UNIVERSIDADE FEDERAL FLUMINENSE
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
TCC00297 - INTELIGÊNCIA ARTIFICIAL

Trabalho de Classificação

BEATRIZ DE OLIVEIRA PIEDADE

NITERÓI
2024

Contents

1 Introdução

O objetivo deste trabalho é realizar a classificação do conjunto de dados "Secondary Mushroom", buscando prever se um cogumelo é comestível ou venenoso. O dataset foi analisado com foco na preparação, construção e avaliação de modelos de Machine Learning que maximizem a performance em métricas relevantes.

O conjunto de dados contém 61.068 registros e 21 atributos. A tabela abaixo apresenta uma visão geral dos atributos disponíveis no conjunto:

Nome	**Papel**	**Tipo**	**Valores ausentes**
class	Target	Categorical	no
cap-diameter	Feature	Continuous	no
cap-shape	Feature	Categorical	no
cap-surface	Feature	Categorical	yes
cap-color	Feature	Categorical	no
does-bruise-or-bleed	Feature	Categorical	no
gill-attachment	Feature	Categorical	yes
gill-spacing	Feature	Categorical	yes
gill-color	Feature	Categorical	no
stem-height	Feature	Continuous	no
stem-width	Feature	Continuous	no
stem-root	Feature	Categorical	yes
stem-surface	Feature	Categorical	yes
stem-color	Feature	Categorical	no
veil-type	Feature	Categorical	yes
veil-color	Feature	Categorical	yes
has-ring	Feature	Categorical	no
ring-type	Feature	Categorical	yes
spore-print-color	Feature	Categorical	yes
habitat	Feature	Categorical	no
season	Feature	Categorical	no

1.1 Coleta de dados

O conjunto de dados foi obtido do Repositório de Machine Learning da UCI.

```
# CARREGANDO DADOS
from ucimlrepo import fetch_ucirepo

# importando dataset
dataset = fetch_ucirepo(id=763)

# coletando as informações
data_frame = dataset.data.original
```

O conjunto de dados, assim como todas as tabelas derivadas, está estruturado no formato *DataFrame*, uma poderosa estrutura de dados fornecida pela biblioteca *pandas*. Essa estrutura simplifica a manipulação, análise e pré-processamento dos dados, permitindo operações eficientes.

1.2 Pré-processamento de dados

O conjunto de dados foi tratado para minimizar o impacto de dados nulos, duplicados ou mal formatados na performance dos modelos. O pré-processamento envolveu as seguintes etapas:

1. A remoção de colunas com muitos valores nulos, utilizando o parâmetro *thresh* para evitar a perda excessiva de informações. O valor do *thresh* é dado pela variável *tolerancia*, que garante que colunas com 70% de dados não nulos sejam mantidas.

2. A remoção de dados duplicados para evitar redundância e influências desproporcionais na modelagem.

3. A transformação de variáveis categóricas em valores numéricos, garantindo que o modelo possa interpretar essas variáveis.

```
# TRATANDO DADOS
import pandas
from sklearn.preprocessing import LabelEncoder

# removendo colunas com muitos nulos
tolerancia = len(data_frame) * 0.7
data_frame = data_frame.dropna(axis=1, thresh=tolerancia)

# removendo dados duplicados
```

```
data_frame = data_frame.drop_duplicates()

# convertendo colunas categóricas em valores inteiros
for coluna in data_frame.columns:
if (data_frame[coluna].dtype == type(object)):
 conversor = LabelEncoder()
 data_frame[coluna] = conversor.fit_transform(
                                        data_frame[coluna])
```

Ao término do processo, o conjunto de dados foi reduzido a 60.922 registros tratados e 15 colunas com até 30% de dados faltantes, sendo elas:

Nome	**Papel**	**Tipo**	**Valores ausentes**
class	Target	Categorical	no
cap-diameter	Feature	Continuous	no
cap-shape	Feature	Categorical	no
cap-surface	Feature	Categorical	yes
cap-color	Feature	Categorical	no
does-bruise-or-bleed	Feature	Categorical	no
gill-attachment	Feature	Categorical	yes
gill-color	Feature	Categorical	no
stem-height	Feature	Continuous	no
stem-width	Feature	Continuous	no
stem-color	Feature	Categorical	no
has-ring	Feature	Categorical	no
ring-type	Feature	Categorical	yes
habitat	Feature	Categorical	no
season	Feature	Categorical	no

1.3 Divisão de dados

Dado o tamanho do conjunto de dados resultantes (60.922) e o número de atributos (15), ele será dividido em 70% para treinamento e 30% para teste, utilizando a função *train_test_split()* do pacote *sklearn.model_selection*.

```
# DIVIDINDO O DATASET TRATADO
import numpy
from sklearn.model_selection import train_test_split
```

```
atributos = data_frame.drop(["class"], axis=1)
respostas = data_frame[["class"]]

a_treino, a_teste, r_treino, r_teste = train_test_split(
                                        atributos,
                                        respostas,
                                        test_size=0.3,
                                        random_state=42)

# convertendo de (N, 1) para (N,)
r_treino = numpy.ravel(r_treino)
r_teste = numpy.ravel(r_teste)
```

Além disso, as tabelas resultantes das respostas de treino e teste serão convertidas para o formato *(n_sample,)*, que é o formato esperado pelos classificadores para que possam processar os dados corretamente.

1.4 Treinamento e avaliação do modelo

Para este trabalho, serão utilizados os modelos de aprendizado de máquina: Árvore de Decisão (DT), Random Forest (RF) e Rede Neural Multilayer Perceptron (MLP). Os modelos passarão por ajustes nos parâmetros do classificador e serão avaliados com base na performance dos algoritmos.

Na avaliação da performance, serão utilizadas as métricas *Acurácia* e *F1-Score*, onde a *Acurácia* é dada por

$$Acuracia = \frac{PrevisoesCorretas}{PrevisoesTotais}$$

e o *F1-Score* é calculado como

$$F1 = 2 \cdot \frac{Precisao \cdot Recall}{Precisao + Recall}$$

$$Precisao = \frac{VerdadeirosPositivos}{VerdadeirosPositivos + FalsosPostivos}$$

$$Recall = \frac{VerdadeirosPositivos}{VerdadeirosPositivos + FalsosNegativos}$$

Para cada modelo com parâmetros ajustados, as métricas de desempenho serão calculadas utilizando as funções *accuracy_score* e *f1_score*.

```
from sklearn.metrics import accuracy_score, f1_score

# algoritmo de treinamento do modelo

# calculando métricas
acuracia = accuracy_score(r_teste, r_previsao)
f1 = f1_score(r_teste, r_previsao, average="weighted")
```

2 Árvore de decisão

2.1 Algoritmo utilizado

O algoritmo utilizado para treinar os modelos deste tipo é dado por:

```
# APLICANDO MODELO
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score

# criando a lista de resultados finais
lista_resultados_ad = []

for parametros in lista_parametros_ad:
  # criando classificador
  classificador = DecisionTreeClassifier(
    criterion=parametros["criterion"],
    max_depth=parametros["max_depth"],
    min_samples_split=parametros["min_samples_split"],
    min_samples_leaf=parametros["min_samples_leaf"],
  )

  # treinando o modelo
  classificador.fit(a_treino, r_treino)

  # prevendo respostas
  r_previsao_teste = classificador.predict(a_teste)

  # calculando métricas
  acuracia = accuracy_score(r_teste, r_previsao_teste)
  f1 = f1_score(r_teste, r_previsao_teste, average="weighted")

  # salvando resultados
  lista_resultados_ad.append([parametros["id"], acuracia, f1])
```

2.2 Variações de parâmetros

As variações de parâmetros, em busca da melhor performance, foram:

```python
# CRIANDO PARÂMETROS PARA O MODELO
lista_parametros_ad = [
    {"id": "Var_AD_1", "criterion": "gini", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_AD_2", "criterion": "gini", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_AD_3", "criterion": "gini", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_AD_4", "criterion": "gini", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 10},
    {"id": "Var_AD_5", "criterion": "gini", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_AD_6", "criterion": "gini", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_AD_7", "criterion": "gini", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_AD_8", "criterion": "gini", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 10},
    {"id": "Var_AD_9", "criterion": "entropy", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_AD_10", "criterion": "entropy", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_AD_11", "criterion": "entropy", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_AD_12", "criterion": "entropy", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 10},
    {"id": "Var_AD_13", "criterion": "entropy", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_AD_14", "criterion": "entropy", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_AD_15", "criterion": "entropy", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_AD_16", "criterion": "entropy", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 10},
]
```

✓ 0.0s Python

2.3 Performance

Observando os dados armazenados em *lista_resultados_ad* é possível obter:

	Identificador	Acurácia	F1-score
0	Var_AD_1	0.918367	0.918594
1	Var_AD_2	0.917820	0.918050
2	Var_AD_3	0.918148	0.918376
3	Var_AD_4	0.917820	0.918050
4	Var_AD_5	0.991465	0.991465
5	Var_AD_6	0.988510	0.988513
6	Var_AD_7	0.990316	0.990317
7	Var_AD_8	0.988784	0.988786
8	Var_AD_9	0.871423	0.871752
9	Var_AD_10	0.870767	0.871095
10	Var_AD_11	0.871314	0.871642
11	Var_AD_12	0.870548	0.870878
12	Var_AD_13	0.994419	0.994419
13	Var_AD_14	0.991574	0.991575
14	Var_AD_15	0.993763	0.993763
15	Var_AD_16	0.991738	0.991739

3 Random Forest

3.1 Algoritmo utilizado

O algoritmo utilizado para treinar os modelos deste tipo é dado por:

```
# APLICANDO MODELO
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

# criando a lista de resultados finais
lista_resultados_rf = []

for parametros in parametros_rf:
 # criando classificador
 classificador = RandomForestClassifier(
  criterion=parametros["criterion"],
  max_depth=parametros["max_depth"],
  min_samples_split=parametros["min_samples_split"],
  min_samples_leaf=parametros["min_samples_leaf"],
  n_estimators=100,
  random_state=42
 )

 # treinando o modelo
 classificador.fit(a_treino, r_treino)

 # prevendo respostas
 r_previsao = classificador.predict(a_teste)

 # calculando métricas
 acuracia = accuracy_score(r_teste, r_previsao)
 f1 = f1_score(r_teste, r_previsao, average="weighted")

 # salvando resultados
 lista_resultados_rf.append([parametros["id"], acuracia, f1])
```

3.2 Variações de parâmetros

As variações de parâmetros, em busca da melhor performance, foram:

```python
# CRIANDO PARÂMETROS PARA OS MODELOS
parametros_rf = [
    {"id": "Var_RF_1", "criterion": "gini", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_RF_2", "criterion": "gini", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_RF_3", "criterion": "gini", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_RF_4", "criterion": "gini", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 10},
    {"id": "Var_RF_5", "criterion": "gini", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_RF_6", "criterion": "gini", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_RF_7", "criterion": "gini", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_RF_8", "criterion": "gini", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 10},
    {"id": "Var_RF_9", "criterion": "entropy", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_RF_10", "criterion": "entropy", "max_depth": 10, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_RF_11", "criterion": "entropy", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_RF_12", "criterion": "entropy", "max_depth": 10, "min_samples_split": 20, "min_samples_leaf": 10},
    {"id": "Var_RF_13", "criterion": "entropy", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 5},
    {"id": "Var_RF_14", "criterion": "entropy", "max_depth": 20, "min_samples_split": 10, "min_samples_leaf": 10},
    {"id": "Var_RF_15", "criterion": "entropy", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 5},
    {"id": "Var_RF_16", "criterion": "entropy", "max_depth": 20, "min_samples_split": 20, "min_samples_leaf": 10}
]
```

✓ 0.0s Python

3.3 Performance

Observando os dados armazenados em *lista_resultados_rf* é possível obter:

	Identificador	Acurácia	F1-score
0	Var_RF_1	0.990425	0.990424
1	Var_RF_2	0.990918	0.990916
2	Var_RF_3	0.990699	0.990699
3	Var_RF_4	0.990918	0.990916
4	Var_RF_5	0.998906	0.998906
5	Var_RF_6	0.998523	0.998523
6	Var_RF_7	0.998960	0.998960
7	Var_RF_8	0.998523	0.998523
8	Var_RF_9	0.988401	0.988400
9	Var_RF_10	0.985282	0.985285
10	Var_RF_11	0.990042	0.990044
11	Var_RF_12	0.985282	0.985285
12	Var_RF_13	0.998960	0.998960
13	Var_RF_14	0.998304	0.998304
14	Var_RF_15	0.998851	0.998851
15	Var_RF_16	0.998304	0.998304

4 Rede Neural Multilayer Perceptron

4.1 Algoritmo utilizado

O algoritmo utilizado para treinar os modelos deste tipo é dado por:

```
# APLICANDO MODELO
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, f1_score

# criando a lista de resultados finais
lista_resultados_mlp = []

for parametros in lista_parametros_mlp:
 # criando classificador
 classificador = MLPClassifier(
  hidden_layer_sizes=parametros["hidden_layer_sizes"],
  activation=parametros["activation"],
  max_iter=500,
 )

 # treinando o modelo
 classificador.fit(a_treino, r_treino)

 # prevendo respostas
 r_previsao = classificador.predict(a_teste)

 # calculando métricas
 acuracia = accuracy_score(r_teste, r_previsao)
 f1 = f1_score(r_teste, r_previsao, average="weighted")

 # salvando resultados
 lista_resultados_mlp.append([parametros["id"], acuracia, f1])
```

4.2 Variações de parâmetros

As variações de parâmetros, em busca da melhor performance, foram:

```python
# CRIANDO PARÂMETROS PARA OS MODELOS

lista_parametros_mlp = [
    {"id": "Var_MLP_1", "hidden_layer_sizes": (50,), "activation": "relu"},
    {"id": "Var_MLP_2", "hidden_layer_sizes": (50,), "activation": "tanh"},
    {"id": "Var_MLP_3", "hidden_layer_sizes": (50,), "activation": "logistic"},
    {"id": "Var_MLP_4", "hidden_layer_sizes": (100,), "activation": "relu"},
    {"id": "Var_MLP_5", "hidden_layer_sizes": (100,), "activation": "tanh"},
    {"id": "Var_MLP_6", "hidden_layer_sizes": (100,), "activation": "logistic"},
    {"id": "Var_MLP_7", "hidden_layer_sizes": (150,), "activation": "relu"},
    {"id": "Var_MLP_8", "hidden_layer_sizes": (150,), "activation": "tanh"},
    {"id": "Var_MLP_9", "hidden_layer_sizes": (150,), "activation": "logistic"},
    {"id": "Var_MLP_10", "hidden_layer_sizes": (50, 50), "activation": "relu"},
    {"id": "Var_MLP_11", "hidden_layer_sizes": (50, 50), "activation": "tanh"},
    {"id": "Var_MLP_12", "hidden_layer_sizes": (50, 50), "activation": "logistic"},
    {"id": "Var_MLP_13", "hidden_layer_sizes": (100, 100), "activation": "relu"},
    {"id": "Var_MLP_14", "hidden_layer_sizes": (100, 100), "activation": "tanh"},
    {"id": "Var_MLP_15", "hidden_layer_sizes": (100, 100), "activation": "logistic"},
    {"id": "Var_MLP_16", "hidden_layer_sizes": (150, 150), "activation": "relu"},
    {"id": "Var_MLP_17", "hidden_layer_sizes": (150, 150), "activation": "tanh"},
    {"id": "Var_MLP_18", "hidden_layer_sizes": (150, 150), "activation": "logistic"}
]
```

✓ 0.0s Python

4.3 Performance

Observando os dados armazenados em *lista_resultados_mlp* é possível obter:

	Identificador	Acurácia	F1-score
0	Var_MLP_1	0.990808	0.990806
1	Var_MLP_2	0.996881	0.996882
2	Var_MLP_3	0.997757	0.997757
3	Var_MLP_4	0.997538	0.997538
4	Var_MLP_5	0.998796	0.998796
5	Var_MLP_6	0.998851	0.998851
6	Var_MLP_7	0.998030	0.998031
7	Var_MLP_8	0.998632	0.998632
8	Var_MLP_9	0.999070	0.999070
9	Var_MLP_10	0.997811	0.997812
10	Var_MLP_11	0.997647	0.997647
11	Var_MLP_12	0.994091	0.994093
12	Var_MLP_13	0.998960	0.998960
13	Var_MLP_14	0.997428	0.997428
14	Var_MLP_15	0.996334	0.996333
15	Var_MLP_16	0.999070	0.999070
16	Var_MLP_17	0.999289	0.999289
17	Var_MLP_18	0.997374	0.997374

5 Conclusão

Observando os modelos de classificação gerados:

Árvore de Decisão: o melhor resultado foi obtido com o conjunto de parâmetros *Var_AD_13* (critério de divisão "entropy", profundidade máxima 20, no mínimo 10 amostras necessárias para dividir um nó e no mínimo 5 amostras necessárias para dividir um nó folha) que possui Acurácia e F1-Score de 0.994419.

Random Forest: o melhor resultado foi obtido nos conjuntos de parâmetros *Var_RF_07* (critério de divisão "gini", profundidade máxima 20, no mínimo 20 amostras necessárias para dividir um nó e no mínimo 5 amostras necessárias para dividir um nó folha) e *Var_RF_13* (critério de divisão "entropy", profundidade máxima 20, no mínimo 10 amostras necessárias para dividir um nó e no mínimo 5 amostras necessárias para dividir um nó folha) quem possuem Acurácia e F1-Score de 0.998960.

Rede Neural Multilayer Perceptron: o melhor resultado foi obtido com o conjunto de parâmetros *Var_MLP_17* (duas camadas ocultas com 150 neurônios cada e função de ativação "tanh") que possui Acurácia e F1-Score de 0.999289.

Levando em consideração somente as métricas Acurácia e F1-Score, o melhor modelo para colocar em produção dentro do domínio do dataset "Secondary Mushroom" é o modelo *Var_MLP_17* de Rede Neural Multilayer Perceptron.

Contudo, ao observarmos os tempos médios de execução de cada modelo, é possível perceber que o tipo Multilayer Perceptron apresenta um desempenho mais lento em comparação ao Random Forest e à Árvore de Decisão.

Tipo	**Variações**	**Tempo total**	**Tempo médio**
Árvore de decisão	16	4,7s	0,29s
Random Forest	16	72,5s	4,53s
Multilayer Perceptron	18	846,3s	47,01s

Desta forma, o modelo que apresenta o melhor equilíbrio entre desempenho e tempo de execução é o Random Forest, configurado com os parâmetros *Var_RF_07* ou *Var_RF_13*.