



**Multilayer Perceptron para regressão e classificação**  
**Profa. Dra. Roseli Aparecida Francelin Romero**  
**Disciplina: SCC 0270 - Redes Neurais**

**Lucas Yudi Sugi - 9293251**

## Introdução

O algoritmo de aprendizado supervisionado Multilayer Perceptron (MLP) é uma evolução do perceptron por permitir que haja múltiplas camadas escondidas, cada uma delas podendo ter um número variado de neurônios. Tal evolução viabiliza a resolução de problemas que sejam não linearmente separáveis.

Assim, dada a possibilidade de haver várias camadas e neurônios diferentes, a grande dificuldade da utilização do MLP é por conta de ajustar esses valores(parâmetros) para obter uma melhor classificação/regressão na base de dados.

Tendo isso em mente e sabendo que existem outros valores a serem ajustados no algoritmo como velocidade de aprendizado, momentum e ciclos de treinamento, iremos realizar uma análise em duas bases de dados para compreender um pouco melhor como é o seu processo de escolha e ajuste.

## Objetivos

Utilizar o algoritmo MLP nas bases de dados:

- <https://archive.ics.uci.edu/ml/datasets/Wine>
- <https://archive.ics.uci.edu/ml/datasets/Geographical+Original+of+Music>

Realizando classificação na primeira e regressão na última, com o intuito de estudar os seguintes parâmetros:

- Número de camadas intermediárias
- Ciclos usados no treinamento
- Velocidade de aprendizado e momentum
- Proporção de dados para treinamento e teste

## Desenvolvimento

Inicialmente para que houvesse um melhor aprendizado por parte do algoritmo, foi necessário realizar um pré-processamento dos dados. Este por sua vez foi obtido utilizando a biblioteca pandas do python e o pré-processamento gerado foi a normalização dos dados para ambos os dataset's assim como a construção das classes:

- Wine: Neste dataset como possuímos as classes categóricas 1,2 e 3 gerou-se três variáveis binárias para representar cada uma: 1(0 0 1), 2(0 1 0)

e 3(1 0 0). Assim, é possível obter a ideia de que para uma determinada classe apenas um neurônio é ativado e o resto não.

- Default features 1059 tracks: Neste dataset como desejamos realizar uma regressão, utilizou-se novamente uma normalização (a mesma para os dados) com a seguinte equação:  $X = (X - \text{MIN}) / (\text{MAX} - \text{MIN})$ .

Toda esta etapa consta no arquivo *pre\_processing.py*. Após esse pré-processamento podemos descrever sobre o desenvolvimento do MLP, este por sua vez, foi dividido em três principais funções:

- Architecture: Nesta função é realizada a etapa de criação da arquitetura do MLP. Então, por exemplo, supondo que temos uma camada intermediária e seja fornecido 3 características, 4 neurônios e 2 classes, esta função irá criar 2 matrizes (uma para hidden layer e outra para a output layer) contendo as seguintes dimensões: (4x3) e (2x4).
- Forward: Nesta função é realizado o que é conhecido como forward em uma rede neural, i.e, ocorre a multiplicação das matrizes (representante dos pesos) pelos valores de entrada. Portanto, aqui ocorre a geração dos net's e f(net)'s de cada neurônio.
- Backward: Nesta função é onde realmente ocorre o aprendizado. Aqui são calculadas todas as derivadas necessárias para realizar o ajuste dos pesos.

Importante salientar que as funções acima são as principais do algoritmo e foram descritas de forma superficial pois não é objetivo deste documento entrar em detalhes específicos de implementação. Caso o leitor deseje entender melhor o seu funcionamento é aconselhável que o mesmo analise os códigos escritos em python dispostos no mesmo diretório deste documento.

Além disso, é importante notar que por questões de facilidade o algoritmo MLP foi dividido em dois arquivos:

- mlp\_1\_layer: Contém o MLP para uma camada escondida.
- mlp\_2\_layer: Contém o MLP para duas ou mais camadas escondidas.

## Pastas e arquivos

Iremos descrever como está organizado o projeto em pastas e arquivos:

- data: Pasta contendo os dataset's.
  - default\_features\_1059\_tracks.txt: Dataset utilizado para regressão.
  - wine.data: Dataset utilizado para classificação.
- data\_pre: Pasta contendo os dataset's pré-processados.

- default\_features\_1059\_tracks\_pre.txt: Dataset pré-processado utilizado para regressão.
- wine\_pre.data: Dataset pré-processado utilizado para classificação.
- src: Pasta contendo todos os arquivos fontes em python.
  - pre\_processing.py: Realização do pré-processamento.
  - mlp\_1\_layer: MLP para uma camada escondida.
  - mlp\_2\_layer: MLP para várias camadas escondidas.
- Makefile: Automatização do algoritmo usando makefile.
- input.txt: Entradas necessárias para o algoritmo.

## Entrada do algoritmo

O algoritmo utiliza o arquivo input.txt que contém as entradas necessárias para um teste. Caso o leitor deseje testar o MLP de outras maneiras ele deve alterar este arquivo seguindo esta ordem:

- mlp\_1\_layer: Temos as seguintes entradas:
  - Caminho para o dataset.
  - Número de variáveis do dataset (mesmo número de neurônios na camada de entrada).
  - Número de neurônios na camada escondida.
  - Número de classes (mesmo número de neurônios na camada de saída).
  - Número de iterações.
  - Velocidade de aprendizado (eta).
  - Momentum.
  - Proporção de dados para o conjunto de treinamento.
  - Classificação(C) ou regressão(R) a ser realizado no dataset.
- mlp\_2\_layer: Temos as seguintes entradas:
  - Caminho para o dataset.
  - Número de variáveis do dataset (mesmo número de neurônios na camada de entrada).
  - Número de neurônios na primeira camada escondida.
  - Número de neurônios na segunda camada escondida.
  - Número de classes (mesmo número de neurônios na camada de saída).
  - Número de iterações.
  - Velocidade de aprendizado (eta).
  - Momentum.
  - Proporção de dados para o conjunto de treinamento.
  - Classificação(C) ou regressão(R) a ser realizado no dataset.

## Geração do conjunto de treinamento e teste

Um dos pontos importantes a serem abordados é quanto a geração do conjunto de treinamento e teste. Ambos são gerados de forma totalmente aleatória segundo as proporções passadas para o algoritmo.

Assim, é importante salientar que nesta geração não há uma verificação quanto a sobreposição de classes de um conjunto sobre o outro. Acredita-se, no entanto, que pelo fato da geração ser totalmente aleatória e tendo várias amostras aleatórias não haverá uma interferência significativa desta sobreposição. Obviamente devemos realizar um número considerável de amostras para que a premissa seja aceita.

Ademais a geração dos conjuntos ocorre no próprio algoritmo de MLP.

## Teste

Nesta etapa de teste nós avaliamos a acurácia e o erro médio quadrático para os problemas de classificação e regressão, respectivamente. Essa avaliação foi realizada em torno de 20 vezes variando os parâmetros descritos na seção de objetivos.

Assim, o teste foi realizado da seguinte maneira: para cada dataset testamos primeiro o MLP com uma camada escondida e depois com duas, sempre permutando os parâmetros.

Desse modo, a seguinte lógica foi seguida: para um determinado número de ciclos baixo (alto) a tupla velocidade de aprendizado e momentum eram altas (baixa) e para cada uma delas testou-se diferentes valores de treinamento/teste.

Importante notar que valores fixos de parâmetros foram escolhidos para facilitar o teste. Abaixo estão as tabelas com os valores amostrados:

### Classificação(wine)

N° camadas intermediárias	Ciclos treinamento	Velocidade de aprendizado e momentum	Proporção treinamento/ teste	Ácurácia
1	50	(0.2,0.7)	(0.7,0.3)	0.981
1	50	(0.2,0.7)	(0.5,0.5)	0.955
1	25	(0.4,0.6)	(0.7,0.3)	0.982
1	25	(0.4,0.6)	(0.5,0.5)	0.932

2	150	(0.2,0.7)	(0.7,0.3)	0.996
2	150	(0.2,0.7)	(0.5,0.5)	0.988
2	100	(0.4,0.6)	(0.7,0.3)	0.962
2	100	(0.4,0.6)	(0.5,0.5)	0.925

Tabela 1: Medição de acurácia para os parâmetros.

### **Classificação(default\_feature\_1059\_tracks)**

N° camadas intermediárias	Ciclos treinamento	Velocidade de aprendizado e momentum	Proporção treinamento/ teste	Erro médio quadrático
1	500	(0.2,0.7)	(0.7,0.3)	0.0821
1	500	(0.2,0.7)	(0.5,0.5)	0.0848
1	250	(0.4,0.6)	(0.7,0.3)	0.0716
1	250	(0.4,0.6)	(0.5,0.5)	0.0802
2	300	(0.2,0.7)	(0.7,0.3)	0.0615
2	300	(0.2,0.7)	(0.5,0.5)	0.0530
2	150	(0.4,0.6)	(0.7,0.3)	0.0561
2	150	(0.4,0.6)	(0.5,0.5)	0.0554

Tabela 2: Medição do erro quadrático médio para os parâmetros.

Devemos salientar que para o dataset wine foi utilizado 4 neurônios na mlp\_1\_layer e 2 em cada camada no mlp\_2\_layer. Com relação ao default\_feature\_1059\_tracks temos 10 neurônios no mlp\_1\_layer e 7 em cada camada no mlp\_2\_layer.

### **Conclusão**

Após analisar as tabelas 1 e 2 da seção de testes, podemos realizar certas conclusões que serão feitas em separado para cada base de dados.

## Wine

Note que em geral temos uma pequena piora quando utilizamos menos dados para o treinamento, apesar da diferença ser bem pequena. Também é interessante observar que para menores (maiores) números de ciclo com maiores (menores) velocidades de aprendizado e momentum obtemos um resultado satisfatório. Certamente se usássemos o número de ciclos, velocidade de aprendizado e momentum altos, teríamos uma conversão mais rápida e com bons resultados.

Outro ponto interessante é com relação ao número de camadas. Note que temos uma pequena melhora na acurácia quando se usa duas camadas intermediárias mas a mesma não é tão relevante assim.

Desse modo, para o dataset wine podemos concluir que o melhor seria utilizar apenas uma camada intermediária (diminuindo a complexidade do algoritmo), pois lembre-se que se um problema pode ser resolvido da maneira mais simples não há razão em se utilizar algo mais complexo. Com relação aos outros parâmetros o ideal seria utilizá-los com menor número de ciclos e velocidade de aprendizado e momentum mais altos, diminuindo a complexidade. A proporção do conjunto de treinamento não parece afetar tanto a acurácia e ficaria a critério do usuário.

## Default\_feature\_1059\_tracks

Neste dataset temos que a proporção treinamento/teste afeta um pouco o erro médio quadrático mas nada de tão relevante. O mais interessante a se observar na tabela é que quando aumentamos o número de camadas e deixamos os ciclos mais baixos com velocidade de aprendizado e momentum altos, é quando o erro médio quadrático tende a ser o menor de todos.

O que pode ser observado de um modo geral é que quando tentou-se treinar mais o algoritmo os resultados acabaram não sendo tão bons. Isso está diretamente relacionado com o que é chamado de overfitting. Como estamos trabalhando com um problema de regressão é mais interessante não treinar tanta a MLP para que ela seja capaz de possuir bons resultados no conjunto de teste.

Certamente isso faz total sentido, pois caso “aprendemos” muito no treinamento o algoritmo não será capaz de generalizar os novos pontos que encontrar. Observe, contudo, que quando aumentamos o número de camadas intermediárias obtemos um melhor resultado. Isso se deve pelo fato do problema ser mais “difícil” com o comparado ao wine, por exemplo. Isto é, seu espaço de dados necessita de uma complexidade maior.

Assim, podemos concluir que o MLP deveria ser utilizado neste dataset de forma a não causar muito overfitting (usar poucos ciclos com os parâmetros velocidade de aprendizado e momentum altos) e mais camadas intermediárias devido a complexidade do problema.

## **Conclusão final**

Ao final destas duas conclusões fica evidente que não existe uma fórmula mágica de definição dos parâmetros. Para cada dataset que o MLP for utilizado é necessário realizar um estudo comparando os diversos valores na tentativa de encontrar aquele que satisfaça os objetivos definidos pelo usuário.

Além disso, conforme dito anteriormente caso o problema seja solucionado da maneira mais simples não há o porquê de aumentar a complexidade do algoritmo demorando mais tempo para conversão, gastando mais memória, etc.

Concluimos, portanto que sempre é necessário realizar testes a fim de verificar qual caminho satisfaz melhor os objetivos do usuário.