DETEÇÃO AUTOMÁTICA DA OCUPAÇÃO DUM ESPAÇO INTERIOR

Realizado por:

Pedro Ribeiro a37557

Nelson Aguiar a39110

João Gonçalves a43520

- Começamos o nosso trabalho por dar load às seguintes bibliotecas: Pandas, Matplotlib e Seaborn para possibilitar o uso de funções que tornam o desenvolvimento do trabalho mais fácil e eficiente. Carregámos também as Datasets que iremos utilizar.
- Estas bibliotecas proporcionam algoritmos já criados com funções bastantes poderosas, em Machine Learning e manipulação de dados que utilizaremos para alterar as Datasets .

```
# Load Pandas
import pandas as pd
# Load Pyplot
import matplotlib.pyplot as plt
# Load Seaborn
import seaborn as sns

# Load Dataset from Local Directory
dataset = pd.read_csv('C:\IA\dataset.csv')
dataset_naoclassificado = pd.read_csv('C:\IA\dataset_naoclassificado.csv')
```

- Logo depois foi feito o carregamento das primeiras 5 linhas do Dataset que irá treinar a nossa ML com o comando “head()”, dando-nos assim uma ideia dos valores apresentados pelo mesmo.
- Ao usar o método “describe()”, obtemos uma visão mais estatística do Dataset, o que nos permitiu encontrar valores nulos nas amostras de dados.
- Após a deteção de erros, utilizamos o comando “fillna()”, com o commando “mean()”, para preencher os valores nulos encontrados anteriormente com as médias dessas colunas.
- Para finalizar o pre-processameto, foi usado o método “drop()”, para remover as colunas sem relevância, de modo a não influenciar treino da nossa ML.

```
M↓
# Load First 5 rows of Dataset
dataset.head()

M↓
# Examine Dataset
dataset.describe()

M↓
# Fill NaNs with mean values
dataset.temperatura = dataset.temperatura.fillna
(dataset.temperatura.mean())
dataset.luz = dataset.luz.fillna(dataset.luz.mean())

# Drop id_registo from Dataset
dataset = dataset.drop('id_registo', axis = 1)
# Drop id_registo and humidade_absoluta from Dataset_NaoClassificado
dataset_naoclassificado = dataset_naoclassificado.drop(['id_registo',
'humidade_absoluta'], axis = 1)
```

No gráfico apresentado, podemos ver a distribuição da “ocupacao” na relação entre cada uma das variáveis do Dataset.

```
# Show the relation between every variable
sns.pairplot(dataset, hue = 'ocupacao')
plt.show()
```

O próximo passo é particionar o Dataset em 2 partes: Treino e Teste. Para isso atribuímos os atributos ao objeto "X" e o objetivo ao objeto "Y".

```
# Define Features (X) and Target (y)
X = dataset.drop(['ocupacao', 'humidade_absoluta'], axis = 1)
y = dataset['ocupacao']
```

Decidimos usar 25% das amostras totais para não existir qualquer tipo de ajustamento na ML. De forma a obtermos os melhores resultados possíveis, resolvemos normalizar os dados.

```
# Load Train Test Split
from sklearn.model_selection import train_test_split

# Train and Test Split of Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.25, random_state = 1)
```

```
# Load Scaler
from sklearn.preprocessing import StandardScaler

# Scale all Features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Scale dataset_naoclassificado
scaled_naoclassificado = scaler.transform(dataset_naoclassificado)
```

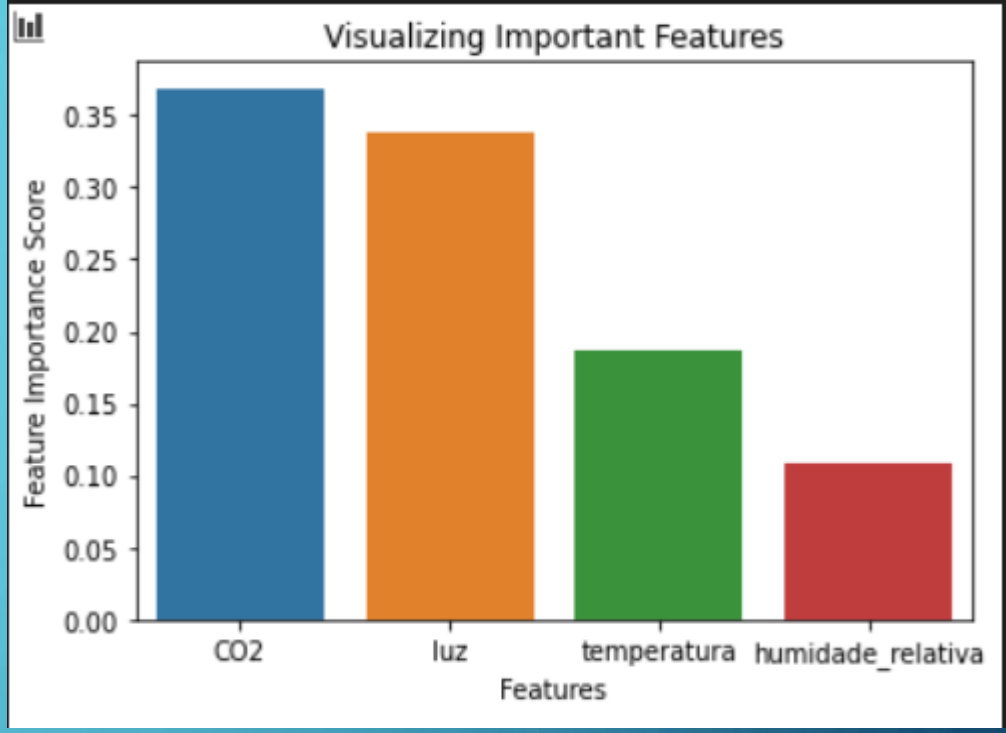
- Em primeiro lugar, criámos uma “Floresta” com 1000 árvores. De seguida, alimentamos a mesma com atributos e objetivo conhecidos. Por fim, fornecemos apenas atributos novos para ver a precisão da mesma.
- Para obter as previsões para o Dataset_NaoClassificado usamos o objeto criado em cima. Após esse primeiro passo, tornamos o objeto “classificacao_estimada” num DataFrame, podendo assim juntar os resultados obtidos ao Dataset_NaoClassificado Original. Finalizamos, exportando o DataFrame final para o ficheiro “classificacao_estimada.csv”.
- De modo a avaliarmos a precisão da nossa ML usamos 2 métricas: classification_report e accuracy_score, onde obtivemos 94.93(3)%.

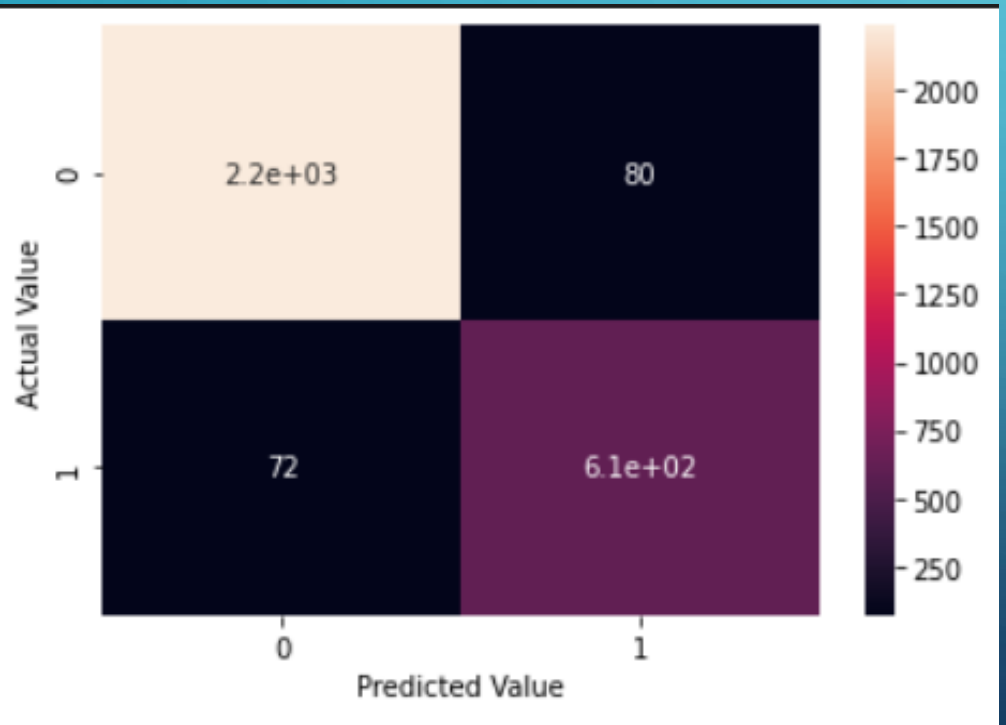
```
# Load Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# Execute Random Forest Classifier Algorithm
clf = RandomForestClassifier(n_estimators = 1000, random_state
= 1234)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# Predict Dataset_NaoClassificado
classificacao_estimada = clf.predict(scaled_naoclassificado)
# Create a DataFrame with Predictions Results
classificacao_estimada_df = pd.DataFrame
(classificacao_estimada)
# Create Final DataFrame with id_registo and ocupacao
dataset_naoclassificado = pd.read_csv
('C:\IA\dataset_naoclassificado.csv')
dataset_naoclassificado = dataset_naoclassificado.drop(
['temperatura', 'luz', 'CO2', 'humidade_relativa',
'humidade_absoluta'], axis = 1)
dataset_naoclassificado['ocupacao'] = classificacao_estimada_df
# Export DataFrame to CSV File
dataset_naoclassificado.to_csv
('C:\IA\classificacao_estimada.csv', index = None)
```

```
# Load Metrics
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score

# Print differents metrics of evaluation
print(classification_report(y_test,y_pred))
print("\nAccuracy Score: ", accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      2315
           1       0.88      0.89      0.89       685

    accuracy                           0.95      3000
   macro avg       0.93      0.93      0.93      3000
weighted avg       0.95      0.95      0.95      3000


Accuracy Score:  0.9493333333333334
```

No gráfico à direita, está representada a importância de cada atributo no Dataset estudado. Foi removido o atributo “humidade_absoluta” pelo facto de ter impactado de forma negativa os nossos resultados previamente.





Neste gráfico, é utilizada outra métrica de avaliação “confusion_matrix”, onde podemos observar os resultados do cruzamento entre os valores reais e os valores previstos pela ML.

Foi usada a classe “RandomizedSearchCV”, com o intuito de descobrir os melhores parâmetros para a “Random Forest Classifier”.

Após várias tentativas, encontramos os seguintes dados, que fizeram aumentar a percentagem de acerto da ML.

“{'n_estimators': 1000, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': None, 'bootstrap': True}”

De salientar que o resultado obtido acima, são os valores definidos por defeito na classe “RandomForestClassifier”, à exceção do parâmetro “n_estimators”.

```
# Load Metrics
#from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Load Random Forest Classifier
#from sklearn.ensemble import RandomForestClassifier

# Execute Random Forest Classifier Algorithm
#rf = RandomForestClassifier(random_state = 1234)

# Load RandomizedSearchCV
#from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
#n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
num = 10)]
# Number of features to consider at every split
#max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
#max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
#max_depth.append(None)
# Minimum number of samples required to split a node
#min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
#min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
#bootstrap = [True, False]

# Create the random grid
#random_grid = {'n_estimators': n_estimators,
               #'max_features': max_features,
               #'max_depth': max_depth,
               #'min_samples_split': min_samples_split,
               #'min_samples_leaf': min_samples_leaf,
               #'bootstrap': bootstrap}

# Random Search of Parameters
#rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
random_grid, n_iter = 100, random_state = 1234, cv = 3, verbose=2,
n_jobs = -1)

# Fit the random search model
#rf_random.fit(X_train, y_train)

#rf_random.best_params_

#best_random = rf_random.best_estimator_

#best_random.score(X_test, y_test)"""
```