# MLP Classifier - Classificação do vidro

Milena Carneiro Rios de Oliveira[1]

[1] *Universidade Federal do Tocantins, Ciência da Computação, Tocantins, Brasil*

Reception date of the manuscript:
Acceptance date of the manuscript:
Publication date:

**Abstract**— Como se sabe, redes neurais são uma poderosa ferramenta técnologica, que se baseia no funcionamento do cérebro humano, uma vez que são compostas por neurônios artificiais conectados ponderadamente. Todavia, ao considerar as redes neurais de uma só camada limitam o uso para dados não linearmente separáveis e, isso costuma-se ser um problema, uma vez que no mundo real, majoritariamente dos dados são linearmente separáveis. Dessa forma, uma única camada implica em padrões de entrada similares que irão resultar em padrões de saída também similares, levando à incapacidade do sistema de mapear de forma eficaz. Assim, surge a necessidade uma evolução quanto à essa área do aprendizado profundo e, por isso, as redes do tipo Multilayer Perceptron ou apenas Multi-camadas, sendo esse o mais utilizado no cotidiano. Diante disso, esse trabalho consiste em desenvolver um estudo de caso sobre a aplicabilidade de redes neurais MLP para a classificação de um conjunto tradicional, o Glass Dataset.

**Keywords**—Deep Learning, Redes Neurais, MLP, Classificação

## I. Introdução

Um neurônio artificial é o componente principal para o processamento de informações quando se trata de redes neurais, incluindo suas conexões de entrada, combinador linear e, claro, a função de ativação. Nesse sentido, as conexões de entrada possuem pesos que são multiplicados por cada sinal de entrada e, que por sua vez, são somados pelo combinador linear, resultando em um coeficiente potencial de ativação, comumente chamado de U, que por sua vez, é avaliado através da função de ativação.

Nesse contexto, redes neurais podem consistir em multicamadas de neurônios, sendo uma camada de entrada e uma camada de saída com uma ou mais camadas ocultas. Nessas redes, cada camada tem uma função única, onde a camada intermediária envia estimulos para a camada de saída e, assim, pode-se dizer que as camadas intermediárias funcionam como extratoras de características, para os quais os pesos codificam, permitindo que a rede crie sua própria representação, mais rica e complexa.

Assim, essa pesquisa trata-se de desenvolver uma MLP Classifier com as ferramentas proporcionadas pelas seguintes bibliotecas para a linguagem Python: Pandas e Numpy, que servem para operações matriciais; SciKit Learn e TensorFlow, oferencendo desde o tratamento à modelagem em Machine Learning.

Diante disso, a aplicação da MLP Classifier visa a classificação de um conjunto de dados sobre vidros, o qual será descrito mais detalhadamente nos próximos capítulos.

Contact data: Milena C. Rios de Oliveira, milena.rios@mail.uft.edu.br

## II. Metodologia

Em posse das bibliotecas descritas anteriormente, o primeiro passo consiste na compreensão dos dados coletados, que corresponde a um dataset com 214 registros de dados sobre vidro, os quais possuem um valor para cada um de seus 9 atributos (colunas), que são descritos abaixo.

- RI: Índice de Refração
- Na: Sódio (porcentagem do peso)
- Mg: Magnésio (porcentagem do peso)
- Al: Alumínio (porcentagem do peso)
- Si: Silício (porcentagem do peso)
- K: Potássio (porcentagem do peso)
- Ca: Cálcio (porcentagem do peso)
- Ba: Bário (porcentagem do peso)
- Fe: Ferro (porcentagem do peso)

O conjunto foi coletado da UC Repository of Machine Learning e também conta com uma coluna nomeada de Type, que indica a classificação do vidro, podendo ser 1, 2, 3, 4, 5, 6 ou 7. Dessa forma, buscou-se extrair a classificação dos dados, assim:

- $X_{train}$: contém todos os dados com as colunas, exceto Type, que foi dropada

- $Y_{train}$: contém apenas a coluna de tipo do vidro (Type)

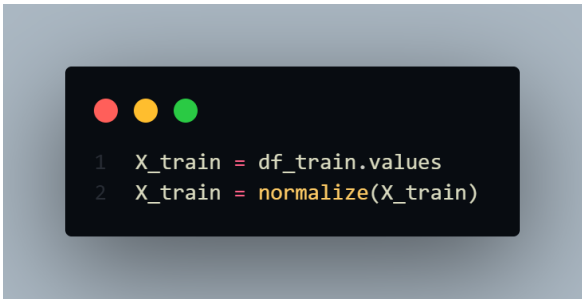
Assim, como os próprios nomes sugere, $X_{train}$ e $Y_{train}$ serão usador para treinar o modelo e onde será pego uma porcentagem nas etapas seguintes.

Diante disso, para melhor entendimento dos dados optou-se por plotar um gráfico de barras, representando a quantidade de registros de cada tipo, para isso o trecho de código abaixo desempenhou a função descrita.

```
glass_class = Y_train.unique()
values = Y_train.value_counts()
colors = np.arange(len(glass_class))
plt.bar(glass_class, values, color=plt.cm.Set3(colors))
plt.title('Train set')
plt.xlabel('Glass Classes')
plt.ylabel('Examples count')
plt.show()
```

**Fig. 1:** Gráfico da relação dos vidros e seus respectivos tipos

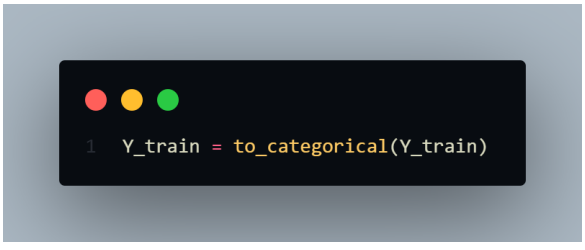
Além disso, visando englobar ainda mais essa análise exploratória, utilizou-se o método do pandas para demonstrar as informações do box plot de cada atributo de $X_{train}$ como count, mean, std, min, max e quartis. Posteriormente, os dados foram normalizados, isto é, colocados na mesma escala entre 0 e 1. Para isso utilizou o método normalize do tensorflow, conforme a imagem abaixo:



```
X_train = df_train.values
X_train = normalize(X_train)
```

**Fig. 2:** Normalização dos dados

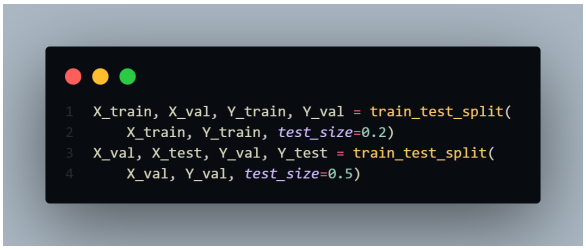
Em seguida, buscou se converter os valores de type que estão armazenados em $Y_{train}$ com a função to categorical, sendo comumente utilizada em problemas de classificação multiclasse para converter as categorias ou rótulos em uma representação one-hot encoding. A função to categorical converte os valores de $Y_{train}$ em uma matriz binária, onde cada categoria tem uma coluna correspondente, e o valor correspondente é 1 na coluna correspondente à categoria e 0 nas demais colunas. O código abaixo foi aplicado.



```
Y_train = to_categorical(Y_train)
```

**Fig. 3:** Categorização de Type - One Hot Encoder

Assim posto, os dados foram divididos em conjunto teste e conjunto treinamento. Sabendo que o conjunto não é tão grande, optou-se por dividir em 80% para treinar e 10% para testar e validar. Para isso, utilizando a função de train test split, aplicou-se o seguinte código:



```
X_train, X_val, Y_train, Y_val = train_test_split(
    X_train, Y_train, test_size=0.2)
X_val, X_test, Y_val, Y_test = train_test_split(
    X_val, Y_val, test_size=0.5)
```

**Fig. 4:** Categorização de Type - One Hot Encoder

Para a criação do modelo, utilizou se o Keras (biblioteca integrada ao TensorFlow) com os seguintes atributos:

- O modelo é construído com várias camadas densas (totalmente conectadas) que serão empilhadas uma após a outra, utilizando o método Dense

- A primeira camada densa possui 256 neurônios e usa a função de ativação 'relu'. Ela também especifica o formato de entrada esperado, que é um vetor de tamanho 10.

- Cria-se uma camada de normalização em lotes (BatchNormalization()) para normalizar os valores de entrada e melhorar a estabilidade do treinamento.

- Adiciona-se uma camada de dropout (Dropout(0.3)) é adicionada para combater o overfitting, descartando aleatoriamente 30% das conexões entre os neurônios durante o treinamento.

- Por fim, mais duas camadas de 256 e 512 neurônios, respectivamente, seguidas por camadas de normalização em lotes e dropout. A última camada densa tem 8 neurônios e usa a função de ativação 'softmax', que é bastante utilizada para problemas de classificação multiclasse, como é o caso do conjunto de vidros aplicados a esse estudo e conta com 7 classes distintas.

Após a definição do modelo, o método compile é executado com o objetivo de configurar a função de perda, o otimizador e acurácia. Assim, para a função de perda, utilizou-se categoricalCrossentropy e o Adam para otimizador. Para isso, a seguinte célula foi programada.

```python
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, input_shape=(10,), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(8, activation='softmax')
])

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(0.0001),
              metrics=['acc'])
model
```

**Fig. 5:** Criação do modelo de rede neural

Para sumarização dos resultados, utilizou o método sumary que indica todas as camadas utilizadas com a quantidade de neurônio para cada. Com o modelo criado, foi hora de treiná-lo:

```python
history = model.fit(X_train, Y_train, epochs=400,
                    validation_data=(X_val, Y_val),
                    verbose=2)
```

**Fig. 6:** Treinamento do modelo

Assim, utilizando a métrica de acurácia para avaliação, buscou-se plotar um gráfico com o comparativo para o conjunto de treinamento e conjunto de teste no modelo desenvolvido.

```python
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title("Acurácia do modelo")
plt.ylabel('acurácia')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

**Fig. 7:** Acurácia do Modelo

Em seguida também foi desenvolvido o gráfico demonstrando os valores de perda (loss) do modelo criado, sendo acessada a partir do objeto history, que é retornado ao treinar um modelo no TensorFlow.

```python
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

**Fig. 8:** Acurácia do Modelo

Diante disso, uma matriz de confusão foi desenvolvida a fim de mostrar a performance do modelo MLP, comparando as previsões feitas pelo modelo com as classes reais dos dados presente em $Y_{train}$.

```python
Y_pred = model.predict(X_test)
Y_pred_cl = np.argmax(Y_pred, axis=1)
Y_true = np.argmax(Y_test, axis=1)

confusion_matrix(Y_true, Y_pred_cl)
```

**Fig. 9:** Desenvolvimento da Matriz de Confusão

## III. Resultados

Quanto ao processo metodológico proposto pela figura 1, o seguinte gráfico foi gerado:

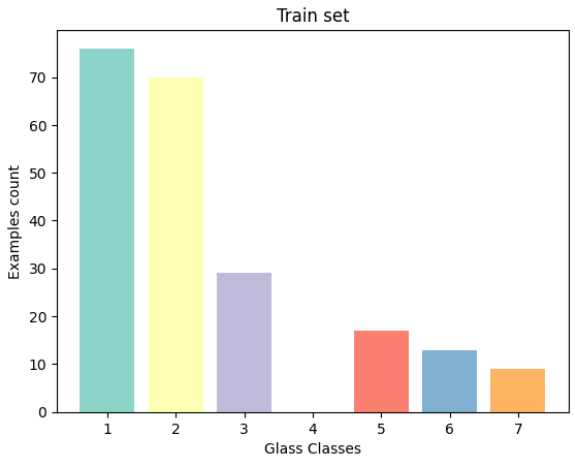



**Fig. 10:** Desenvolvimento da Matriz de Confusão

Quanto aos resultados de exploração pelo método describe, as informações de boxplot foram resultantes de:

| | RI | Na | Mg | Al | Si | K | Ca | Ba | Fe |
|---|---|---|---|---|---|---|---|---|---|
| count | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 |
| mean | 1.518365 | 13.407850 | 2.684533 | 1.444907 | 72.650935 | 0.497056 | 8.956963 | 0.175047 | 0.057009 |
| std | 0.003037 | 0.816604 | 1.442408 | 0.499270 | 0.774546 | 0.652192 | 1.423153 | 0.497219 | 0.097439 |
| min | 1.511150 | 10.730000 | 0.000000 | 0.290000 | 69.810000 | 0.000000 | 5.430000 | 0.000000 | 0.000000 |
| 25% | 1.516522 | 12.907500 | 2.115000 | 1.190000 | 72.280000 | 0.122500 | 8.240000 | 0.000000 | 0.000000 |
| 50% | 1.517680 | 13.300000 | 3.480000 | 1.360000 | 72.790000 | 0.555000 | 8.600000 | 0.000000 | 0.000000 |
| 75% | 1.519157 | 13.825000 | 3.600000 | 1.630000 | 73.087500 | 0.610000 | 9.172500 | 0.000000 | 0.100000 |
| max | 1.533930 | 17.380000 | 4.490000 | 3.500000 | 75.410000 | 6.210000 | 16.190000 | 3.150000 | 0.510000 |

**Fig. 11:** Informações de BoxPlot para o conjunto de vidros

Além disso, o resumo das camadas criadas no modelo MLP, são descritas na imagem abaixo:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               2816

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 512)               131584

 batch_normalization_2 (Batc  (None, 512)              2048
 hNormalization)

 dropout_2 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 8)                 4104

=================================================================
Total params: 208,392
Trainable params: 206,344
```

**Fig. 12:** Summary do modelo MLP

Já a avaliação utilizando a acurácia foi resultado de:

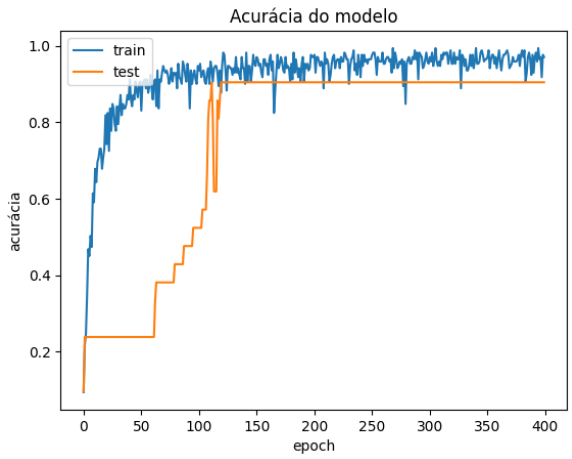



**Fig. 13:** Acurácia do modelo MLP

E os valores de perda, plotados da seguinte maneira:

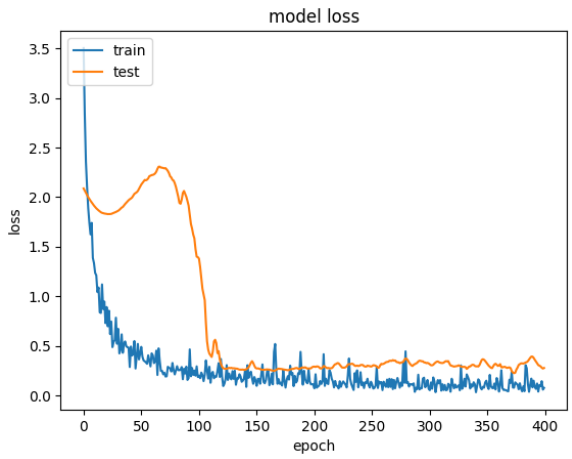



**Fig. 14:** Acurácia do modelo MLP

E, por fim, a matriz de confusão resultante foi:

```
1/1 [==============================] - 0s 236ms/step
array([[ 6,  0,  0,  0,  0],
       [ 0, 10,  0,  0,  0],
       [ 0,  0,  2,  0,  0],
       [ 0,  0,  0,  2,  1],
       [ 0,  0,  0,  0,  1]], dtype=int64)
```

**Fig. 15:** Matriz de Confusão do modelo MLP

## IV. Conclusão

Diante disso, o presente trabalho consistiu em desenvolver um classificador MLP (Multilayer Perceptron) utilizando a linguagem de programação Python com as bibliotecas: Pandas, Numpy, SciKit Learn e TensorFlow. O conjunto de dados escolhido aborda informações sobre vidros, com nove atributos e uma coluna indicando o tipo do vidro em sete tipos diferentes.

Os dados foram explorados e visualizados por meio de gráficos e análises estatísticas, como o boxplot. Em seguida, eles foram normalizados e divididos em conjuntos de treinamento e teste. O modelo MLP foi criado usando a biblioteca Keras, com camadas densas, normalização em lotes e dropout para evitar overfitting. O modelo foi treinado e avaliado usando métricas como acurácia, além de demonstrar os valores de perda.

No geral, este estudo demonstra a aplicação do MLP para classificação de vidros e destaca a relevância das redes neurais Multi-Camadas para os dias atuais.

## References

[1] André C. P. L. F. de Carvalho. *Redes Neurais Artificiais: Multilayer Perceptron*. Disponível em: `https://sites.icmc.usp.br/andre/research/neural/mlp.htm`.

[2] Ensina.AI. *Rede Neural: Perceptron Multicamadas*. Disponível em: `https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9`.

[3] Unknown. *Glass Identification Dataset*. Disponível em: `https://archive.ics.uci.edu/dataset/42/glass+identification`.

[4] Haykin, Simon. *Redes Neurais. Princípios e prática*, 2ª ed. Bookman, 2001. ISBN 9788573077186.