Universidade Federal do Maranhão - UFMA
Departamento de Informática - DEINF
Inteligência Artificial
Prof°. Dr. Tiago Bonini Borchartt

Alan Marques Bezerra
Jose Emanuel Passos Barros
Karla Felicia Carvalho Da Silva

# RELATÓRIO

# IMPLEMENTAÇÃO DO ALGORITMO DE ARVORE DE DECISÃO

São Luís - MA
2022

## 1. Introdução

O seguinte relatório de Implementação do Algoritmo de Arvore de Decisão do curso de Ciência da Computação da Universidade Federal do Maranhão (UFMA), tem como objetivo implementar um classificador para gerar um **modelo de classificação** e classificar os 30 atletas que não possuem classe atribuída. Para isso, a metodologia utilizada foi a utilização do algoritmo de Arvore de Decisão, Implementado pela biblioteca sklearn.tree.DecisionTreeClassifier" .

## 2. Arvore de Decisão

Uma árvore de decisão é um algoritmo de aprendizado de máquina supervisionado que é utilizado para classificação e para regressão. Isto é, pode ser usado para prever categorias discretas (sim ou não, por exemplo) e para prever valores numéricos (o valor do lucro em reais).

Assim como um fluxograma, a árvore de decisão estabelece nós (decision nodes) que se relacionam entre si por uma hierarquia. Existe o nó-raiz (root node), que é o mais importante, e os nós-folha (leaf nodes), que são os resultados finais. No contexto de machine learning, o raiz é um dos atributos da base de dados e o nó-folha é a classe ou o valor que será gerado como resposta.

## 3. Algoritmo Desenvolvido Para Realizar o Trabalho e Passo a Passo

(Obs: O código foi feito utilizando o Jupyter Notebook.)

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:
#O dataset é transformado em um dataframe utilizando o pandas
import pandas as pd
base = pd.read_csv('jogadores.csv',sep = ';')
base.head()


# In[5]:
#O dataset com os dados para classificar é transformado em um dataframe
utilizando o pandas
base_validacao = pd.read_csv('validacao.csv',sep = ';')
base_validacao.head()
```

```
# In[6]:
#Vizualizamos os valores da classe que servirão para classificar
base['Classe'].unique()


# In[7]:
#vizualizar a quantidade de registro dessa base de dados
base.shape


# In[8]:
#Separamos os dados de suas determinadas classes
x_previsores = base.iloc[:,0:9].values
y_classe = base.iloc[:,9].values
x_valida = base_validacao.iloc[:,0:9].values
y_valida = base_validacao.iloc[:,9].values


# In[9]:
#Vizualizamos os valores
x_previsores


# In[10]:
#Vizualizamos os valores
y_classe


# In[11]:
#Fazemos um preprocessamento escalonando os dados
from sklearn.preprocessing import StandardScaler
scaler_x = StandardScaler()
x_previsores = scaler_x.fit_transform(x_previsores)

scaler_x2 = StandardScaler()
x_valida = scaler_x2.fit_transform(x_valida)


# In[12]:
#Vizualizamos os valores escalonados
x_previsores[0], x_valida[0]


# In[13]:
#Fazemos a divisão entre base de dados de treinamento e base de teste
from sklearn.model_selection import train_test_split
x_treinamento, x_teste, y_treinamento, y_teste = train_test_split(x_previsores,
y_classe, test_size = 0.3)
```

```
# In[14]:
#vizualizar a quantidade de registro dessa base de dados
x_treinamento.shape


# In[15]:
#vizualizar a quantidade de registro dessa base de dados
y_treinamento.shape

# In[16]:
#vizualizar a quantidade de registro dessa base de dados
x_teste.shape

# In[17]:
#vizualizar a quantidade de registro dessa base de dados
y_teste.shape


# In[18]:
#Criamos a Arvore importando a implementação da biblioteca
sklearn.tree.DecisionTreeClassifier
# E feito o Treinamento
from sklearn.tree import DecisionTreeClassifier
classificador = DecisionTreeClassifier(random_state=0)
classificador.fit(x_treinamento , y_treinamento)


# In[19]:
#Colocamos o classificador pra fazer as prediçoes no conjunto de teste
previsoes = classificador.predict(x_teste)


# In[20]:
#Vizualização das predições
previsoes


# In[21]:
#Vizualizamos o conjunto com as respostas para ver se ele acertou
y_teste


# In[22]:
#Metricas de avaliação
from sklearn.metrics import classification_report # metricas de validação

print(classification_report(y_teste, previsoes))
```

```
# In[23]:
#Submetemos o classificador ao conjunto de validaçao
previsoes_2 = classificador.predict(x_valida)
previsoes_2


# In[24]:
#vizualizamos as prediçoes feitas pelo classificador
base_validacao.iloc[:,9] = previsoes_2
base_validacao
```

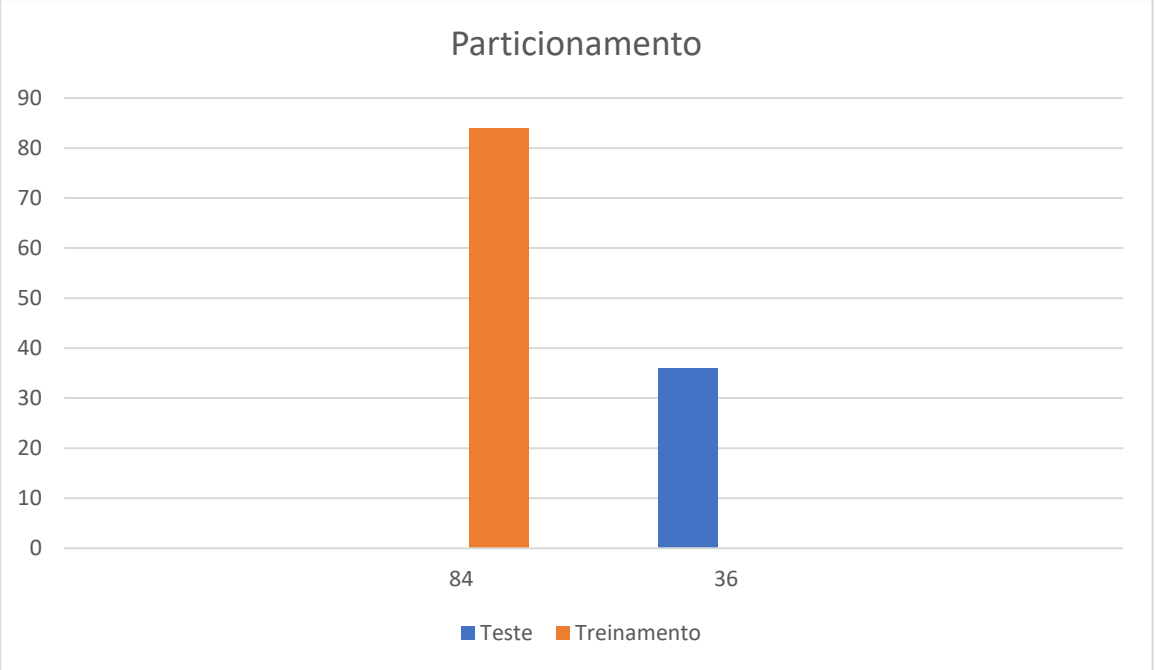
4. **Parâmetros Utilizados No Classificador**

Os parametros utilizados foram:
*criterion='gini',*
*splitter='best',*
*max_depth=None,*
*min_samples_split=2,*
*min_samples_leaf=1,*
*min_weight_fraction_leaf=0.0,*
*max_features=None,*
*random_state=0,*
*max_leaf_nodes=None,*
*min_impurity_decrease=0.0,*
*class_weight=None,*
*ccp_alpha=0.0*

Obs: Todos os parametros são padrões, com excessão do *random_state = 0, para que o estimador* obtenha um comportamento determinístico durante as divisões permutando cada feature aleatoriamente.

5. **Particionamento Para Teste e Treino**

A base de dados “jogadores.csv” contem os dados de 120 jogadores, para o treinamento do algoritmo foram utilizados 70% dos dados na base de dados(84 jogadores). Para testes foi utilizado 30% de dados na base de dados(36 jogadores).




6. **Métricas de avaliação do teste**

No Python é possível analisar todas as métricas de uma vez utilizando o comando *classification_report()*. Na primeira linha e primeira coluna temos o valor da matriz de confusão igual a 93%.

Na primeira linha e segunda coluna temos a Especificidade igual a 86%. Na segunda linha e primeira coluna temos o Precision igual a 83% e na segunda linha e segunda coluna temos o Recall igual a 90%.

As outras métricas é o F1-Score para cada classe e a acurácia e suas variações.

| | **Precision** | **Recall** | **F1 Score** | **Quantidade** |
|---|---|---|---|---|
| Atacante | **0.95** | **1.00** | **0.98** | **20** |
| Defensor | **1.00** | **0.94** | **0.97** | **16** |
| | | | | |
| Acurácia | | | **0.97** | **36** |

| | | | | |
|---|---|---|---|---|
| Macro avg | **0.98** | **0.97** | **0.97** | **36** |
| Weighted avg | **0.97** | **0.97** | **0.97** | **36** |

## 7. Resultado Para os 30 Casos Não Rotulados

| Id | Idade | Altura | Tecnica | Passe | Chute | Forca | Velocidade | Drible | Classe |
|---|---|---|---|---|---|---|---|---|---|
| 201 | 19 | 186 | 71 | 67 | 52 | 77 | 75 | 65 | Defensor |
| 202 | 19 | 175 | 65 | 63 | 66 | 60 | 82 | 70 | Atacante |
| 203 | 20 | 181 | 64 | 64 | 67 | 70 | 67 | 68 | Atacante |
| 204 | 20 | 170 | 69 | 66 | 70 | 62 | 76 | 75 | Atacante |
| 205 | 21 | 176 | 66 | 61 | 68 | 65 | 75 | 72 | Atacante |
| 206 | 22 | 171 | 64 | 65 | 66 | 60 | 78 | 71 | Atacante |
| 207 | 22 | 176 | 70 | 72 | 70 | 64 | 77 | 77 | Atacante |
| 208 | 22 | 188 | 72 | 69 | 50 | 76 | 75 | 65 | Defensor |
| 209 | 22 | 180 | 68 | 74 | 67 | 71 | 72 | 71 | Atacante |
| 210 | 22 | 177 | 67 | 68 | 69 | 65 | 77 | 73 | Atacante |
| 211 | 23 | 185 | 77 | 79 | 65 | 79 | 75 | 79 | Atacante |
| 212 | 23 | 178 | 66 | 68 | 65 | 65 | 80 | 76 | Atacante |
| 213 | 23 | 180 | 68 | 65 | 51 | 77 | 71 | 61 | Defensor |
| 214 | 23 | 170 | 68 | 66 | 68 | 62 | 75 | 74 | Atacante |
| 215 | 24 | 182 | 79 | 73 | 58 | 77 | 77 | 65 | Defensor |
| 216 | 24 | 178 | 62 | 68 | 46 | 70 | 65 | 63 | Defensor |
| 217 | 24 | 189 | 65 | 65 | 55 | 70 | 69 | 58 | Defensor |
| 218 | 25 | 181 | 72 | 67 | 58 | 77 | 81 | 73 | Defensor |
| 219 | 25 | 185 | 69 | 65 | 58 | 75 | 72 | 68 | Defensor |
| 220 | 25 | 178 | 70 | 67 | 63 | 66 | 83 | 77 | Defensor |
| 221 | 26 | 174 | 67 | 64 | 65 | 66 | 82 | 70 | Atacante |
| 222 | 26 | 169 | 78 | 78 | 73 | 67 | 83 | 80 | Atacante |
| 223 | 27 | 190 | 70 | 64 | 58 | 81 | 74 | 60 | Defensor |
| 224 | 27 | 183 | 70 | 66 | 55 | 81 | 67 | 65 | Defensor |
| 225 | 27 | 184 | 78 | 68 | 76 | 68 | 92 | 83 | Atacante |
| 226 | 28 | 178 | 72 | 64 | 79 | 74 | 73 | 69 | Atacante |
| 227 | 31 | 189 | 70 | 66 | 59 | 84 | 71 | 63 | Defensor |
| 228 | 33 | 187 | 71 | 63 | 55 | 82 | 73 | 64 | Defensor |
| 229 | 37 | 188 | 72 | 63 | 54 | 84 | 70 | 60 | Defensor |
| 230 | 37 | 182 | 72 | 65 | 59 | 84 | 67 | 63 | Defensor |

**Considerações Finais**

O Algoritmo é bastante eficiente Para  bases de dados não muito grandes, consegue processar muito bem um conjunto de médio porte, mas se houver muitos dados e juntamente com muitos dados desnecessários isso irá implicar numa alta demanda de processamento.

Quando são entregues poucos dados para treinamento o algoritmo tem pouco acertos, pois ele tende a chutar a resposta. Recomenda-se usar no mínimo 10% dos dados do conjunto de dados para treinamento, caso o contrário, o algoritmo não irá aprender e vai chutar as respostas, ou definirá todos como DEFENSOR ou Todos como ATACANTE.

**Bibliografia:**


https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

https://pandas.pydata.org/docs/user_guide/index.html

https://numpy.org/doc/stable/reference/generated/numpy.unique.html