Projeto 3
Árvores de decisão

Alunos - Matrícula
Lucas Dalle Rocha - 17/0016641
Matheus Breder Branquinho - 17/0018997

[1]Dep. Ciência da Computação - Universidade de Brasília (UnB)
Introdução à Inteligência Artificial, Turma A, 2/2019
Prof. Díbio

1. Solução do problema

Inicialmente, para a resolução do nosso problema, utilizamos, por padrão, uma função da biblioteca *pandas*, responsável pela leitura dos dados em formato csv. Dessa forma, o programa deve ler o arquivo *selfie_dataset.txt* como um csv e, em seguida, manipular os dados em forma de linhas e colunas, como uma tabela, de modo a deixá-los prontos para serem utilizados durante o treinamento do modelo.

```
# Lê dataset das notas e atributos das fotos e aplica "one-hot encoding"

features = pd.read_csv('selfie_dataset.txt')
features = pd.get_dummies(features)
```

Figure 1. Leitura dos dados.

Após a leitura dos dados, o algoritmo deve escolher o atributo que servirá como parâmetro para todos os outros que queremos prever (antes de realizar o treinamento). Para isso, utilizamos funções da biblioteca *numpy*, de modo a converter os dados disponibilizados em formato csv para uma matriz.

```
# Variável 'labels' são os valores que queremos prever
labels = np.array(features['photo_rate'])
features= features.drop('photo_rate', axis = 1)
feature_list = list(features.columns)
features = np.array(features)
```

Figure 2. Transformação dos dados em array.

Agora, entramos na parte de treinamento e teste dos dados. Primeiramente, trataremos da parte do treinamento. Assim, separamos os dados que serão utilizados para treino daqueles que serão utilizados para testes.

```
# Separa dados para treinamento de modelo
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.25, random_state = np.random)
```

Figure 3. Separação dos dados.

Destarte, para treinamento, utilizamos funções da biblioteca *scikit-learn*, que nos permite esquematizar um número limitado de árvores e, a partir delas, treinar o nosso modelo para ser posto em teste, posteriormente. Dessa forma, conseguimos aplicar os princípios de florestas randômicas, com o intuito de maximizar nossa acurácia durante os testes.

```
# Seta modelo com 1000 árvores de decisão
rf = RandomForestRegressor(n_estimators = 1000, random_state = np.random)
# Treina modelo com as datas separadas
rf.fit(train_features, train_labels)
```

Figure 4. Treinamento do modelo.

Assim, já podemos estabelecer uma ordem de importância dos atributos e armazenar em uma lista de tuplas, com o nome do atributo e sua importância, que é um número racional que varia de 0 a 1 (é importante notar que a soma das importâncias é igual a 1).

```
#       IMPORTANCIA            #
# Lista importância de atributos
importances = list(rf.feature_importances_)
# Cria tupla de atributos e sua importância
feature_importances = [(feature, round(importance, 3)) for feature, importance in zip(feature_list, importances)]
# Ordena do mais importante ao menos importante
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
```

Figure 5. Tuplas de importância.

Além disso, disponibilizamos também, utilizando funções das bibliotecas *pydot* e *graphviz*, uma conversão dos dados após o treinamento do modelo para um arquivo .dot e, consequentemente, a geração de uma imagem em formato .png da árvore correspondente.

```
# Escolhe uma árvore da floresta
tree = rf.estimators_[5]
# Exporta imagem para um arquivo .dot
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = True, precision = 1)
# Cria grafo com arquivo .dot
(graph, ) = pydot.graph_from_dot_file('tree.dot')
# Gera imagem em .png
graph.write_png('tree.png')
```

Figure 6. Geração da árvore em imagem.

2. Análise de resultados

Dessa forma, nosso programa dá opção ao usuário de gerar ou não uma imagem da árvore que foi esquematizada. Porém, por se tratar de uma árvore com altura de no máximo o número de atributos, fica inviável analisá-la. Para isso, utilizamos uma outra função para gerar uma árvore menor, de altura limitada à escolha do programador, que muitas vezes não condiz com os atributos mais importantes.

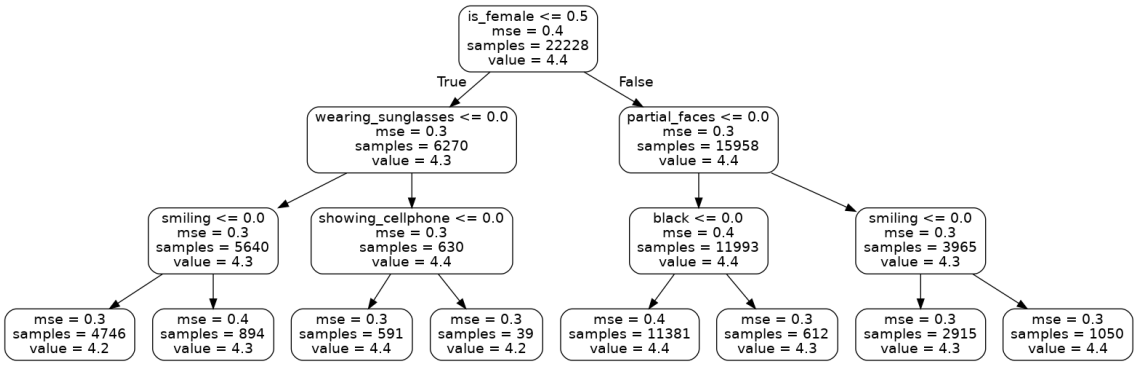

Figure 7. Imagem da árvore reduzida gerada.

Por fim, podemos ver a saída (com as importâncias) e uma função que nos retorna a acurácia, ou seja, o quão certo estão nossos testes realizados com os dados, levando em consideração a quantidade de erros.

```
---------------------------------------------------------------

               Lista de atributos e suas importancias:

Atributo: white                Importancia: 0.057
Atributo: brown_hair           Importancia: 0.05
Atributo: wearing_glasses      Importancia: 0.046
Atributo: smiling              Importancia: 0.045
Atributo: is_female            Importancia: 0.041
Atributo: round_face           Importancia: 0.041
Atributo: oval_face            Importancia: 0.04
Atributo: black_hair           Importancia: 0.04
Atributo: youth                Importancia: 0.038
Atributo: straight_hair        Importancia: 0.038
Atributo: duck_face            Importancia: 0.035
Atributo: wearing_hat          Importancia: 0.035
Atributo: teenager             Importancia: 0.034
Atributo: heart_face           Importancia: 0.033
Atributo: mouth_open           Importancia: 0.033
Atributo: wearing_lipstick     Importancia: 0.033
Atributo: curly_hair           Importancia: 0.033
Atributo: asian                Importancia: 0.031
Atributo: blond_hair           Importancia: 0.029
Atributo: showing_cellphone    Importancia: 0.025
Atributo: using_mirror         Importancia: 0.025
Atributo: using_earphone       Importancia: 0.024
Atributo: tongue_out           Importancia: 0.02
Atributo: partial_faces        Importancia: 0.019
Atributo: black                Importancia: 0.019
Atributo: wearing_sunglasses   Importancia: 0.019
Atributo: middle_age           Importancia: 0.018
Atributo: red_hair             Importancia: 0.016
Atributo: harsh_lighting       Importancia: 0.016
Atributo: dim_lighting         Importancia: 0.016
Atributo: child                Importancia: 0.015
Atributo: braid_hair           Importancia: 0.012
Atributo: braces               Importancia: 0.012
Atributo: frowning             Importancia: 0.011
Atributo: baby                 Importancia: 0.004
Atributo: senior               Importancia: 0.001

Acuracia: 88.66 %.

---------------------------------------------------------------
```

Figure 8. Saída das importâncias e acurácia.

Para uma melhoria na acurácia pode vir a ser útil o aumento de árvores analisadas, o que acarreta em uma melhoria de estimativa na parte de treinamento do modelo. Porém, ao aumentar o número de árvore, a performance do programa é afetada e pode gerar uma demora na execução dessa parte do processo (treinamento).