = dim_entrada
        self.dim_ocultos1 = dim_ocultos1
        self.dim_ocultos2 = dim_ocultos2
        self.dim_saida = dim_saida
        self.taxa_aprendizado = taxa_aprendizado


# Inicialização dos pesos e bias das camadas ocultos1, ocultos2,
e de saída
        self.pesos_ocultos1 = np.random.rand(
            self.dim_entrada, self.dim_ocultos1)
        self.bias_ocultos1 = np.zeros(self.dim_ocultos1)

        self.pesos_ocultos2 = np.random.rand(
            self.dim_ocultos1, self.dim_ocultos2)
        self.bias_ocultos2 = np.zeros(self.dim_ocultos2)

        self.pesos_saida = np.random.rand(self.dim_ocultos2, self.
dim_saida)
        self.bias_saida = np.zeros(self.dim_saida)
```

Figura 1: Trecho de código ilustrando o processo de inicialização com duas camadas ocultas.

O processo de inicialização foi estendido para acomodar a segunda camada oculta. Um novo parâmetro, `dim_oculta2`, foi introduzido para representar o número de neurônios na segunda camada oculta. Além disso, pesos

(`pesos_ocultos2`) e vieses (`bias_ocultos2`) para a segunda camada oculta foram inicializados.

4.2 Feedforward

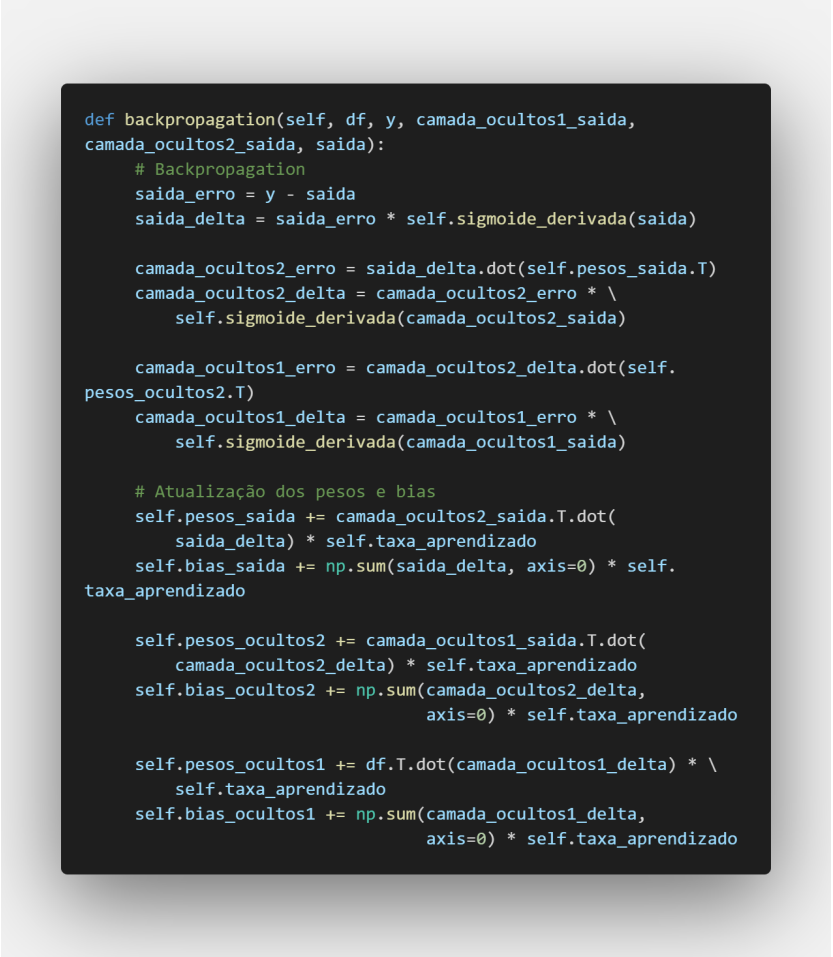
```
def feedforward(self, df):
    # Calculo da saida da 1ª Camada oculta
    camada_ocultos1_entrada = np.dot(
        df, self.pesos_ocultos1) + self.bias_ocultos1
    camada_ocultos1_saida = self.sigmoide(
camada_ocultos1_entrada)

    # Calculo da saida da 2ª Camada oculta
    camada_ocultos2_entrada = np.dot(
        camada_ocultos1_saida, self.pesos_ocultos2) + self.
bias_ocultos2
    camada_ocultos2_saida = self.sigmoide(
camada_ocultos2_entrada)

    # Calculo da saida da Camada de saída
    camada_saida_entrada = np.dot(
        camada_ocultos2_saida, self.pesos_saida) + self.
bias_saida
    saida = self.sigmoide(camada_saida_entrada)

    return camada_ocultos1_saida, camada_ocultos2_saida, saida
```

Figura 2: Trecho de código ilustrando o processo de feedforward com duas camadas ocultas.

O processo de feedforward mantém o cálculo da saída da primeira camada oculta usando a função de ativação sigmoidal. Um novo cálculo foi introduzido para a saída da segunda camada oculta, empregando a função de ativação sigmoidal. O cálculo da saída para a camada final permanece inalterado.

4.3 Backpropagation

O processo de retropropagação foi estendido para considerar a segunda camada oculta. O cálculo de erro e delta para a camada de saída permanece inalterado. O erro é então propagado para trás, e os pesos e vieses para a segunda camada oculta são ajustados usando a derivada da função sigmoidal.



```
def backpropagation(self, df, y, camada_ocultos1_saida,
camada_ocultos2_saida, saida):
    # Backpropagation
    saida_erro = y - saida
    saida_delta = saida_erro * self.sigmoide_derivada(saida)

    camada_ocultos2_erro = saida_delta.dot(self.pesos_saida.T)
    camada_ocultos2_delta = camada_ocultos2_erro * \
        self.sigmoide_derivada(camada_ocultos2_saida)

    camada_ocultos1_erro = camada_ocultos2_delta.dot(self.
pesos_ocultos2.T)
    camada_ocultos1_delta = camada_ocultos1_erro * \
        self.sigmoide_derivada(camada_ocultos1_saida)

    # Atualização dos pesos e bias
    self.pesos_saida += camada_ocultos2_saida.T.dot(
        saida_delta) * self.taxa_aprendizado
    self.bias_saida += np.sum(saida_delta, axis=0) * self.
taxa_aprendizado

    self.pesos_ocultos2 += camada_ocultos1_saida.T.dot(
        camada_ocultos2_delta) * self.taxa_aprendizado
    self.bias_ocultos2 += np.sum(camada_ocultos2_delta,
                                 axis=0) * self.taxa_aprendizado

    self.pesos_ocultos1 += df.T.dot(camada_ocultos1_delta) * \
        self.taxa_aprendizado
    self.bias_ocultos1 += np.sum(camada_ocultos1_delta,
                                 axis=0) * self.taxa_aprendizado
```

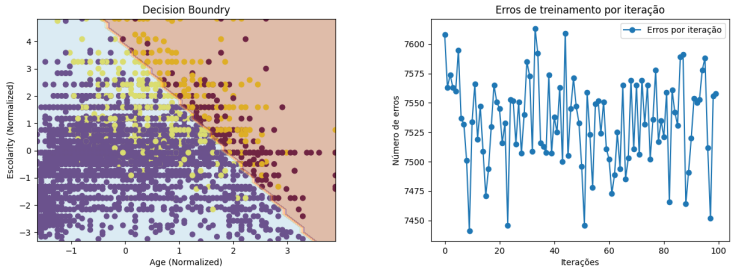
Figura 3: Trecho de código ilustrando o processo de retropropagação com duas camadas ocultas.

O erro é propagado ainda mais para trás, e os pesos e vieses para a primeira camada oculta são ajustados de maneira semelhante.

Essas melhorias permitem que o modelo incorpore duas camadas ocultas, proporcionando uma maior capacidade para aprender representações intricadas dos dados de entrada. Os ajustes garantem a utilização efetiva de ambas as camadas ocultas durante os passes para frente e para trás.

5 Análise dos Resultados

Nesta seção, será realizada uma análise dos resultados obtidos a partir da execução de três modelos de aprendizado de máquina: Perceptron, MLP

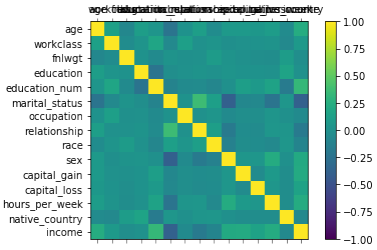


(a) Borda de Decisão do Perceptron.

(b) Erro em função das Épocas (Perceptron).

Figura 4: Análise dos Resultados: Perceptron.

Figura 5: Heatmap dos atributos da base de dados.



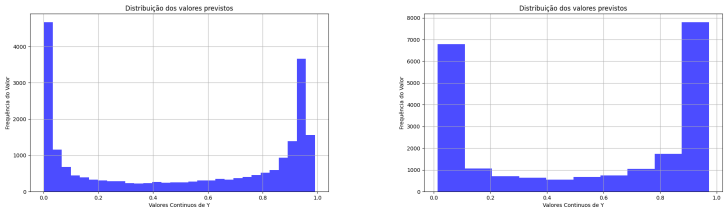
(Multilayer Perceptron) e MLP com 2 camadas ocultas.

O gráfico 5 apresenta a borda de decisão do Perceptron, revelando sua habilidade em separar os conjuntos de dados de maneira razoável. Vale ressaltar que a seleção dos atributos para a plotagem da borda de decisão foi realizada com base no heatmap de correlação (Figura **??**), priorizando aqueles com alta correlação, a fim de destacar padrões significativos.

O gráfico 4(b) proporciona uma visão dinâmica, exibindo a evolução do erro ao longo das épocas durante o treinamento do Perceptron. Ambos os aspectos visuais são fundamentais para a compreensão do desempenho e aprendizado do modelo Perceptron, oferecendo insights valiosos sobre seu comportamento em relação aos dados estudados.

Expandindo a análise para os modelos MLP, os gráficos na Figura 6 apresentam os resultados após a execução do MLP (Figura 6(a)) e do MLP com 2 camadas ocultas (Figura 6(b)).

A Figura 6(a) evidencia os resultados do MLP, enquanto a Figura 6(b) apresenta os resultados do MLP com 2 camadas ocultas. Estas representações gráficas permitem uma comparação visual imediata entre os modelos, destacando padrões e tendências nos dados.



(a) Resultados após execução do MLP.

(b) Resultados após execução do MLP com 2 camadas ocultas.

Figura 6: Análise dos Resultados: MLP e MLP com 2 camadas ocultas.

Para uma análise quantitativa mais aprofundada, consideramos métricas específicas, tais como precisão, verdadeiros positivos e negativos, falsos positivos e falsos negativos. Os resultados após 15 testes são resumidos abaixo. Vale ressaltar, que foram utilizados 100 épocas, uma taxa de aprendizagem de 0.05 e a semente 42 no momento de separação dos conjuntos de teste e treino, para facilitar a reprodução dos testes feitos como parâmetros para todos os modelos:

- **MLP 2 Camadas:** Precisão: 81.79 (Dimensão Oculta 1 = 5 e Dimensão Oculta 2 = 8)
  - Verdadeiros Positivos: 9064
  - Verdadeiros Negativos: 8327
  - Falsos Positivos: 2545
  - Falsos Negativos: 1808
- **MLP:** Precisão: 80.12 (Dimensão Oculta 1 = 12)
  - Verdadeiros Positivos: 8114
  - Verdadeiros Negativos: 9101
  - Falsos Positivos: 1771
  - Falsos Negativos: 2758
- **Perceptron:** Precisão: 79.84
  - Verdadeiros Positivos: 879
  - Verdadeiros Negativos: 6422
  - Falsos Positivos: 323

– Falsos Negativos: 1421

Sendo assim, pode-se perceber que o modelo mais preparado para classificação de dados no contexto do nosso problema é o MLP de 2 Camadas Ocultas com os parâmetros especificadamente atribuidos com (5 e 8 camadas dentro das camadas ocultas) respectivamente. Por fim, este conjunto de análises visa proporcionar esclarecimentos visuais sobre o desempenho relativo dos modelos citados, permitindo conclusões informadas sobre suas capacidades e limitações.

6 Dificuldades

Ao longo da condução desta análise, nos deparamos com alguns desafios que demandaram reflexão e estratégias específicas para superação. Uma das áreas sensíveis foi a manipulação eficiente dos dados, levantando questões sobre como abordar esse processo de maneira eficaz. A tipagem fraca do Python apresentou-se como um ponto crítico, frequentemente gerando erros que exigiam uma abordagem cuidadosa de debugging para identificação e correção.

Outro ponto de desafio concentrou-se na análise da base de dados, exigindo uma exploração meticulosa para extrair informações relevantes de forma eficiente. A consideração constante sobre como analisar o conjunto de dados de maneira mais eficaz permeou o desenvolvimento da análise, envolvendo a aplicação de métodos estatísticos e técnicas exploratórias.

Um desafio peculiar surgiu ao observarmos o comportamento do modelo em relação à quantidade de dados de treinamento. Notou-se que, ao treinar o modelo com mais de 3000 linhas de dados, os valores convergiam, resultando em uma previsão incorreta, onde o modelo passava a categorizar erroneamente apenas uma classe. A resolução desse desafio demandou uma revisão cuidadosa dos parâmetros de treinamento e adaptações estratégicas para garantir a estabilidade e precisão do modelo.

Esses desafios, embora representativos, foram enfrentados com sucesso ao longo da análise, destacando a importância de uma abordagem sistemática e flexível diante das complexidades inerentes a conjuntos de dados e algoritmos de aprendizado de máquina. Essa seção busca oferecer uma perspectiva mais autêntica e detalhada sobre os obstáculos encontrados durante o desenvolvimento da análise.

7 Conclusão

Em suma, este trabalho evidenciou que a aplicação da inteligência artificial não apenas pode, mas deve ser encarada como um facilitador fundamental para a compreensão do mundo ao nosso redor. Ao abordar referências regionais, sociais, raciais e de gênero, percebemos que os padrões identificados neste estudo podem ser extrapolados para diversos outros contextos globais.

A capacidade da inteligência artificial em fornecer níveis significativos de previsão, fundamentados nos padrões mencionados, destaca-se como uma ferramenta valiosa. A análise dos modelos Perceptron, MLP e MLP com 2 camadas ocultas ofereceu insights profundos sobre seu desempenho em um cenário específico, ampliando o entendimento sobre suas capacidades e limitações.

Além disso, as dificuldades enfrentadas durante o processo, desde a manipulação eficiente dos dados até a adaptação dos modelos para situações específicas, reforçam a necessidade de uma abordagem flexível e estratégica ao lidar com problemas complexos de aprendizado de máquina.

Assim, concluímos que a inteligência artificial não apenas desempenha um papel crucial na interpretação e análise de dados, mas também oferece uma perspectiva valiosa para a compreensão mais profunda de questões sociais e regionais. Este trabalho serve como um testemunho da capacidade da inteligência artificial em contribuir de maneira significativa para a exploração e compreensão dos intricados padrões que moldam nosso mundo.

Bibliografia

BISHOP, C. M. Pattern Recognition and Machine Learning. Springer, 2006.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. An Introduction to Statistical Learning. Springer, 2013.

MÜLLER, A. C.; GUIDO, S. Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, 2016.

RASCHKA, S.; MIRJALILI, V. Python Machine Learning. Packt Publishing, 2017.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep Learning. MIT Press, 2016.

GÉRON, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, 2019.

HARRIS, C. R.; MILLMAN, K. J.; VAN DER WALT, S. J.; GOURSKIPY, D.; VIJAYKUMAR, S.; LORCHER, D. F.; JACOBSEN, M. J.; DOLENKO, B.; ZHAO, H.; others. Array programming with NumPy. Nature, 2020.