escola britânica de artes criativas & tecnologia

Módulo | Análise de Dados: Aprendizado de Máquina, Classificação

Caderno de **Aula**
Professor André Perez

Tópicos

1. Classificação;
2. Dados;
3. Treino;
4. Avaliação;
5. Predição;

Aulas

0. Abordagens estatísticas

- **Descritiva**: foco no passado para entender o **presente**.
- **Preditiva**: foca no passado para inferir o **futuro**.

1. Classificação

1.1. Motivação

> Dado a **largura** e o **comprimento** das pétalas e sépalas de uma flor do gênero *iris*, qual é a sua **espécie**: *versicolor*, *setosa* ou *virginica*?

Iris Versicolor **Iris Setosa** **Iris Virginica**

Queremos um conjunto de regras que representem essa relação. Uma possível solução seria o conjunto de condições `if-else` como no exemplo abaixo:

```
def f(petal_length: float, petal_width: float, sepal_length: float,
sepal_width: float) -> str:
  if sepal_width > 5.0:
    if petal_width > 2.0:
      return 'versicolor'
    else:
      return 'virginica'
  else:
    return 'setosa'
```

Este conjunto de regras pode ser representado gráficamente por uma **árvore de decisão**, onde as **folhas** representam as classes do atributo categórico ou variável resposta a ser predito e os **nós** as regras de decisão.

> Qual o melhor conjunto de regras (atributos e valores de corte) para esse conjunto de dados?

1.2. Árvore de decisão

A árvore de decisão é uma abordagem estatística que busca encontrar a relação entre um atributo categórico alvo y (variável resposta) e um conjunto de atributos preditores x_i através de um conjunto de regras simples que, quando combinadas, formam uma complexa classificação. De maneira geral, utiliza **métodos exaustivos** (força bruta) para definir a quantidade de **nós** necessário para classificar as clases do atributo alvo.

O racional por trás da construção de um nó vem do uso do conceito de **impureza** do nó.

> para cada atributo x_i:
>
> > para cada* valor w_i entre $min(x_i)$ e $max(x_i)$:
> >
> > > calcule a impureza do nó
>
> selecione x_i e w_i com a menor impureza
>
> crie um nó com x_i e w_i
>
> repita

As métricas de impureza mais utilizadas são Gini e Entropia. Uma das grandes vantagens

das árvores de decisão é a sua capacidade de **explicação** de relação entre a variável resposta e os atributos preditores, uma vez que é possível **visualiza-la**. Outra vantagem é que a técnica dispensa o tratamento dos atributos preditores (normalização, padronização, *one-hot encoding*, etc.) pois estes não são comparados entre si.

1.3. Pacote Scikit-Learn

Pacote Python para ciência de dados e *machine learning*. A documentação pode ser encontrada neste link. Possuí diversos modelos para aprendizado supervisionado, não supervisionado, etc. além de métodos auxiliares. Para a árvore de decisão, temos:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
model = DecisionTreeClassifier()
```

2. Dados

2.1. Pré-processamento

Neste módulo, vamos utilizar dados sobre as características físicas das plantas do gênero **iris**. O conjunto de dados é um dos mais famosos no aprendizado de máquina e pode ser carregado diretamente do pacote Python Seaborn.

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

In [ ]:
```
iris = sns.load_dataset('iris')
```

Vamos conhecer um pouco melhor o conjunto de dados.

In [ ]:
```
iris.head()
```

In [ ]:
```
iris.info()
```

Como o objetivo é classificar plantas nas suas espécies, vamos explorar as características de cada uma:

In [ ]:
```
iris[["species"]].drop_duplicates()
```

- **Versicolor**

In [ ]:
```
iris.query("species == 'versicolor'").describe().T
```

- **Setosa**

In [ ]:
```
iris.query("species == 'setosa'").describe().T
```

- **Virginica**

In [ ]:
```
iris.query("species == 'virginica'").describe().T
```

Nota-se que o comprimento e a largura das pétalas e sépalas diferem nas espécies. Vamos visualizar estas diferenças:

In [ ]:
```
with sns.axes_style('whitegrid'):

  grafico = sns.scatterplot(
      data=iris,
      x="petal_length",
      y="sepal_length",
      hue="species",
      palette="pastel"
  )
  grafico.set(
      title='Comprimento da pétala por comprimento da sépala',
      xlabel='Pétala (cm)',
      ylabel='Sépala (cm)'
  );
  grafico.get_legend().set_title("Espécie");
```

In [ ]:
```
with sns.axes_style('whitegrid'):

  grafico = sns.scatterplot(
      data=iris,
      x="petal_width",
      y="sepal_width",
      hue="species",
      palette="pastel"
  )
  grafico.set(
      title='Largura da pétala por largura da sépala',
      xlabel='Pétala (cm)',
      ylabel='Sépala (cm)'
  );
  grafico.get_legend().set_title("Espécie");
```

Já temos evidências suficientes para modelar a espécie como uma função do comprimento e largura de suas sépalas e pétalas:

In [ ]:
```
data = iris[
    [
        "species",
        "sepal_length",
        "sepal_width",
        "petal_length",
        "petal_width"
    ]
]
```

In [ ]:
```
data.head()
```

O resultado do pré-processamento nos trás um dado limpo e pronto para ser utilizado no treino do modelo.

> Lembre-se que o modelo de classificação com a árvore de decisão não compara atributos entre si, logo, não é necessário fazer a **escala** (numéricos) ou **codificação** (categóricos) dos valores dos atributos.

In [ ]:
```
data.query("species == 'versicolor'").head()
```

In [ ]:
```
data.query("species == 'setosa'").head()
```

In [ ]:
```
data.query("species == 'virginica'").head()
```

2.2. Treino / Teste

De maneira geral, um modelo de aprendizagem supervisionada precisa ser treinado com um conjunto de dados e avaliado com outro, assim conseguimos enter um pouco melhor a capacidade do modelo em **generalizar** as predições com dados não visto, que é a situação real em que será utilizado. Para tanto, dividimos nossa base de dados em duas: uma maior de **treino** e uma menor de **testes**.

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
predictors_train,
predictors_test,
target_train,
target_test = train_test_split(
    data.drop(['species'], axis=1),
    data['species'],
    test_size=0.25,
    random_state=123
)
```

- **Variáveis preditoras (predictors)**

In [ ]:
```
predictors_train.head()
```

In [ ]:
```
predictors_train.shape
```

In [ ]:
```
predictors_test.head()
```

In [ ]:
```
predictors_test.shape
```

- **Variável resposta (target)**

In [ ]:
```
target_train.head()
```

In [ ]:
```
target_train.shape
```

In [ ]:
```
target_test.head()
```

In [ ]:
```
target_test.shape
```

3. Treino

O treino de modelos de aprendizagem supervisionada consiste na etapa de calculo dos parametros do modelo baseado na associação da variável resposta com os variáveis preditoras através do uso de um ou mais algoritmos. No caso da árvore de decisão, estamos interessados em encontrar um conjunto de regras simples (**nós**) que vão dividir o conjunto de dados em categorias (**folhas**) através do uso de uma métrica de **impureza**.

3.1. Algoritmo

O treino de um modelo de árvore de decisão (do pacote Python Scikit Learn) é feito através do uso do algoritmo **CART** (explicação neste link). A explicação completa do algoritmo foge do escopo desse curso mas o raciocínio por trás é semelhante ao apresentado na aula 1 deste módulo: um processo iterativo que constrói **nós** (`if-else`) através da escolha de um atributo e um valor de corte utilizando uma métrica de **impureza**.

In [ ]:
```
model = model.fit(predictors_train, target_train)
```

In [ ]:
```
model.__dict__
```

Podemos visualizar a árvore de decisão gerada com o auxílio do pacote Python Graphviz.

In [ ]:
```
import graphviz
from sklearn import tree

tree_data = tree.export_graphviz(model, out_file=None)
graph = graphviz.Source(tree_data)
graph
```

Com o modelo treinado, estamos prontos para fazer predições.

In [ ]:
```
data.head(1)
```

In [ ]:
```
features = np.array([-0.897674, 1.015602, -1.335752, -1.311052])
prediction = model.predict(features.reshape(1, -1))

print(prediction)
```

4. Avaliação

Para enter o poder preditivo do modelo de aprendizagem supervisionada, precisamos avaliar sua capacidade de generalização, ou seja, avalivar as predições em dados "não vistos" na etapa de treino. Comparamos então as predições com os dados reais através de uma métrica.

- **Posição predita**

In [ ]:
```
target_predicted = model.predict(predictors_test)
```

In [ ]:
```
target_predicted[0:5]
```

In [ ]:
```
target_predicted.shape
```

- **Posição teste**

In [ ]:
```
target_test[0:5]
```

In [ ]:
```
target_test.shape
```

4.1. Matriz de confusão

A matriz de confusão é uma técnica que resume as predições feitas por um modelo supervisionado de classificação em uma matriz, facilitando a comparação com as classes reais. A soma dos elementos da diagonal princial da matriz apresenta a quantidade de classes corretamente classificadas. Da matriz de confusão derivam-se diversas outras métricas, como a **acurácia**.

In [ ]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

In [ ]:
```
confusion_matrix = confusion_matrix(target_test, target_predicted)
print(confusion_matrix)
```

O pacote Scikit-Learn possuí um método que facilita a visualização da matriz de confusão em um gráfico de **mapa de calor**.

In [ ]:
```
from sklearn.metrics import plot_confusion_matrix
```

In [ ]:
```
plot_confusion_matrix(model, predictors_test, target_test)
plt.show()
```

4.2. Acurácia

A **acurácia** é uma métrica que combina a exatidão (o quão certo) e a precisão (o quanto desvia) das predições de um modelo supervisionado de classificação. É definida como a soma das classificações corretas (real igual ao predito) dividida pelo total de classificações realizadas.

> $ACC(y,\hat{y}) = {\frac{\sum_{i=1}^{n} 1}{n}, \forall y_i=\hat{y}_i}$

In [ ]:
```
total = confusion_matrix.sum()
```

```
print(total)
```

In [ ]:
```
acertos = np.diag(confusion_matrix).sum()
print(acertos)
```

In [ ]:
```
acuracia = acertos / total
print(acuracia)
```

In [ ]:
```
print(f"{round(100 * acuracia, 2)}%")
```

O pacote Scikit-Learn possuí uma série de métodos que facilitam o cálculo de métricas de avaliação, como a **acurácia**. Uma lista completa de métricas implementadas pelo pacote por ser encontrada neste link.

In [ ]:
```
from sklearn.metrics import accuracy_score
```

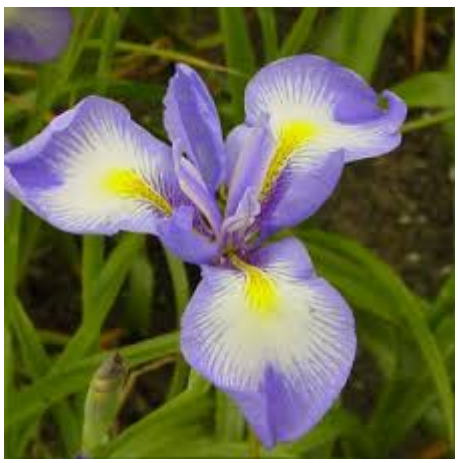
In [ ]:
```
acuracia = accuracy_score(target_test, target_predicted)
print(acuracia)
```

In [ ]:
```
print(f"{round(100 * acuracia, 2)}%")
```

5. Predição

Com o modelo treinado, avaliado e selecionado, podemos utiliza-lo para resolver os problemas reais que motivaram sua construção.

- **Exemplo**: Flor do gênero **íris** com sépala de 6.39cm e 2.71cm e pétala de 6.03cm e 2.23cm, sendo a primeira medida o comprimento e a segunda a largura, respectivamente.



In [ ]:
```
flor = np.array([6.39, 2.71, 6.03, 2.23])
```

In [ ]:
```
especie = model.predict(flor.reshape(1, -1))
print(especie)
```

Lembre-se que a classe predita nada mais é do que a aplicação de uma função ou conjunto de regras, com parametros "aprendidos" durante o treino do modelo, aos dados de entrada. Regras estas que, no caso da classificação por um modelo de **árvore de decisão**, podemos acompanhar visualmente:

In [ ]:
```
model.classes_
```

In [ ]:
```
graph
```

Conclui-se então que a espécie da flor é **íris virginica**.