Aprendizagem Automática - Trabalho Prático - Estimating Aerosols

Ruben Farinha nº48329, Gustavo Gomes nº48392, Gonçalo Veríssimo nº48738

Leitura dos dados

```python
import numpy as np
import pandas as pd
from sklearn.metrics import precision_score, recall_score,
accuracy_score

#carregar dados
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
print(train.info())
print(test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11177 entries, 0 to 11176
Data columns (total 24 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 11177 non-null  int64
 1   elevation          11177 non-null  int64
 2   ozone              11177 non-null  int64
 3   NO2                11177 non-null  float64
 4   azimuth            11177 non-null  float64
 5   zenith             11177 non-null  float64
 6   B1                 11177 non-null  float64
 7   B2                 11177 non-null  float64
 8   B3                 11177 non-null  float64
 9   B4                 11177 non-null  float64
 10  B5                 11177 non-null  float64
 11  B6                 11177 non-null  float64
 12  B7                 11177 non-null  float64
 13  B8                 11177 non-null  float64
 14  B8A                11177 non-null  float64
 15  B9                 11177 non-null  float64
 16  B10                11177 non-null  float64
 17  B11                11177 non-null  float64
 18  B12                11177 non-null  float64
 19  water_vapor        11177 non-null  int64
 20  scene              11177 non-null  object
 21  incidence_azimuth  11177 non-null  float64
 22  incidence_zenith   11177 non-null  float64
 23  AOT_550            11177 non-null  float64
```

```
dtypes: float64(19), int64(4), object(1)
memory usage: 2.0+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1973 entries, 0 to 1972
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  1973 non-null   int64
 1   elevation           1973 non-null   int64
 2   ozone               1973 non-null   int64
 3   NO2                 1973 non-null   float64
 4   azimuth             1973 non-null   float64
 5   zenith              1973 non-null   float64
 6   B1                  1973 non-null   float64
 7   B2                  1973 non-null   float64
 8   B3                  1973 non-null   float64
 9   B4                  1973 non-null   float64
 10  B5                  1973 non-null   float64
 11  B6                  1973 non-null   float64
 12  B7                  1973 non-null   float64
 13  B8                  1973 non-null   float64
 14  B8A                 1973 non-null   float64
 15  B9                  1973 non-null   float64
 16  B10                 1973 non-null   float64
 17  B11                 1973 non-null   float64
 18  B12                 1973 non-null   float64
 19  water_vapor         1973 non-null   int64
 20  scene               1973 non-null   object
 21  incidence_azimuth   1973 non-null   float64
 22  incidence_zenith    1973 non-null   float64
dtypes: float64(18), int64(4), object(1)
memory usage: 354.7+ KB
None
```

Tratamento dos Dados

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Definir a coluna id como Index
train = train.set_index('id')
test = test.set_index('id')

# Split data
X = train.drop(['AOT_550'], axis=1)
y = train['AOT_550']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=42)
```

```python
# Conversão dos valores de "scene" para inteiros
le = LabelEncoder()
X_train['scene'] = le.fit_transform(X_train['scene'])
X_test['scene'] = le.transform(X_test['scene'])
test['scene'] = le.transform(test['scene'])

X_train.head(3)
```

```
       elevation  ozone    NO2  azimuth  zenith      B1      B2
B3  \
id

10523         25    308  0.200    165.3    47.5  0.1336  0.0960
0.0629
9586          93    273  0.133     32.2    40.8  0.2408  0.1961
0.1799
2130          59    324  0.199    155.5    49.0  0.1405  0.1084
0.0897

           B4      B5  ...      B8     B8A      B9     B10     B11
B12  \
id                     ...

10523  0.0391  0.0325  ...  0.0229  0.0216  0.0096  0.0023  0.0077
0.0042
9586   0.1768  0.1831  ...  0.2615  0.3079  0.0794  0.0014  0.2478
0.1836
2130   0.0844  0.1023  ...  0.1495  0.1632  0.0895  0.0018  0.2169
0.1580

       water_vapor  scene  incidence_azimuth  incidence_zenith
id
10523         1365      5              166.1               2.8
9586          1562      4              216.1               3.3
2130           382      4              179.4               2.8

[3 rows x 22 columns]
```

```python
X_test.head(3)
```

```
       elevation  ozone    NO2  azimuth  zenith      B1      B2
B3  \
id

9639         631    288  0.218    164.8    54.4  0.1286  0.0999
0.0943
10687        520    360  0.184    128.2    38.6  0.3949  0.4137
0.3652
7022         423    321  0.188    163.6    68.4  0.1569  0.1194
0.0866
```

```
              B4      B5  ...      B8     B8A      B9     B10     B11
B12  \
id                      ...

9639   0.0757  0.1277  ...  0.2795  0.2983  0.1099  0.0015  0.1911
0.1018
10687  0.3719  0.3888  ...  0.4432  0.4325  0.3083  0.0433  0.3978
0.3512
7022   0.0744  0.0753  ...  0.0775  0.0902  0.0409  0.0015  0.0977
0.0788

       water_vapor  scene  incidence_azimuth  incidence_zenith
id
9639           725      3              120.6               3.7
10687          490      8              107.4               6.6
7022           776      1              286.8               9.8

[3 rows x 22 columns]
```

Aplicação de algoritmos

```python
# Aplicação do modelo Random Forest

from sklearn.ensemble import RandomForestRegressor

# Inicializar o modelo Random Forest
RF_Model = RandomForestRegressor(random_state=42)

# Treinar o modelo
RF_Model.fit(X_train, y_train)

# Fazer previsões
RF_y_pred = RF_Model.predict(X_test)

# Calcular as métricas
from sklearn.metrics import mean_squared_error

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, RF_y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_RF = np.sqrt(mse)

print(f'Root Mean Squared Error (RMSE): {rmse_RF}')

Root Mean Squared Error (RMSE): 0.11711433079444178

# Aplicação do modelo KNN

from sklearn.neighbors import KNeighborsRegressor
```

```python
# Inicializar o modelo KNN para regressão
knn_model = KNeighborsRegressor(n_neighbors=7)

# Treinar o modelo
knn_model.fit(X_train, y_train)

# Fazer previsões
knn_y_pred = knn_model.predict(X_test)

# Calcular as métricas
from sklearn.metrics import mean_squared_error


# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, knn_y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_knn = np.sqrt(mse)

print(f'Root Mean Squared Error (RMSE): {rmse_knn}')
```

```
Root Mean Squared Error (RMSE): 0.14934347070257378
```

```python
# Aplicação do modelo Gradient Boosting

from sklearn.ensemble import GradientBoostingRegressor

# Inicializar o modelo Gradient Boosting
gbr = GradientBoostingRegressor()

# Treinar o modelo
gbr.fit(X_train, y_train)

# Fazer previsões
gbr_y_pred = gbr.predict(X_test)

# Calcular as métricas
from sklearn.metrics import mean_squared_error


# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, gbr_y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_gbr = np.sqrt(mse)

print(f'Root Mean Squared Error (RMSE): {rmse_gbr}')
```

```
Root Mean Squared Error (RMSE): 0.12623399681401584
```

```python
# Aplicação do modelo Decision Tree Regressor
```

```python
from sklearn.tree import DecisionTreeRegressor

# Inicializar o modelo Decision Tree
dt = DecisionTreeRegressor()

# Treinar o modelo
dt.fit(X_train, y_train)

# Fazer previsões
dt_y_pred = dt.predict(X_test)

# Calcular as métricas
from sklearn.metrics import mean_squared_error


# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, dt_y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_dt = np.sqrt(mse)

print(f'Root Mean Squared Error (RMSE): {rmse_dt}')
```

```
Root Mean Squared Error (RMSE): 0.17254393199270002
```

```python
# Aplicação do modelo Logistic Regression
from sklearn.linear_model import LinearRegression

# Initialize the Linear Regression model
lr = LinearRegression()

# Train the model
lr.fit(X_train, y_train)

# Make predictions
lr_y_pred = lr.predict(X_test)

# Calcular as métricas
from sklearn.metrics import mean_squared_error


# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, lr_y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_lr = np.sqrt(mse)

print(f'Root Mean Squared Error (RMSE): {rmse_lr}')
```

```
Root Mean Squared Error (RMSE): 0.14258843618471465
```

```python
# Comparação dos Modelos Testados para averiguação de qual o possivel a ser usado
models = pd.DataFrame({
    'Modelo' : ['RandomForestClassifier', 'KNN',
'DecisionTreeClassifier', 'Gradient Boosting Classifier', 'Logistic Regression'],
    'RSME': [rmse_RF, rmse_knn, rmse_dt, rmse_gbr, rmse_lr]})

models.sort_values(by='RSME', ascending=True)
```

	Modelo	RSME
0	RandomForestClassifier	0.117114
3	Gradient Boosting Classifier	0.126234
4	Logistic Regression	0.142588
1	KNN	0.149343
2	DecisionTreeClassifier	0.172544

Estratégia de escolha do modelo utilizado:

A escolha do RandomForestClassifier como modelo final foi baseada na métrica RSME, onde o modelo obteve o valor mais baixo entre os modelos considerados, sendo que os valores mais próximos de zero indicam um desempenho superior.

Resultados

Ao analisar a tabela, fica evidente que o algoritmo Random Forest apresentou os melhores resultados em termos de Root Mean Squared Error (RMSE). Diante desse desempenho superior, optaremos por utilizar este algoritmo para realizar os testes nos dados.

```python
# Lista de valores para n_estimators que deseja testar
n_estimators_values = [100, 300, 500, 800, 1000]

# Dicionário para armazenar os resultados
results = {}

# Loop sobre os diferentes valores de n_estimators
for n_estimators in n_estimators_values:
    # Inicializar o modelo Random Forest com o valor atual de n_estimators
    RF_Model = RandomForestRegressor(n_estimators=n_estimators, random_state=42)

    # Treinar o modelo
    RF_Model.fit(X_train, y_train)

    # Fazer previsões
    RF_y_pred = RF_Model.predict(X_test)

    # Avaliar o desempenho (usando RMSE neste exemplo)
    rmse = np.sqrt(mean_squared_error(y_test, RF_y_pred))
```

```
    # Armazenar os resultados no dicionário
    results[n_estimators] = rmse

# Exibir os resultados
for n_estimators, rmse in results.items():
    print(f'n_estimators={n_estimators}: RMSE={rmse}')
```

```
n_estimators=100: RMSE=0.11711433079444178
n_estimators=300: RMSE=0.11629121231470844
n_estimators=500: RMSE=0.11606342629360607
n_estimators=800: RMSE=0.11593064713238105
n_estimators=1000: RMSE=0.11593425268787118
```

Escolha do n_estimator:

Optámos por escolher n_estimador=500 pois ao comparar o RMSE para 500 estimadores com 800 estimadores, notou-se uma diferença mínima, indicando que o ganho adicional de precisão não justificava o aumento substancial no custo computacional, e, pensámos que a escolha de 500 estimadores tenha sido uma decisão equilibrada.

```
# Lista de valores para max_depth que deseja testar
max_depth_values = [5, 10, 20, 50]

# Lista de valores para max_leaf_nodes que deseja testar
max_leaf_nodes_values = [50, 100, 250, 500, 1000]

# Dicionário para armazenar os resultados
results = {}

# Loop sobre os diferentes valores de max_depth e max_leaf_nodes
for max_depth in max_depth_values:
    for max_leaf_nodes in max_leaf_nodes_values:
        # Inicializar o modelo Random Forest com os valores atuais
        RF_Model = RandomForestRegressor(
            n_estimators=500,
            max_depth=max_depth,
            max_leaf_nodes=max_leaf_nodes,
            random_state=42
        )

        # Treinar o modelo
        RF_Model.fit(X_train, y_train)

        # Fazer previsões
        RF_y_pred = RF_Model.predict(X_test)

        # Avaliar o desempenho (usando RMSE neste exemplo)
        rmse = np.sqrt(mean_squared_error(y_test, RF_y_pred))
```

```
        # Armazenar os resultados no dicionário
        results[(max_depth, max_leaf_nodes)] = rmse

# Exibir os resultados
for (max_depth, max_leaf_nodes), rmse in results.items():
    print(f'max_depth={max_depth}, max_leaf_nodes={max_leaf_nodes}: 
RMSE={rmse}')
```

```
max_depth=5, max_leaf_nodes=50: RMSE=0.13210711467369438
max_depth=5, max_leaf_nodes=100: RMSE=0.13210711467369438
max_depth=5, max_leaf_nodes=250: RMSE=0.13210711467369438
max_depth=5, max_leaf_nodes=500: RMSE=0.13210711467369438
max_depth=5, max_leaf_nodes=1000: RMSE=0.13210711467369438
max_depth=10, max_leaf_nodes=50: RMSE=0.12633414617121 8
max_depth=10, max_leaf_nodes=100: RMSE=0.12317489068546965
max_depth=10, max_leaf_nodes=250: RMSE=0.120812022327903 8
max_depth=10, max_leaf_nodes=500: RMSE=0.12066586745717256
max_depth=10, max_leaf_nodes=1000: RMSE=0.12066582873131025
max_depth=20, max_leaf_nodes=50: RMSE=0.12633521871369655
max_depth=20, max_leaf_nodes=100: RMSE=0.12307291514388707
max_depth=20, max_leaf_nodes=250: RMSE=0.11908499259287132
max_depth=20, max_leaf_nodes=500: RMSE=0.11740016976407742
max_depth=20, max_leaf_nodes=1000: RMSE=0.11652639128861028
max_depth=50, max_leaf_nodes=50: RMSE=0.12633521871369655
max_depth=50, max_leaf_nodes=100: RMSE=0.12307234231789918
max_depth=50, max_leaf_nodes=250: RMSE=0.11909032723757794
max_depth=50, max_leaf_nodes=500: RMSE=0.11739467715367563
max_depth=50, max_leaf_nodes=1000: RMSE=0.11647427410106698
```

Estratégia de escolha dos modelos submetidos no Kaggle:

Optámos por submeter os dois modelos com os parâmetros max_depth=20, max_leaf_nodes=1000 e max_depth=50, max_leaf_nodes=1000 no Kaggle pois essas configurações específicas resultaram nos menores valores de RMSE em comparação com outras combinações de hiperparâmetros testadas.

```
max_depth_values = [20, 50]

for max_depth in max_depth_values:
    RF_Model = RandomForestRegressor(n_estimators=500, 
max_depth=max_depth, max_leaf_nodes=1000, random_state=42)
    RF_Model.fit(X_train, y_train)
    final_pred = RF_Model.predict(test)

    # Resultadoss
    results_df = pd.DataFrame({'id': test.index, 'AOT_550': 
final_pred})
    file_name = f'predicted_max_depth_{max_depth}.csv'
    results_df.to_csv(file_name, index=False)
```

5 Melhores submissões no Kaggle:

- RandomForest (n_estimators=500, max_depth=20, max_leaf_nodes=1000), com score de 0.1396
- RandomForest (n_estimators=500, max_depth=50, max_leaf_nodes=1000), com score de 0.1395
- RandomForest (n_estimators=100 random_state=42), com score de 0.1352
- RandomForest (random_state=42), com score de 0.1355
- KNN (n_neighbors=7), com score de 0.1656

Escolhemos os dois primeiros modelos Random Forest (Max_depth=20 e Max_depth=50) como nossas principais submissões no Kaggle. Embora esses modelos não tenham obtido a melhor pontuação pública na plataforma, optamos por eles porque acreditamos que apresentam uma melhor capacidade de generalização, sendo assim mais adequados para lidar com uma quantidade maior de dados, conforme evidenciado pelos resultados de rmse.