Relatório de Atividade Prática

Árvores de Decisão

IA1s2024

Aluno
Rennam Faria - RA 123456

São José dos Campos
2024

1 Estudo comparativos de Parâmetros das Árvores de Decisão

1.1 Criterion

Colocar gráficos e discutir resultados;

1.2 Splitter

Colocar gráficos e discutir resultados;

1.3 Max depth

Colocar gráficos e discutir resultados;

1.4 Min sample split

Colocar gráficos e discutir resultados;

2 Etapas de Mineração de Dados

Nesta seção você explicar as estratégias adotadas em cada uma das etapas da MD.

2.1 Conhecimento do Domínio

2.2 Pré-processamento

2.2.1 Valores faltantes

2.2.2 Alteração de tipo de dados

2.2.3 Remoção de atributos

2.3 Extração de Padrões

2.4 Pós-Processamento

3 Conclusão

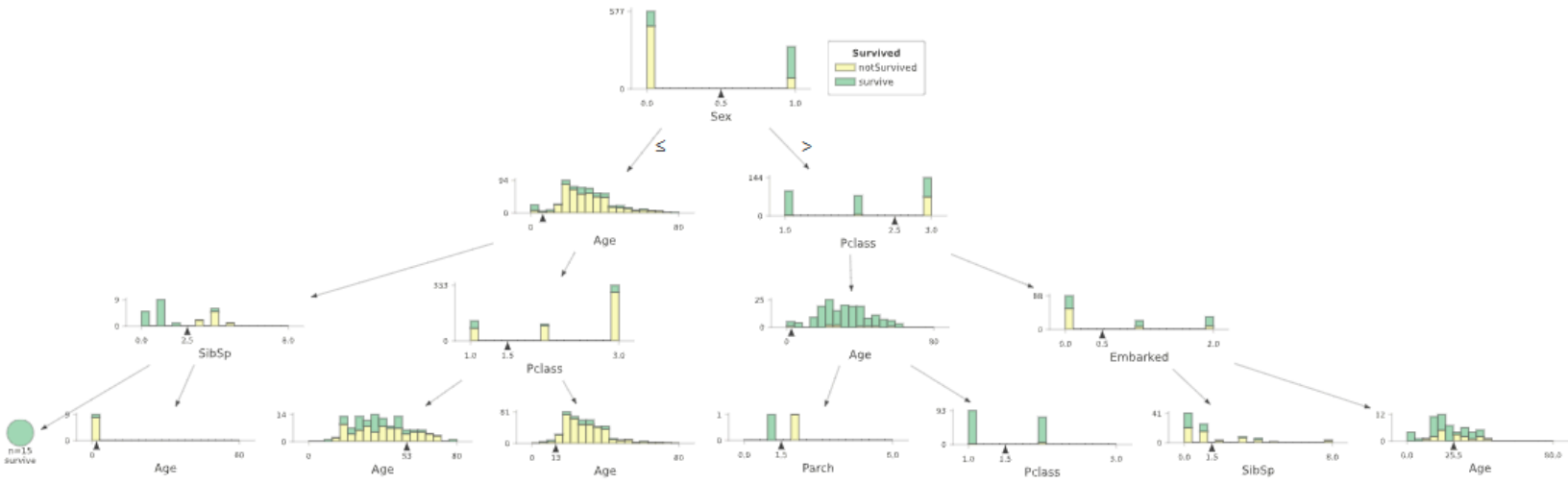
1 Estudo comparativos de Parâmetros das Árvores de Decisão

Todos os testes foram executados utilizando o algoritmo original dado, sem alteração, o parâmetro "random_state=0" dando uma semente inicial fixa. Foi adotada para garantir a consistência na construção da árvore em todas as repetições do teste, evitando variações.

Os modelos e gráficos utilizados para a análise dos pós-processamentos foram de análise de ramificações das árvores e suas folhas. Os gráficos mostrados foram os que tiveram maiores mudanças entre os testes comparados, os gráficos não mostrados não tiveram uma variação muito relevante

Um dos modelos que utilizaremos será o de análise de ramificação, que pega o melhor atributo para a separação e mostra seus dados detalhadamente:

Figura 1 - Exemplo de Modelo para as ramificações



1.1 Criterion

Gini test:

- **balanced_accuracy_score** = 0.7758458646616542
- **Code:**
  DecisionTreeClassifier(criterion='gini', random_state=0)

Entropy test:

- **balanced_accuracy_score =** 0.7805451127819549
- **Code:**
  DecisionTreeClassifier(criterion='entropy', random_state=0)

Comparação:

Entre os dois códigos o “entropy” possuiu uma leve melhora na sua precisão de acerto, isso por conta que suas folhas foram bem mais balanceados e menores que do teste com o “gini”(Figura 2 e 3), sendo que a altura máxima pelo gini foi de 20 e do entropy foi de 19, não é uma diferença muito alta mais o suficiente em aumentar levemente sua precisão, além de que a disposição dos ramos também foi outro diferencial(Figura 4 e 5)

Figura 2 - “Gini” histograma de tamanho das folha

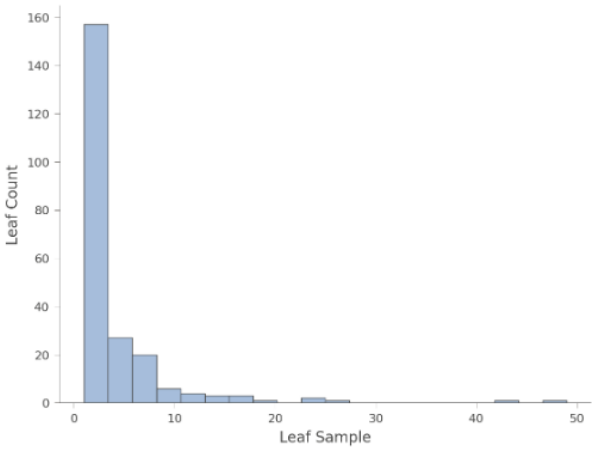


Figura 3 - “Entropy” histograma de tamanho das folhas

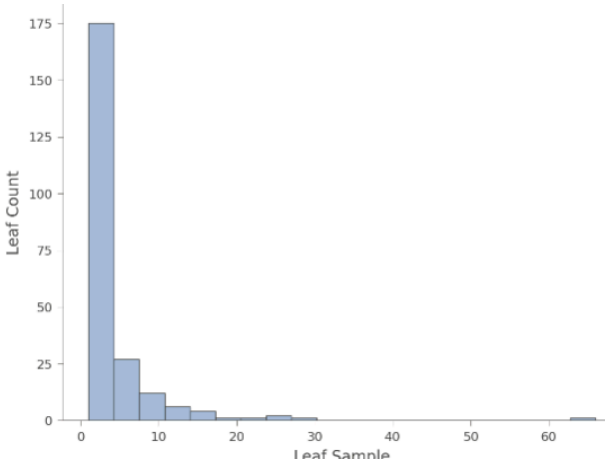


Figura 4 - “Gini” árvore gerada

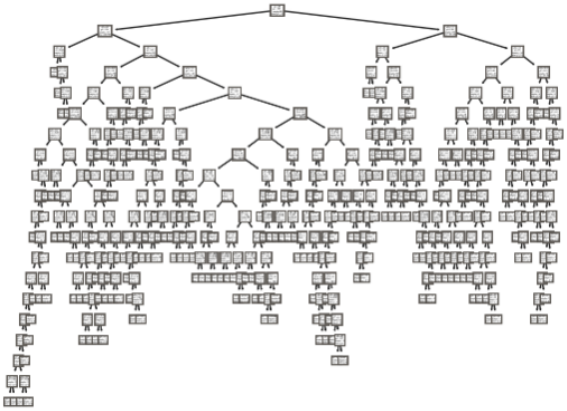
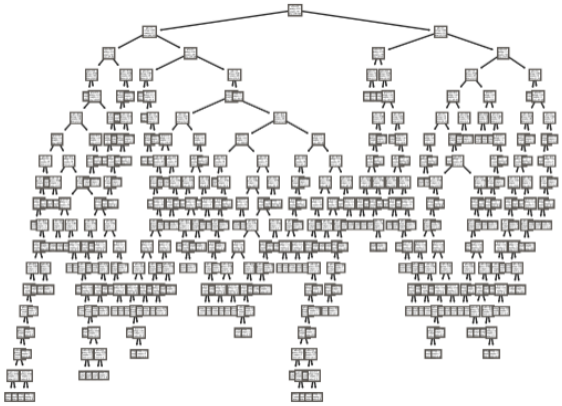


Figura 5 - “Entropy” árvore gerada



1.2 Splitter

Best test:

- **balanced_accuracy_score** = 0.7758458646616542
- **Code:**
  DecisionTreeClassifier(splitter='best', random_state=0)

Random test:

- **balanced_accuracy_score** = 0.7866541353383458
- **Code:**
  DecisionTreeClassifier(splitter='random', random_state=0)

Comparação:

Comparando ambos os códigos, o “random” conseguiu um pequeno resultado na sua precisão de acerto, como no teste passado, uma pequena melhora. E a possível motivo de melhorar foi por causa da sua distribuição melhor dos ramos,que tende mais para o centro das ramificações(Figura 6 e 7) e podemos ver melhor o que está ocorrendo para essa decisão pelo outro modelo abaixo(Figura 8 e 9), analisando podemos observar que o “ëntropy” consegue fazer uma separação muito melhor de

mortos e vivos muito melhor do que “best”, principalmente em “PClass”(Figura 9) que possui muitos dados para separar e consegue separa uma bela parte.

Figura 6 - “best” árvore gerada

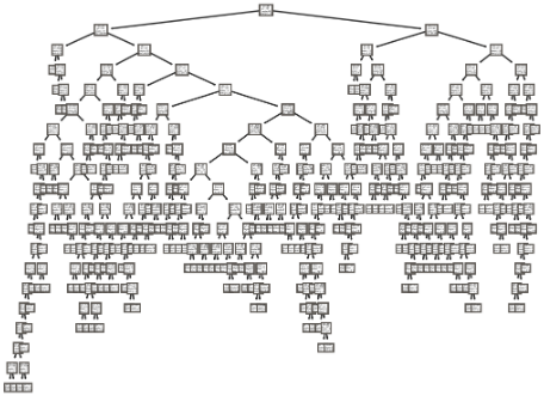


Figura 7 - “random” árvore gerada

Figura 8 - “best” árvore gerada

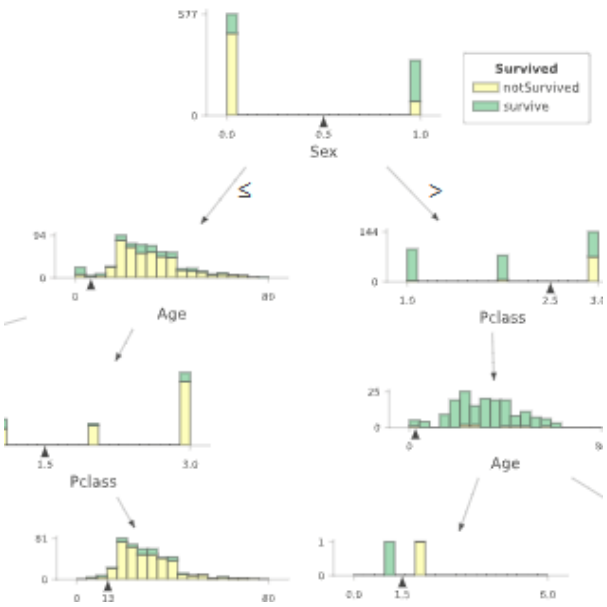
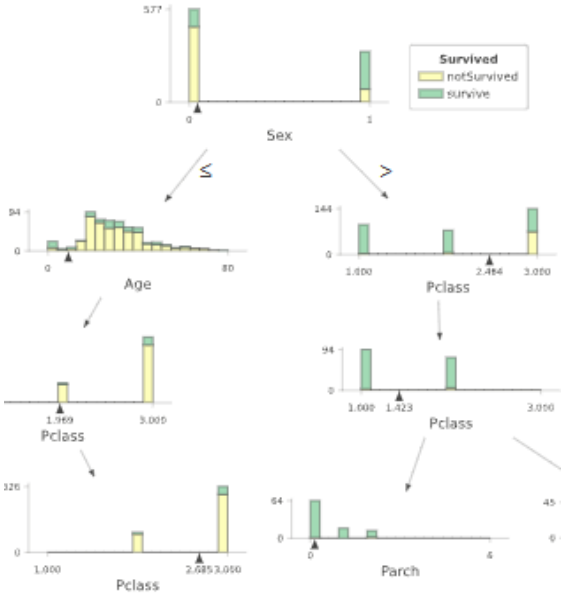


Figura 9 - “random” árvore gerada



1.3 Max depth

Max depth (2):

- **balanced_accuracy_score** = 0.7481203007518797
- **Code:**
  DecisionTreeClassifier(max_depth=2, random_state=0)

Max depth (4):

- **balanced_accuracy_score** = 0.8472744360902256
- **Code:**
  DecisionTreeClassifier(max_depth=4, random_state=0)

Comparação:

Agora, com uma limitação de altura máxima podemos conseguir analisar melhor esses gráficos e modelos, nesse caso o "max_depth=4" possuiu um resultado muito melhor do que o outro, o principal motivo é pela altura, com uma altura muito limitada ele não possui ramificações suficientes para gerar os resultados corretos, observe o gráfico "max_depth=2" de importância dos atributo para sobrevivência ou não, e perceba que ela é feita apenas pelos atributos "Sex" e "PClass" (Figura 10) que é bem limitado para encontrar uma grande taxa de acertos apenas com eles, mas se olharmos o gráfico de "max_depth=4"(Figura 11) podemos ver que ele utiliza um pouco mais de atributos, não todos, mas o suficiente para gerar os principais critérios de separação e não criar um algoritmo muito específico com uma profundidade muito alto, algo que acontecia nos testes anteriores

Figura 10 - "max_depth=2" dado de importância

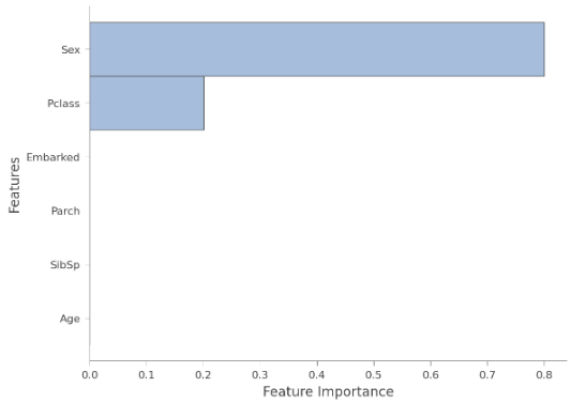
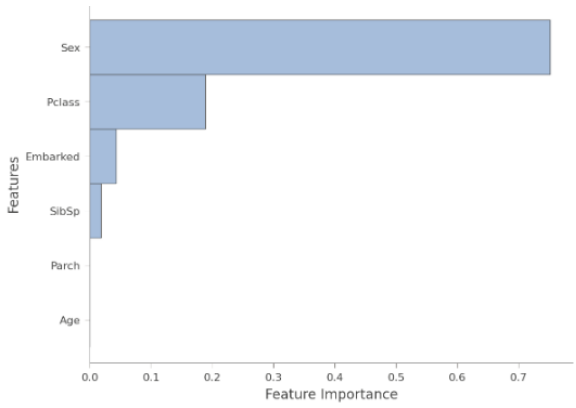


Figura 11 - "max_depth=4" dado de importância



1.4 Min sample split

Min sample (2):

- **balanced_accuracy_score =** 0.7758458646616542
- **Code:**
  DecisionTreeClassifier(min_samples_split=2, random_state=0)

Min sample (10):

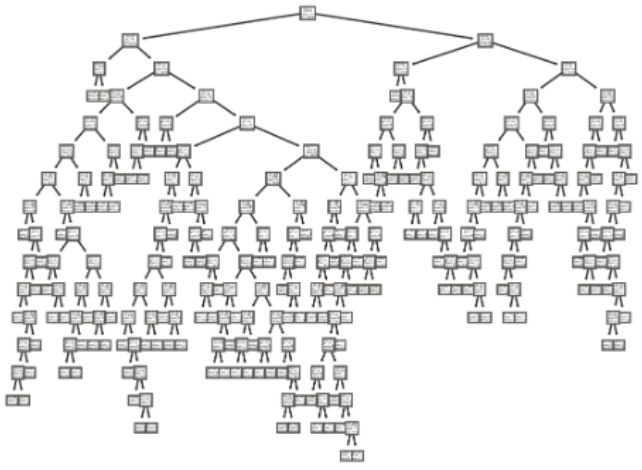
- **balanced_accuracy_score =** 0.8171992481203008
- **Code:**
  DecisionTreeClassifier(min_samples_split=10, random_state=0)

Comparação:

Note que o resultado do teste é igual ao “gini” e ao “best”, pois eles são o mesmo teste, por causa que eles são o padrão do código utilizado, então não há nada de novo vindo dele, mas para o “min_samples_split=10” gera resultados melhores que o padrão (Figura 12).

O principal motivo para que o “min_samples_split=10” obtenha um resultado melhor que o anterior é por causa que ele não irá criar uma árvore gigante com comparações praticamente inúteis para se separar de situações específicas, pois quanto mais específica e grande a árvore mais chances de errar, pois ela não saberá tratar dados gerais

Figura 12 - “min_samples_split=10” árvore gerada



2 Etapas de Mineração de Dados

Primeiramente, adotaremos os processos padrões de análise iniciais do algoritmo, buscando alcançar uma acurácia balanceada superior à do algoritmo original fornecido. Nesses passos iniciais, faremos uma análise da base de dados existente pelo o desafio do Titanic, identificando quais informações podemos extrair dela e por fim os resultados. A partir daí, iniciaremos o processo de manipulação dos dados, transformando-os e ajustando-os conforme necessário, até que possamos construir nossa árvore de decisões e analisar os resultados.

2.1 Conhecimento do Domínio

Com a base de dados, podemos começar a analisá-los para ver o que precisa ser feito futuramente no processo de Pré-processamento:

Figura 13 - Informações dos dados

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
```

Como observado, possui alguns dados faltantes, como o “age”, “cabin” e “embarked” que precisaremos modificar de algumas forma para que o algoritmo crie a árvore corretamente, se iremos completar ou removê-lo depende do contexto, no caso do “embarked” não seria um grande problema

removê-lo por causa que possui muitos poucos dados faltantes e eles não fará tanta falta para o uso desses dados.

Tendo uma noção melhor dos dados podemos usar ferramentas que possam nos ajudar, tais como as bibliotecas sklearn.tree, para a montagem e criação da árvore, o matplotlib, para criação de gráficos e outras formas de analisar os dados pré e pós processada, e o pandas, para processamento de dados em geral.

Com essas ferramentas e análise inicial podemos já começar a mexer em nosso código com o objetivo de aumentar o máximo a precisão final da árvore de decisões

2.2 Pré-processamento

2.2.1 Valores faltantes

Primeiramente foram tratados os atributos com valores faltantes, que no caso foram o 'Embarked' e 'Age'. Nestes casos analisei os dados e verifiquei se a remoção desses dados fariam ou não falta para o Dataset original

-Embarked: Decidido remover os dados faltantes, por serem poucos dados, apenas 2 dados faltantes de 891

Figura 14 - Remoção de 'Embarked' vazios

```
for dataset in alldata:
    dataset.dropna(subset=['Embarked'], inplace=True)     #removendo
    dataset.reset_index(drop=True, inplace=True)
```

-Age: Decidido a completar os dados faltantes, por serem muitos dados, 177 dados faltantes de 89. Para completar os dados faltantes usou-se da média das idades para completá-las (mean)

Figura 15 - Inserção de dados para 'Age' vazios

```
data_train['Age'] = data_train['Age'].interpolate(method='linear', limit_direction='forward', axis=0)
data_train['Age'] = data_train['Age'].astype(int)

data_test['Age'] = data_test['Age'].interpolate(method='linear', limit_direction='forward', axis=0)
data_test['Age'] = data_test['Age'].astype(int)
```

2.2.2 Alteração de tipo de dados

Como a Árvore de Decisões recebe apenas em valores numéricos e possuímos outros tipos de valores, precisamos manipulá-los para termos pelo menos alguma representação para numéricos, na quais são: ‘Sex’, ‘Embarked’ e ‘Ticket’.

Os atributos Sex e Embarked mapeou-se para que cada palavra fosse um número respectivo:

Figura 16 - Atribuição de palavra para valor numérico

```
genders = {'male': 0, 'female': 1}

for dataset in alldata:
    dataset['Sex'] = dataset['Sex'].map(genders)

ports = {"S": 0, "C": 1, "Q": 2}

for dataset in alldata:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

Já o atributo Ticket foi preciso fazer uma operação para remover os caracteres diferentes que possuem neles, fazendo sobrar apenas valores numéricos.

Neste caso, irá gerar muitos valores distintos, mas ele conseguirá conseguir calcular os valores médios para aumentar a sua acurácia, vista que testes feitos antes da inserção dos dados Tickets tiveram valores menores do que depois dessa inserção:

Figura 17 - Modificando caracter para valor numérico

```
for dataset in alldata:
    dataset['Ticket'] = dataset.Ticket.str.split().apply(lambda x : 0 if x[:][-1] == 'LINE' else x[:][-1])
    dataset['Ticket'] = dataset['Ticket'].astype(int)
```

2.2.3 Remoção de atributos

Já os restantes dos atributos foram removidos, por não terem um impacto significativo na geração dos resultados e para não criar muitas possibilidades de ramificação. Os atributos removidos foram: 'PassengerId', 'Name', 'Cabin' e ‘Fare’.

Figura 18 - Removendo colunas não utilizadas

```
data_train = data_train.drop(['PassengerId','Name','Cabin','Fare'], axis=1)
data_test = data_test.drop(['PassengerId','Name','Cabin','Fare'], axis=1)
```

2.3 Extração de Padrões

Após modificarmos os tipos de dados, removermos e completá-los, teremos uma tabela desta forma com todos os dados em int e com os atributos restantes.

Figura 19 - Tabela de atributos com alguns dados

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Embarked
0	0	3	0	22	1	0	21171	0
1	1	1	1	38	1	0	17599	1
2	1	3	1	26	0	0	3101282	0
3	1	1	1	35	1	0	113803	0
4	0	3	0	35	0	0	373450	0

Além de que podemos analisar melhor os dados que possuímos para tentar algum padrão através de gráficos e os testes usados anteriormente.

Para gráficos temos os respectivos valores mais interessantes, por terem poucas dissipação de dados sendo mais fácil do algoritmo classificar, que são os atributos “PClass”, “sex” e “age”

Figura 20 - Gráfico de barras de PClass

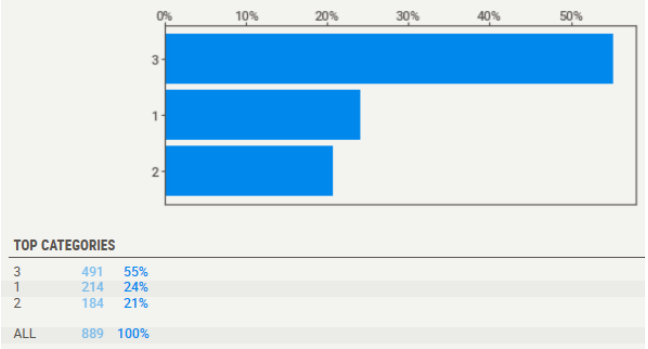


Figura 21 - Gráfico de barras de Sex

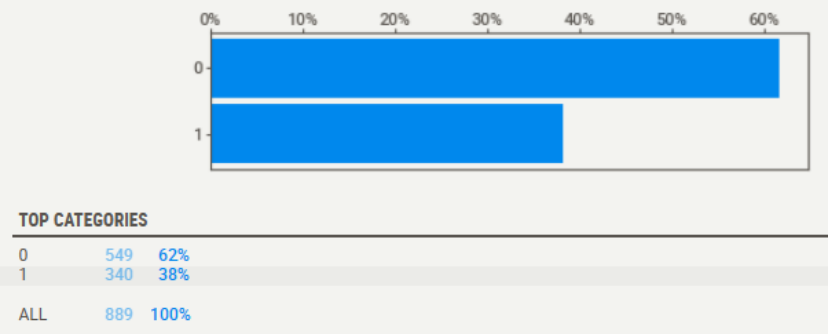
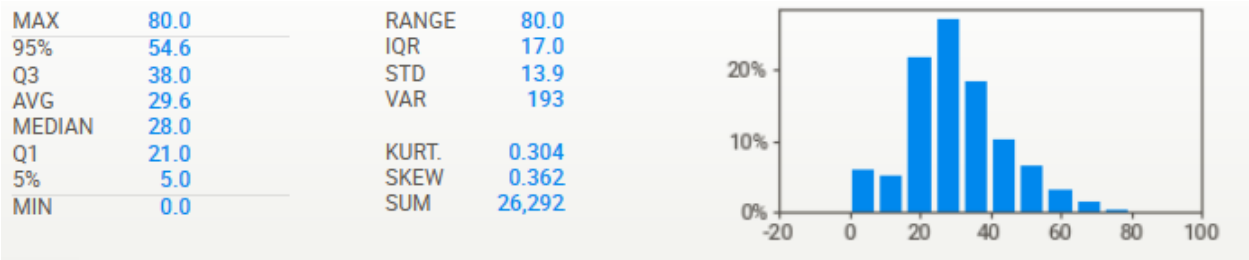


Figura 22 - Gráfico de barras de Sex



E para os testes feitos anteriormente, podemos retirar o que descobrimos de lá e testarmos se possui altos resultados com eles também nesses novos dados pré-processados, então usaremos atributos parecidos com: “criterion='entropy'”, “splitter='random'”, “min_depth=4”, e “min_samples_split=10”.

2.4 Pós-Processamento

O código usado para gerar a árvore final foi feito baseado nos testes com o melhor resultado anteriormente, mas com algumas modificações necessárias.

- **balanced_accuracy_score** = 0.8651315789473684
- **Final Code:**

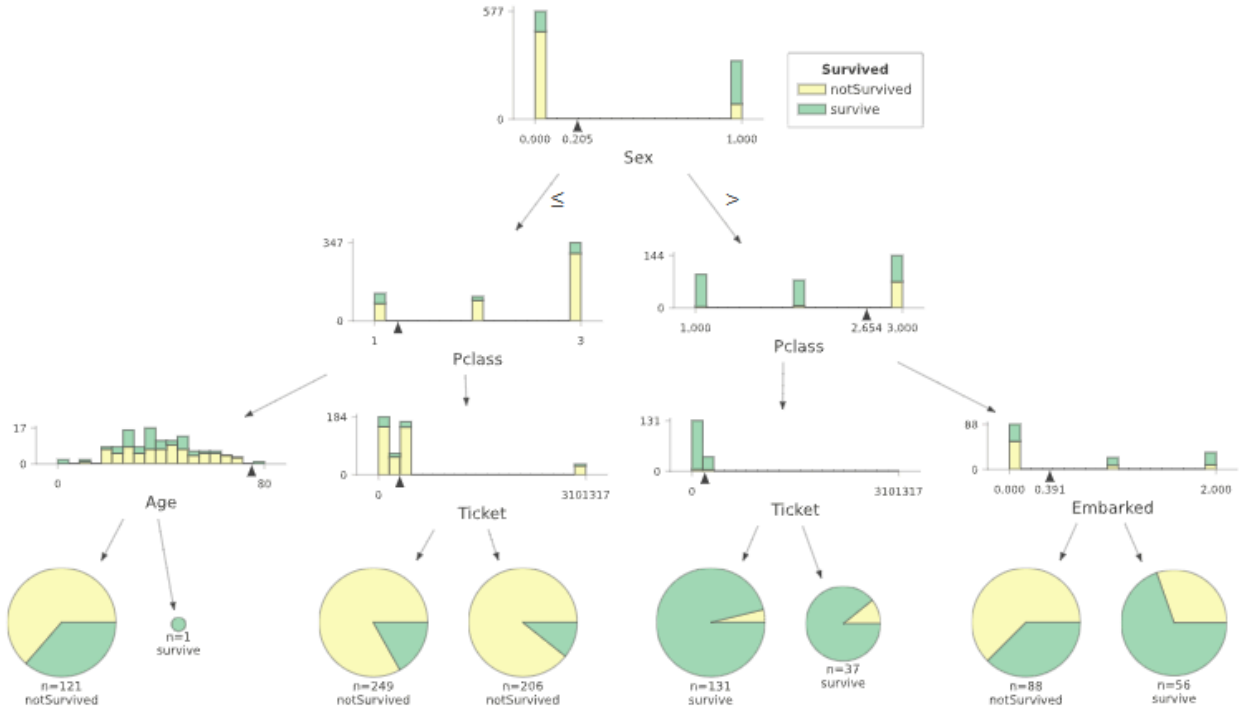
  DecisionTreeClassifier(criterion='entropy', splitter='random', random_state=0, max_depth = 3)

Com todos os testes feitos, alterando a estrutura do algoritmo e a criação da árvore, esse foi o melhor resultado encontrado, não muito diferente ao “max_depth=4” mas é um aprimoramento.

E por fim, a análise dos dados retirados por essa árvore montada e a forma de montagem dela.

Começamos procurando entender qual foi os principais atributos utilizados para a ramificação da árvore e temos(Figura 22).

Figura 23 - Gráfico de importância de atributos prévia



Algo interessante a se notar é que ele dá um valor maior aos atributo “Sex” e “PClass” algo que notamos anteriormente no processo de Extração de Dados. Como imaginado ele começa com valores com poucas classificações de dados, como por exemplo o gênero da pessoa que pode haver apenas homem ou mulher. Outra coisa interessante a se notar é que no final da árvore ele usa o atributo “Ticket” sendo que ele possui diversos valores de ticket, mas ele consegue achar uma forma de média para os valores que têm mais chances de sobreviverem, gerando uma ótima forma de resultados.

E nos outros casos testados é gerados árvores com ramificações com atributos não muito fáceis de separação por terem grande informação nos dados como “SibSp” que acaba gerando apenas treinos muitos específicos e que em testes reais não sabem onde categorizar e acabam com um resultado péssimo. O mesmo acontece se aumentarmos a altura máxima da árvore, fazendo com que gere árvore muito grandes e específicas

3 Conclusão

Por fim, conseguimos explorar os testes feitos das árvores de decisão, seu impacto na precisão do modelo mudando a forma que ela pode ser criada e utilizar esses conhecimentos para criar o nosso próprio algoritmo melhor.

Foi feito o pré-processamento dos dados originais do Titanic, incluindo tratamento de valores faltantes e transformação de atributos categóricos e numéricos, para garantir a qualidade dos dados de entrada sem erro de criar árvores erradas e menos imprecisas. Assim, otimizando os parâmetros das árvores de decisão e realizando um pré-processamento adequado dos dados, conseguimos analisar esses dados de pré-processamento para assim pensar em como podemos gerar a melhor árvore, então desenvolvemos um modelo mais preciso e robusto para tomada de decisões informadas em diversas aplicações.

Que teve como objetivo principal diminuir a profundidade máxima da árvore, para gerar árvores mais eficientes e gerais e achar os melhores atributos para a criação da árvore. Enfim, analisamos os resultados e achamos padronização neles através de gráficos e a árvore montada.

Links:
Código via:
https://github.com/RennamFariaUNIFESP/Titanic_Decision_Tree.git
ou
🗀 Código

DataSet utilizada:
https://www.kaggle.com/c/titanic/data