

MLPRegressor (Aplicação em dados Climaticos)

Ana Paula Vanderley

- Multi-layer Perceptron Regressor é um modelo de regressão baseado em redes neurais artificiais uma poderosa ferramenta para modelagem de regressão para aprender relações complexas entre variáveis de entrada e saída, bastante flexível para lidar com uma variedade de problemas de regressão.
- Além de apenas fazer previsões, o MLPRegressor nos permite entender como cada variável de entrada contribui para as previsões finais. Isso é útil para insights meteorológicos e tomada de decisão baseada em dados.
- O objetivo desse trabalho foi demonstrar a utilização do MLPRegressor com as features selecionadas através de um modelo de Randon Florest processo que foi bastante útil porque permitiu capturar a complexidade das relações entre variáveis meteorológicas, proporcionando uma modelagem robusta e previsões mais precisas das rajadas máximas de vento com esse banco de dados de clima de Salvador.

```
# @title
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
```

BANCO DE DADOS

```
# @title
df = pd.read_csv('climaSalvador.csv' , sep = ';')
```

```
# @title
df.head()
```

	Data Medicao	PRECIPITACAO TOTAL, DIARIO (AUT) (mm)	PRESSAO ATMOSFERICA MEDIA DIARIA (AUT) (mB)	TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT) (°C)	TEMPERATURA MAXIMA, DIARIA (AUT) (°C)	TEMPERATURA MEDIA, DIARIA (AUT) (°C)	TEMPERATURA MINIMA, DIARIA (AUT) (°C)	UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT) (%)	UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT) (%)	VENTO, RAJADA MAXIMA DIARIA (AUT) (m/s)	VENTO, VELOCIDADE MEDIA DIARIA (AUT) (m/s)
0	01/01/2022	0.0	1.006.533.333	218.625	30.4	26.445.833	24.2	76.166.667	64	8.7	1.145.833
1	02/01/2022	0.0	10.071.875	21.675	30.8	26.729.167	23.5	74.333.333	60	5.2	1.004.167
2	03/01/2022	0.0	1007.2	21.258.333	30.9	26.708.333	24.0	72.666.667	56	6.8	10.125
3	04/01/2022	0.0	1006.1	21.733.333	28.9	250.625	23.1	82	69	5.6	.85
4	05/01/2022	8.8	10.055.375	21.970.833	31.2	26.7	22.9	76.041.667	58	8.3	1.179.167

CONTAGEM DO BANCO DE DADOS

```
# @title
df.count()
```

	0
Data Medicao	365
PRECIPITACAO TOTAL, DIARIO (AUT)(mm)	365
PRESSAO ATMOSFERICA MEDIA DIARIA (AUT)(mB)	365
TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)	365
TEMPERATURA MAXIMA, DIARIA (AUT)(°C)	365
TEMPERATURA MEDIA, DIARIA (AUT)(°C)	365
TEMPERATURA MINIMA, DIARIA (AUT)(°C)	365
UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)	365
UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)	365
VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)	365
VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)	365

MEDIDAS DESCRITIVAS

```
# @title
df.describe()
```



	PRECIPITACAO TOTAL, DIARIO (AUT)(mm)	TEMPERATURA MAXIMA, DIARIA (AUT)(°C)	TEMPERATURA MINIMA, DIARIA (AUT)(°C)	UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)	VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)
count	365.000000	365.000000	365.000000	365.000000	365.000000
mean	5.247671	29.115616	23.134795	64.175342	7.341644
std	10.994366	1.890724	1.435603	7.176499	1.813008
min	0.000000	23.700000	19.100000	45.000000	4.000000
25%	0.000000	27.700000	22.100000	60.000000	6.100000
50%	0.800000	29.300000	23.200000	63.000000	7.000000
75%	5.800000	30.600000	24.200000	68.000000	8.300000
max	100.200000	33.200000	26.000000	90.000000	14.600000

TIPO DE DADOS

```
# @title
pd.DataFrame(df.dtypes, columns = ['Tipo'])
```



	Tipo
Data Medicao	object
PRECIPITACAO TOTAL, DIARIO (AUT)(mm)	float64
PRESSAO ATMOSFERICA MEDIA DIARIA (AUT)(mB)	object
TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)	object
TEMPERATURA MAXIMA, DIARIA (AUT)(°C)	float64
TEMPERATURA MEDIA, DIARIA (AUT)(°C)	object
TEMPERATURA MINIMA, DIARIA (AUT)(°C)	float64
UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)	object
UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)	int64
VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)	float64
VENTO. VELOCIDADE MEDIA DIARIA (AUT)(m/s)	object

- Foi preciso converter o tipo de dados de algumas variáveis para seguir com as análises .

```
# @title
df['TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)'] = df['TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)'].str.replace('.', '').astype(float)
df['TEMPERATURA MEDIA, DIARIA (AUT)(°C)'] = df['TEMPERATURA MEDIA, DIARIA (AUT)(°C)'].str.replace('.', '').astype(float)
df['UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)'] = df['UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)'].str.replace('.', '').astype(float)
df['VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)'] = df['VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)'].str.replace('.', '').astype(float)
```

```
# @title
pd.DataFrame(df.dtypes, columns = ['Tipo'])
```



	Tipo
Data Medicao	object
PRECIPITACAO TOTAL, DIARIO (AUT)(mm)	float64
PRESSAO ATMOSFERICA MEDIA DIARIA (AUT)(mB)	object
TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)	float64
TEMPERATURA MAXIMA, DIARIA (AUT)(°C)	float64
TEMPERATURA MEDIA, DIARIA (AUT)(°C)	float64
TEMPERATURA MINIMA, DIARIA (AUT)(°C)	float64
UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)	float64
UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)	int64
VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)	float64
VENTO. VELOCIDADE MEDIA DIARIA (AUT)(m/s)	float64

VISUALIZAÇÃO

```
# @title
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
```

```
# @title
df['Data Medicao'] = pd.to_datetime(df['Data Medicao'], dayfirst=True) #definindo a variavel data medição como datetime

# @title

sns.set(style="whitegrid")

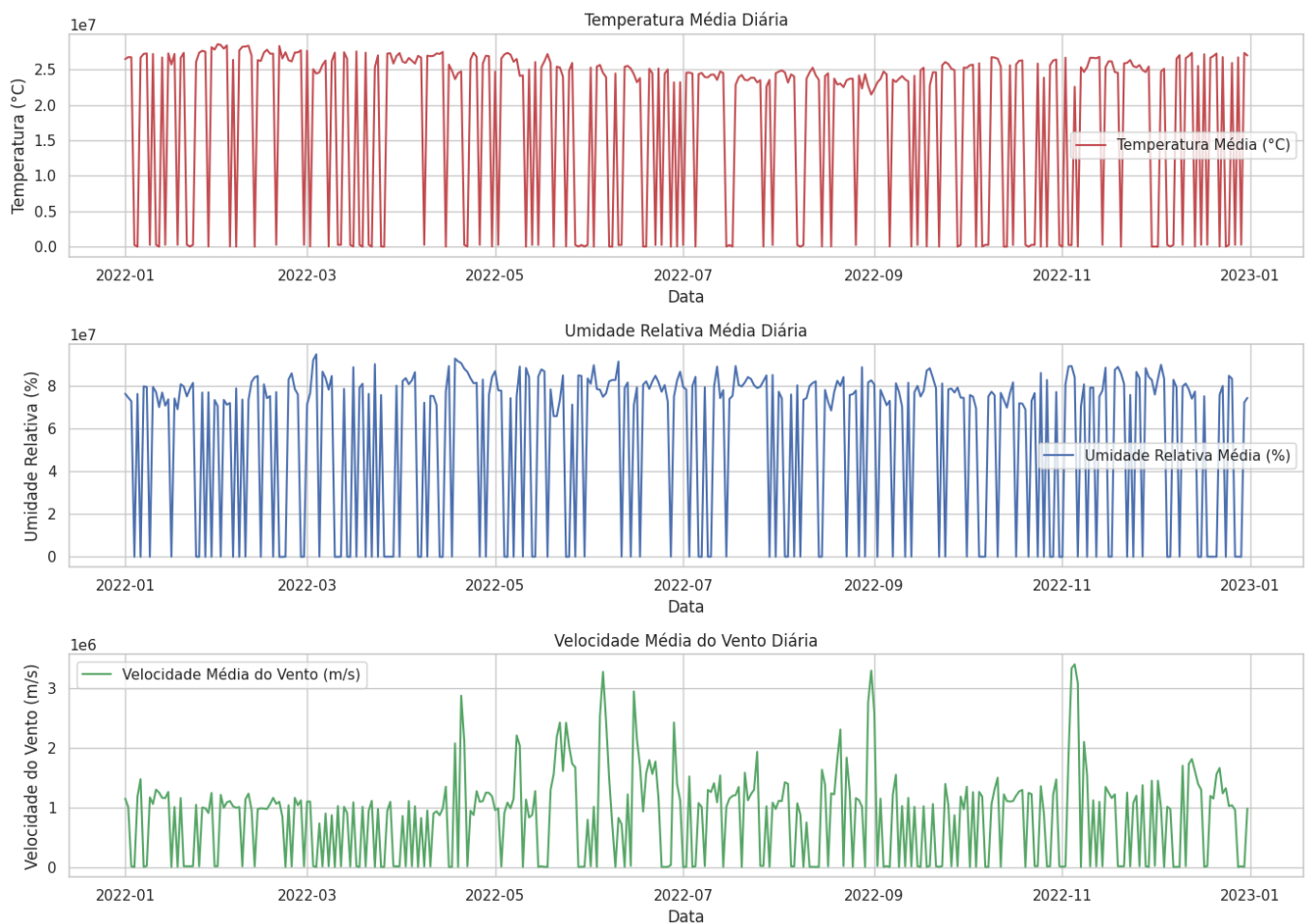
plt.figure(figsize=(14, 10))

plt.subplot(3, 1, 1)
plt.plot(df['Data Medicao'], df['TEMPERATURA MEDIA, DIARIA (AUT)(°C)'], label='Temperatura Média (°C)', color='r')
plt.title('Temperatura Média Diária')
plt.xlabel('Data')
plt.ylabel('Temperatura (°C)')
plt.legend()

plt.subplot(3, 1, 2)
plt.plot(df['Data Medicao'], df['UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)'], label='Umidade Relativa Média (%)', color='b')
plt.title('Umidade Relativa Média Diária')
plt.xlabel('Data')
plt.ylabel('Umidade Relativa (%)')
plt.legend()

plt.subplot(3, 1, 3)
plt.plot(df['Data Medicao'], df['VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)'], label='Velocidade Média do Vento (m/s)', color='g')
plt.title('Velocidade Média do Vento Diária')
plt.xlabel('Data')
plt.ylabel('Velocidade do Vento (m/s)')
plt.legend()

plt.tight_layout()
plt.show()
```




```
# @title
df.isnull().sum()
```



	0
Data Medicao	0
PRECIPITACAO TOTAL, DIARIO (AUT)(mm)	0
PRESSAO ATMOSFERICA MEDIA DIARIA (AUT)(mB)	0
TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)	0
TEMPERATURA MAXIMA, DIARIA (AUT)(°C)	0
TEMPERATURA MEDIA, DIARIA (AUT)(°C)	0
TEMPERATURA MINIMA, DIARIA (AUT)(°C)	0
UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)	0
UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)	0
VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)	0
VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)	0

```
# @title
df.fillna(0,inplace=True) # caso haja valores ausentes
```

```
# @title
df.isnull().sum()
```



	0
Data Medicao	0
PRECIPITACAO TOTAL, DIARIO (AUT)(mm)	0
PRESSAO ATMOSFERICA MEDIA DIARIA (AUT)(mB)	0
TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)	0
TEMPERATURA MAXIMA, DIARIA (AUT)(°C)	0
TEMPERATURA MEDIA, DIARIA (AUT)(°C)	0
TEMPERATURA MINIMA, DIARIA (AUT)(°C)	0
UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)	0
UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)	0
VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)	0
VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)	0

## ✓ RANDOM FLOREST ( Para seleccionar as features mais importantes para o modelo)

```
# @title
from sklearn.ensemble import RandomForestRegressor
```

```
# @title
# Seleccionando as features e o target
X = df[['PRECIPITACAO TOTAL, DIARIO (AUT)(mm)',
        'TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (AUT)(°C)',
        'TEMPERATURA MAXIMA, DIARIA (AUT)(°C)',
        'TEMPERATURA MEDIA, DIARIA (AUT)(°C)',
        'TEMPERATURA MINIMA, DIARIA (AUT)(°C)',
        'UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)',
        'UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)',
        'VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)']]
y = df['VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)']
```

### DIVISÃO BASE DE TREINO E TESTE

```
# @title
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

```
# @title
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=30) # dividindo os dados de treino e teste
```

## ESCALONAMENTO DAS VARIÁVEIS

```
# @title
# Escalonando as features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

## TREINANDO O MODELO

```
# @title
rf_model = RandomForestRegressor(random_state=30)
rf_model.fit(X_train_scaled, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=30)
```

## ✓ Criando um DataFrame com as importâncias das features

```
# @title
feature_importances = rf_model.feature_importances_

# @title

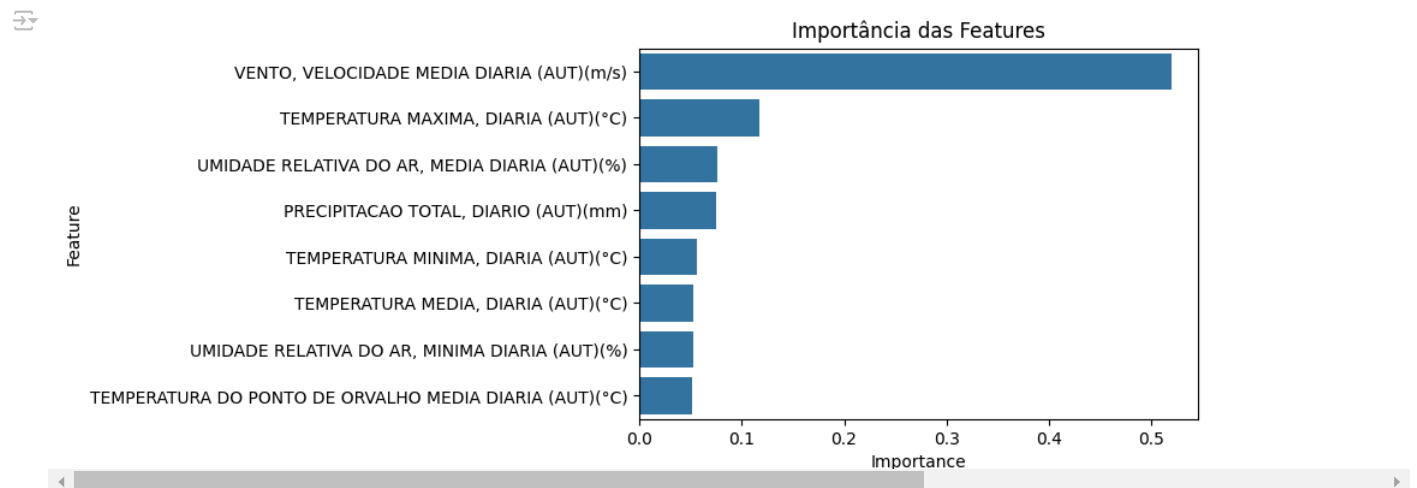
feature_names = X.columns
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})

# @title
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# @title
feature_importance_df
```

	Feature	Importance
7	VENTO, VELOCIDADE MEDIA DIARIA (AUT)(m/s)	0.519382
2	TEMPERATURA MAXIMA, DIARIA (AUT)(°C)	0.116908
5	UMIDADE RELATIVA DO AR, MEDIA DIARIA (AUT)(%)	0.076037
0	PRECIPITACAO TOTAL, DIARIO (AUT)(mm)	0.074873
4	TEMPERATURA MINIMA, DIARIA (AUT)(°C)	0.056202
3	TEMPERATURA MEDIA, DIARIA (AUT)(°C)	0.052729
6	UMIDADE RELATIVA DO AR, MINIMA DIARIA (AUT)(%)	0.052632
1	TEMPERATURA DO PONTO DE ORVALHO MEDIA DIARIA (...)	0.051238

```
# @title
plt.figure(figsize=(6, 4))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Importância das Features')
plt.show()
```



## SELECIONANDO AS FEACTURES MAIS INPORTANTES PARA O MODELO

```
# @title
X = df[['VENTO', VELOCIDADE MEDIA DIARIA (AUT)(m/s)',
        'TEMPERATURA MAXIMA, DIARIA (AUT)(°C)']]
y = df['VENTO, RAJADA MAXIMA DIARIA (AUT)(m/s)']
```

- As features selecionadas foram VENTO, **VELOCIDADE MEDIA DIARIA (AUT)(m/s)** que representa a velocidade média do vento diário. É uma variável importante porque o comportamento do vento ao longo do dia pode influenciar diretamente a velocidade das rajadas máximas de vento e **TEMPERATURA MAXIMA, DIARIA (AUT)(°C)** que representa a temperatura máxima diária outra feature importante pois a temperatura pode afetar a dinâmica atmosférica e, consequentemente, influenciar a velocidade e intensidade do vento.

## MODELO MLPRegressor

- O MLPRegressor é baseado em uma rede neural artificial com várias camadas de neurônios. Cada camada recebe entradas, realiza cálculos com pesos associados e passa esses valores para a próxima camada, até chegar à camada de saída que produz a previsão. As camadas intermediárias entre a entrada e a saída são chamadas de camadas ocultas.
- O modelo é treinado de forma iterativa usando métodos baseados em gradiente, Durante o treinamento, os pesos dos neurônios são ajustados para minimizar a função de perda, no caso da regressão é o erro quadrático médio (MSE).
- O MLPRegressor é adequado para capturar relações não lineares entre as variáveis de entrada (vento médio e temperatura máxima) e a variável de saída (rajada máxima de vento). Isso é importante porque os fenômenos atmosféricos, como o vento, frequentemente exibem comportamentos complexos e não lineares.
- O modelo também é capaz de aprender padrões complexos nos dados, o que é crucial quando se lida com previsões meteorológicas, onde múltiplas variáveis podem interagir de maneiras não óbvias para influenciar a velocidade das rajadas de vento.
- focamos aqui nesse trabalho em variáveis que têm um impacto direto ou indireto na variável de interesse (rajada máxima de vento). Isso nos permite que o modelo capture melhor as variações e padrões nos dados, resultando em previsões mais precisas.

```
# @title
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

## MODELO 1 (com os parâmetros automaticos da própria função)

```
# Escalonando as variáveis de saída
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))
```

```
estimatorNN = MLPRegressor(random_state=30) # modelo basico com os parametros pré determinados
estimatorNN.fit(X_train_scaled, y_train_scaled.ravel())
```

```
⚙ /usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations exceeded.
warnings.warn(
  MLPRegressor
  MLPRegressor(random_state=30)
```

```
y_pred_train_scaled = estimatorNN.predict(X_train_scaled)
y_pred_test_scaled = estimatorNN.predict(X_test_scaled)
```

```
y_pred_train = scaler_y.inverse_transform(y_pred_train_scaled.reshape(-1, 1)).ravel()
y_pred_test = scaler_y.inverse_transform(y_pred_test_scaled.reshape(-1, 1)).ravel()
```

```
mse_train = mean_squared_error(y_train, y_pred_train)
mse_test = mean_squared_error(y_test, y_pred_test)
r2_train = r2_score(y_train, y_pred_train)
r2_test = r2_score(y_test, y_pred_test)
```

```
print(f'MSE Train: {mse_train:.4f}, R² Train: {r2_train:.4f}')
print(f'MSE Test: {mse_test:.4f}, R² Test: {r2_test:.4f}')
```

```
⚙ MSE Train: 1.1024, R² Train: 0.6354
MSE Test: 1.9848, R² Test: 0.5379
```

## MODELO 2 (com os parâmetros encontrados com a aplicação do Grid Search)

ENCONTRANDO OS MELHORES PARAMETROS PARA O MODELO

- Aplicação do Grid Search para encontrar os melhores hiperparâmetros para o modelo de regressão usando rede neural MLP (Multi-layer Perceptron) ele define uma grade de possíveis combinações de hiperparâmetros para serem testadas.
- Foi utilizado o GridSearchCV para explorar todas as combinações possíveis de hiperparâmetros definidos em **param\_grid**. O modelo é validado usando validação cruzada (cv=5) e a métrica de avaliação é o erro quadrático médio negativo (scoring='neg\_mean\_squared\_error').

```
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'hidden_layer_sizes': [100], # uma tupla que define o número de neurônios em cada camada oculta da rede neural.
    'alpha': [0.001, 0.01, 0.1], # parâmetro de regularização L2 para evitar o overfitting da rede neural.
    'max_iter': [500], # número máximo de iterações
    'early_stopping': [True],
    'validation_fraction': [0.1, 0.2],
    'n_iter_no_change': [20, 30],
    'solver': ['adam', 'lbfgs', 'sgd'], # o otimizador usado para ajustar os pesos da rede neural durante o treinamento.
    'activation': ['relu', 'tanh', 'logistic'], # funções de ativação
    'learning_rate_init': [0.001, 0.01, 0.1] # taxas de aprendizado iniciais
}
```

```
estimatorNN = MLPRegressor(learning_rate='adaptive', random_state=30, verbose=False)
```

```
grid_search = GridSearchCV(estimatorNN, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train_scaled.ravel())
```

```
best_params = grid_search.best_params_
print("Melhores parâmetros encontrados:")
print(best_params)
```

```
Melhores parâmetros encontrados:
{'activation': 'relu', 'alpha': 0.1, 'early_stopping': True, 'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'max_iter': 500, 'n_iter_no_cha
```

## TREINAMENTO DO MODELO

```
estimatorNN = MLPRegressor(
    activation= 'relu',
    alpha= 0.1,
    early_stopping= True,
    hidden_layer_sizes= 100,
    learning_rate_init= 0.01,
    max_iter= 500,
    n_iter_no_change= 30,
    solver= 'adam',
    validation_fraction= 0.1
)
```

```
# @title
estimatorNN.fit(X_train_scaled, y_train_scaled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:1625: DataConversionWarning: A column-vector y was passed
y = column_or_1d(y, warn=True)

MLPRegressor
MLPRegressor(alpha=0.1, early_stopping=True, hidden_layer_sizes=100,
learning_rate_init=0.01, max_iter=500, n_iter_no_change=30)
```

## PREVISÕES

```
# @title
y_pred_train_scaled = estimatorNN.predict(X_train_scaled)
y_pred_test_scaled = estimatorNN.predict(X_test_scaled)
```

```
# @title
# Revertendo a escala das previsões
y_pred_train = scaler_y.inverse_transform(y_pred_train_scaled.reshape(-1, 1)).ravel()
y_pred_test = scaler_y.inverse_transform(y_pred_test_scaled.reshape(-1, 1)).ravel()
```

## MEDIDAS DE DESEMPENHO DO MODELO

```
# @title
r2_train = r2_score(y_train, y_pred_train)
r2_test = r2_score(y_test, y_pred_test)

print(f'MSE Train: {mse_train:.4f}, R² Train: {r2_train:.4f}')
print(f'MSE Test: {mse_test:.4f}, R² Test: {r2_test:.4f}')
```

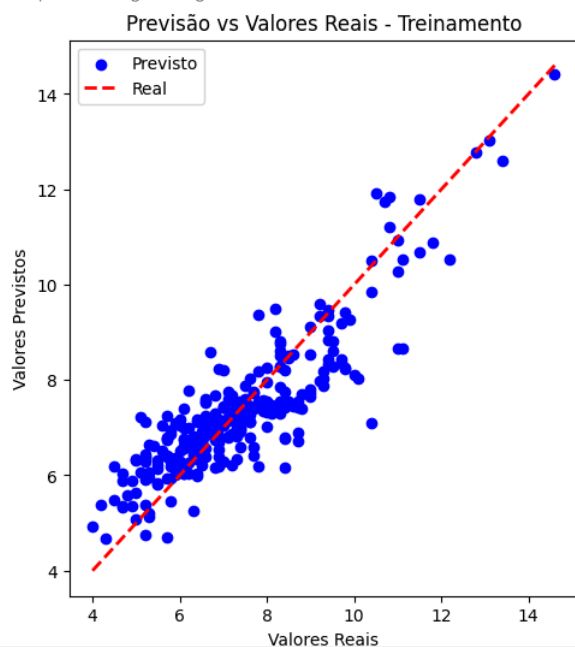
```
↗ MSE Train: 1.1024, R² Train: 0.7669
MSE Test: 1.9848, R² Test: 0.5675
```

- Pela análise dos valores do MSE podemos ver quão próximas estão as previsões dos valores reais. No nosso caso tanto no conjunto de treino quanto no conjunto de teste, os valores de MSE estão relativamente próximos, nos indicando que o modelo está generalizando bem para novos dados.
- já o  $R^2$  é uma medida que nos indica o quão bem as variáveis independentes explicam a variabilidade dos dados variando entre 0 a 1. No nosso caso tanto no treino quanto no teste, o  $R^2$  está acima de 0.5, o que indica que o modelo consegue explicar uma parte significativa da variabilidade desses dados.
- Houve uma pequena melhoria no  $R^2$  Train quando os parâmetros foram ajustados o modelo a se ajustou melhor aos dados de treinamento, aumentando a capacidade de explicar a variância desses dados. E apresentou também uma ligeira melhoria no  $R^2$  Test indicando que o modelo ajustado consegue generalizar melhor para dados não vistos.
- No entanto o MSE permaneceu constante após o ajuste do modelo ,ou seja, não houve um impacto significativo na precisão das previsões.
- Esse trabalho foi feito soente com a finalidade de demonstração das técnicas estatísticas , portanto é Importante levar em consideração que se trata de um banco de dados pequeno com dados de apenas 1 ano ,para futuros trabalhos é interessante testar novas abordagens assim como obter o maior número de dados possivel e observar o comportamento do modelo.

## ✓ PREVISTO X VALORES REAIS

```
# @title
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_train, y_pred_train, color='blue', label='Previsto')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)], '--', color='red', linewidth=2, label='Real')
plt.title('Previsão vs Valores Reais - Treinamento')
plt.xlabel('Valores Reais')
plt.ylabel('Valores Previstos')
plt.legend()
```

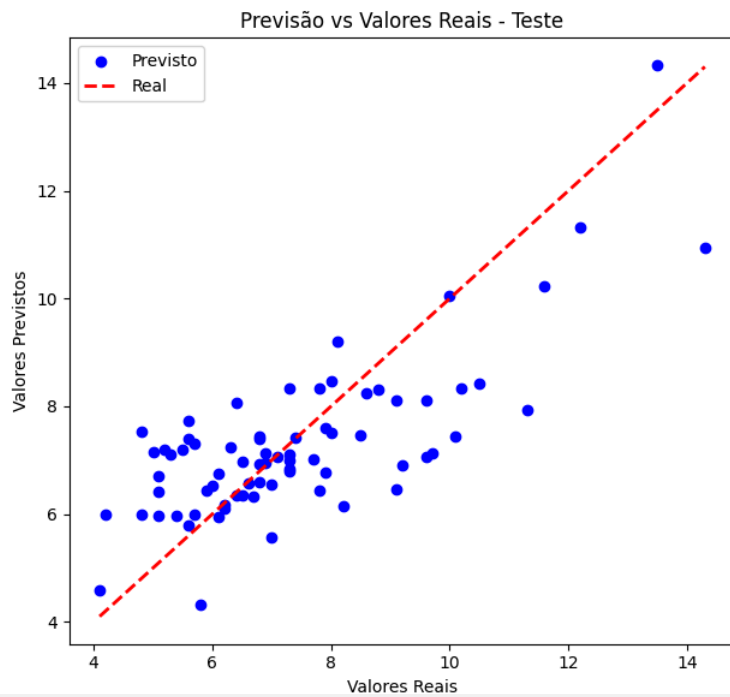
```
↗ <matplotlib.legend.Legend at 0x7b1091daab30>
```



```
# @title
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_test, color='blue', label='Previsto')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', color='red', linewidth=2, label='Real')
plt.title('Previsão vs Valores Reais - Teste')
plt.xlabel('Valores Reais')
plt.ylabel('Valores Previstos')
plt.legend()

plt.tight_layout()
plt.show()
```





- Os resultados obtidos foram previsões mais precisas obtidas ao acrescentar variáveis significativas como a velocidade média do vento e a temperatura máxima, o modelo nos forneceu previsões mais precisas das rajadas máximas de vento, o que é crucial para aplicações como previsão meteorológica e gestão de riscos.
- Não somente para fazer previsões o MLPRegressor nos permitiu entender como cada variável de entrada contribui para as previsões finais. Fato esse bastante útil para insights meteorológicos e tomada de decisão baseada em dados pois foi possível capturar a complexidade das relações entre variáveis meteorológicas, proporcionando uma modelagem robusta e previsões mais precisas das rajadas máximas de vento em dados de clima em Salvador.