



fonte da imagem: <https://fortisedu.info/tips-for-learning-python-for-people-without-coding-background/>

Imersão de Dados - Alura: Análise de Dados ENEM 2019 (Aula 05)

Published on October 23, 2020

Alvaro Carnielo e Silva

Applied Aerodynamics and CFD | MRB | Mechanical Engineer | Data Science

6 articles

✓ Following

Olá a todos,

Hoje teremos o material da última aula da semana de imersão de dados promovida pela ****ALURA Cursos Online****.


Na aula de ontem foram apresentados alguns conceitos básicos de **machine learning** e aplicamos alguns métodos de aprendizado e predição baseados nos dados do ENEM 2019.








Abaixo podemos ver um compilado do que foi feito na última aula:



Messaging












Reactivate Premium for Free

```

5 import seaborn as sns
6
7 from sklearn.model_selection import train_test_split
8 from sklearn.svm import LinearSVR
9 from sklearn.metrics import mean_squared_error as mse
10 from sklearn.metrics import mean_absolute_error as mae
11
12 #-----#
13 ### Importar dados do ENEM
14 fonte = 'https://github.com/alura-cursos/imersao-dados-2-2020/blob/master/MICRODADOS_ENEM_2019_SAMPLE_43278.csv?raw=true'
15 dados = pd.read_csv(fonte) # importa os dados compartilhados (modo CSV - comma separated values) e disponibiliza no
16
17 #-----#
18 ### Criar variável
19 # Listar colunas desejadas com as notas das provas
20 notas = ['NU_NOTA_CN', 'NU_NOTA_CH', 'NU_NOTA_MT', 'NU_NOTA_LC', 'NU_NOTA_REDACAO']
21
22 # Somar as notas das provas e formar uma coluna nova
23 dados['SOMA_NOTAS'] = dados[notas].sum(axis=1)
24
25 #-----#
26 ### Limpar base de dados
27 # Eliminar NaN da base de dados
28 dados_limpos = dados[notas].dropna()
29
30 # Lita de colunas de input
31 entrada = ['NU_NOTA_CN', 'NU_NOTA_CH', 'NU_NOTA_LC', 'NU_NOTA_REDACAO']
32 # Lista de colunas de outputs
33 saida = ['NU_NOTA_MT']

```

```

1 # DFs que serão utilizados (dados sem notas NaN)
2 notas_entrada = X = dados_limpos[entrada]
3 notas_saida = y = dados_limpos[saida]
4
5 #-----#
6 ### Iniciar o processo de ML
7 # Definir um SEED para evitar variabilidade do processo
8 seed=4321
9
10 # Dividir os dados em modelo de treinamento e de teste
11 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=seed)
12
13 # Selecionar o modelo e treinar
14 modelo_LinearSVR = LinearSVR(random_state=seed)
15
16 # Treinar modelo
17 modelo_LinearSVR.fit(X_train, y_train)
18 y_prev_LinearSVR = modelo_LinearSVR.predict(X_test)

```

Quando criamos o **seed** o objetivo era diminuir a aleatoriedade do modelo e sua dependência da amostra utilizada. Contudo, sempre que fizermos um modelo e não utilizarmos o **seed** teremos um erro diferente de erro, como podemos ver abaixo:

```

1 # Selecionar o modelo e treinar
2 modelo_LinearSVR = LinearSVR()
3
4 # Treinar modelo
5 modelo_LinearSVR.fit(X_train, y_train)
6 y_prev_LinearSVR = modelo_LinearSVR.predict(X_test)
7 print(mse(y_test, y_prev_LinearSVR))
8 print(mae(y_test, y_prev_LinearSVR))

```

9286.027901893585
76.02858972672135
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: L
"the number of iterations.", ConvergenceWarning)

```

1 # Selecionar o modelo e treinar
2 modelo_LinearSVR = LinearSVR()
3
4 # Treinar modelo
5 modelo_LinearSVR.fit(X_train, y_train)
6 y_prev_LinearSVR = modelo_LinearSVR.predict(X_test)
7 print(mse(y_test, y_prev_LinearSVR))
8 print(mae(y_test, y_prev_LinearSVR))

```

7532.952847806192
70.48375322027879

Uma das formas de diminuir a dependência que o modelo possui pela **seed** aplicada é o uso de um outro modelo: a árvore de decisões (**DecisionTreeRegressor**)



Messaging





Search

Reactivate
Premium for Free

Como podemos ver, o **DecisionTreeRegressor** diminuiu significativamente a variância da simulação, porém continuamos com a possibilidade de variação devido às amostras, como podemos ver abaixo.

```
1 ### Iniciar o processo de ML
2 # Dividir os dados em modelo de treinamento e de teste
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
4
5 # Importar modelo
6 from sklearn.tree import DecisionTreeRegressor
7 # escolher modelo
8 modelo_DecisionTreeRegressor = DecisionTreeRegressor(max_depth = 3)
9 # Treinar modelo
10 modelo_DecisionTreeRegressor.fit(X_train, y_train)
11 # Prever resultado
12 y_DecisionTreeRegressor = modelo_DecisionTreeRegressor.predict(X_test)
13 print(mse(y_test, y_DecisionTreeRegressor))
14 print(mae(y_test, y_DecisionTreeRegressor))
```

```
6089.60713533443
62.4519508362679
```

```
1 ### Iniciar o processo de ML
2 # Dividir os dados em modelo de treinamento e de teste
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
4
5 # Importar modelo
6 from sklearn.tree import DecisionTreeRegressor
7 # escolher modelo
8 modelo_DecisionTreeRegressor = DecisionTreeRegressor(max_depth = 3)
9 # Treinar modelo
10 modelo_DecisionTreeRegressor.fit(X_train, y_train)
11 # Prever resultado
12 y_DecisionTreeRegressor = modelo_DecisionTreeRegressor.predict(X_test)
13 print(mse(y_test, y_DecisionTreeRegressor))
14 print(mae(y_test, y_DecisionTreeRegressor))
```

```
6021.798720257971
62.077587448520106
```

Uma forma de diminuir a influência da quebra dos dados na previsão do modelo é através da validação cruzada (**cross_validate**).

A validação funciona assim: os dados são quebrados em vários subgrupos, assim, são feitos testes de número igual à quantidade de quebra dos dados em seguida é calculada uma média da qualidade dos modelos

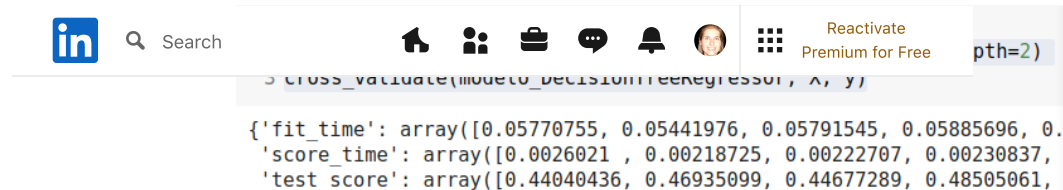
Para entender melhor o processo de validação cruzada, recomenda-se dar uma olhada no link a seguir:

Referência para cross validation:
https://scikit-learn.org/stable/modules/cross_validation.html



Messaging





```

3 cross_validate(modelo_DecisionTreeRegressor, X, y)

{'fit_time': array([0.05770755, 0.05441976, 0.05791545, 0.05885696, 0.05885696]),
'score_time': array([0.0026021, 0.00218725, 0.00222707, 0.00230837, 0.00230837]),
'test_score': array([0.44040436, 0.46935099, 0.44677289, 0.48505061, 0.48505061])}

```

Como é possível ver, o **cross_validate** retorna um **dictionary** contendo 3 parâmetros:

```

fit_time: tempo treino e teste dos dados para cada divisão
score_time: tempo para calcular o score de cada divisão
test_score: a medida de qualidade do modelo para cada divisão

```

É possível ver que o **test_score** não é nada parecido com os valores que estávamos acostumados até agora. Isso se deve à métrica utilizada no **cross_validate** que é o **r2**. Para que tenhamos a mesma métrica que usamos nos modelos anteriores, precisamos mudar o parâmetro **scoring** para **mean_square_error** ou **mean_absolute_error**.

```

Referência métricas
https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

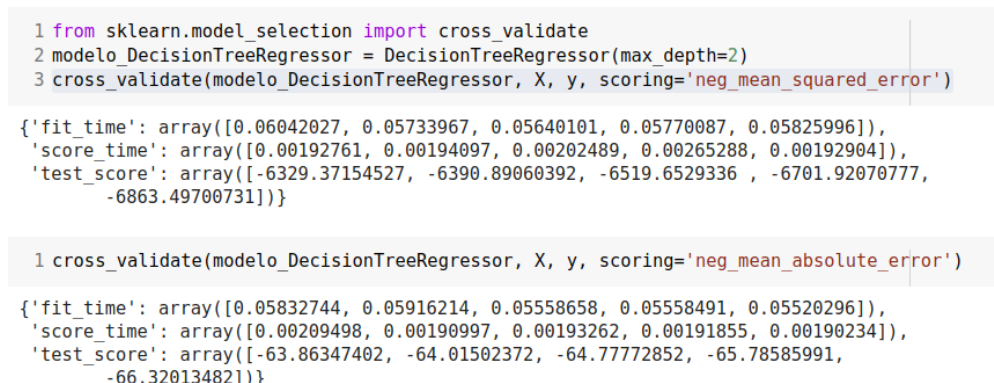
Referência métrica r2
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score

```

Para selecionar o *mean square error* ou o *mean absolute error*, de acordo com a documentação, precisamos selecionar respectivamente **'neg_mean_squared_error'** e **'neg_mean_absolute_error'** (entre aspas mesmo, pois o *input* desse parâmetro é uma **str**).

Esses valores de erro são negativos por conta de uma lógica de otimização que existe dentro do **scikit-learning** de quanto maior o número, melhor ele é (isso é aplicado no **r2** que pode ser a melhor das hipóteses 1).

Assim, dado que para *mean square error* e *mean absolute error* o melhor resultado possível é erro=0, a lógica do código fica preservada.



```

1 from sklearn.model_selection import cross_validate
2 modelo_DecisionTreeRegressor = DecisionTreeRegressor(max_depth=2)
3 cross_validate(modelo_DecisionTreeRegressor, X, y, scoring='neg_mean_squared_error')

{'fit_time': array([0.06042027, 0.05733967, 0.05640101, 0.05770087, 0.05825996]),
'score_time': array([0.00192761, 0.00194097, 0.00202489, 0.00265288, 0.00192904]),
'test_score': array([-6329.37154527, -6390.89060392, -6519.6529336, -6701.92070777, -6863.49700731])}

1 cross_validate(modelo_DecisionTreeRegressor, X, y, scoring='neg_mean_absolute_error')

{'fit_time': array([0.05832744, 0.05916214, 0.05558658, 0.05558491, 0.05520296]),
'score_time': array([0.00209498, 0.00190997, 0.00193262, 0.00191855, 0.00190234]),
'test_score': array([-63.86347402, -64.01502372, -64.77772852, -65.78585991, -66.32013482])}

```



Search

Reactivate
Premium for Freeão. Se qu
divisões

desejamos.

```

1 cross_validate(modelo_DecisionTreeRegressor, X, y, cv=10,
2                 scoring='neg_mean_squared_error')

{'fit_time': array([0.06551909, 0.06058764, 0.06083393, 0.06149578, 0.06
0.06012726, 0.06659055, 0.06572199, 0.06294823, 0.06389213]),
'score_time': array([0.00162935, 0.00160742, 0.00154734, 0.00165534, 0.
0.00159144, 0.00157905, 0.0015974 , 0.00157905, 0.00158906]),
'test_score': array([-6139.92068201, -6537.82641962, -6337.73724622, -6
-6233.80062047, -6795.32790094, -6713.2975924 , -6692.69619117,
-6633.42595486, -7085.89236185])}

```

Assim, caso queiramos obter o valor de erro médio da validação cruzada, temos que multiplicar por -1 e fazer a média dos **test_score**.

```

1 media_cros_valid_mse = (cross_validate(modelo_DecisionTreeRegressor, X, y, cv=10,
2                                     scoring='neg_mean_squared_error')['test_score']
3 media_cros_valid_mae = (cross_validate(modelo_DecisionTreeRegressor, X, y, cv=10,

```

Like Comment Share

6562.798881826287

```
1 media_cros_valid_mae
```

64.96844491395005

Como já vimos anteriormente, a média não significa muita coisa sozinha no estudo de estatística. Para ter uma melhor compreensão sobre a qualidade do modelo precisamos ter informações além da média, como desvio padrão:

```

1 desv_pad_cros_valid_mse = (cross_validate(modelo_DecisionTreeRegressor, X, y,
2                                     cv=10, scoring='neg_mean_squared_error'
3                                     )['test_score']* -1).std()
4 desv_pad_cros_valid_mae = (cross_validate(modelo_DecisionTreeRegressor, X, y,
5                                     cv=10, scoring='neg_mean_absolute_error'
6                                     )['test_score']* -1).std()

```

```
1 desv_pad_cros_valid_mse
```

268.61115177699946

```
1 desv_pad_cros_valid_mae
```

1.2962062806277577

Das teorias de estatística sabemos que 95% dos dados que temos estão entre 2 desvio padrão para baixo (limite inferior) e 2 desvio padrão para cima (limite superior). Podemos assim calcular esse desvio padrão:



No alt text provided for this image

Para facilitar a aplicação no futuro, vamos montar uma função que retorne esses dados que acabamos de calcular:



No alt text provided for this image



Messaging





Search

Reactivate
Premium for Free

Qualidade no

ar o apre

Assim, uma boa prática para evitar isso é misturar os dados. Para isso, o **scikit** tem uma função que facilita bastante o trabalho, **KFold**.



No alt text provided for this image



No alt text provided for this image

É possível ver agora que os modelo não varia tanto mesmo com a mudança do embaralhamento dos dados.

Caso queiramos ter uma reprodutividade do processo mesmo com o embaralhamento, podemos configurar um **SEED** para isso:



No alt text provided for this image

Observe que o resultado se repete mesmo com um novo embaralhamento dos dados.

Continuando nosso estudo. Vamos agora falar do modelo de árvore de decisão em si.

Os parâmetros que são setados antes de iniciar a modelagem são chamados de hiper-parâmetros, pois eles influenciam diretamente nos resultados do nosso modelo.

Um hiper-parâmetro importante é o **max_depth**. Esse parâmetro define o quão "profundo" nossa árvore de decisão deve descer para fazer a previsão. Mais detalhes desse parâmetro podem ser vistos na documentação do **DecisionTreeRegressor**.

Referência decision tree:

<https://scikit-learn.org/stable/modules/tree.html>

https://en.wikipedia.org/wiki/Decision_tree#:~:text=A%20decision%20tree%20is%20a%20

Podemos fazer uma exploração de como o **max_depth** influencia os resultados do modelo. Para isso faremos um *loop* de for repetindo o procedimento anterior, porém dessa vez guardaremos os dados em um **DataFrame**.

Além disso, habilitaremos uma configuração do processo de **cross_validation** que nos permite ver o resultado da qualidade dos testes de treino (**return_train_score=True**).



No alt text provided for this image



No alt text provided for this image

Para enxergar o resultado de forma mais simples, podemos plotar um gráfico com a variação do score de previsão e teste:



No alt text provided for this image



Messaging





Search

Reactivate
Premium for Free

Como é possível ver, aumentar o número de camadas da árvore de decisão sem melhorar o resultado do modelo de treino. O mesmo não ocorre com os resultados de teste. É possível notar uma melhora até 7 camadas da árvore. Depois disso, o modelo passa a ter resultados piores.

Isso ocorre, pois, o modelo começa a "decorar os resultados de treino e ficar muito bom em prever esses dados, porém quando tenta prever os resultados de teste, ele passa a ter resultados muito ruins, pois ele está viciado nos dados de treino. Esse problema é conhecido como "overfitting".

Esse processo pode ser repetido também para outros parâmetros da árvore de decisão (como o `max_leaf_nodes`) até que se tenha a melhor configuração possível para uma dada decisão.



No alt text provided for this image



No alt text provided for this image



No alt text provided for this image



No alt text provided for this image

Nesse último gráfico é fácil notar que conforme aumentamos o número máximo de "folhas" por nó, diminuímos o erro do modelo. Contudo, da mesma forma que vimos para o estudo de variação de camadas, o erro aumenta após um certo valor de "folhas" por nó.

Dessa forma temos que buscar a combinação de parâmetros da árvore de decisão que apresente melhores resultados para o estudo.

Bom, isso é tudo por hoje.

Essa aula foi a última da série de #ImersãoDados promovida pela Alura. Espero que eu tenha conseguido transmitir o conhecimento de forma clara para todos.

Até a próxima!

[Report this](#)

Published by

Alvaro Carnielo e SilvaApplied Aerodynamics and CFD | MRB | Mechanical Engineer | Data Science
Published • 2d

6 articles

✓ Following

Olá a todos,

Por conta da semana de imersão de dados promovida pela Alura Cursos Online resolvi aproveitar o conteúdo e utilizá-lo no último artigo dessa série aqui no LinkedIn.


Nesse artigo damos continuidade à exploração de dados e machine learning sobre os dados do ENEM proposta pela equipe da Alura Cursos Online.







Obrigado a toda a equipe da Alura Cursos Online, por compartilhar esse conhecimento durante a



Messaging



 Search



 [Reactivate Premium for Free](#)

Ter carro far você tirar nota no ENEM?
Alvaro Carnielo e Silva on LinkedIn

Imersão de Dados - Alura: Análise de Dados ENEM 2019 (Aula 04)
Alvaro Carnielo e Silva on LinkedIn

Imersão de Dados - Alura: Análise de Dados ENEM 2019 (Aula 03)
Alvaro Carnielo e Silva on LinkedIn

[See all 6 articles](#)