Nome: Michel Felipe de Sena Nassif

Desafio Data Scientist - Qual a quantidade de bicicletas alugadas por hora em Montreal?

Desafio:

O desafio será desenvolver um modelo para prever a quantidade de bicicletas alugadas por hora ou por dia (fica a sua escolha).

Neste desafio, utilizaremos os dados da ***BIXI Montréal***. Também, os dados sobre as condições climáticas que foram obtidos através do Governo do Canadá.

I. Preparar e Explorar:

Os dados estão disponíveis publicamente em:

- ***https://www.bixi.com/en/open-data***
- ***http://www.climate.weather.gc.ca***

Mas também, neste **link**, disponibilizamos os dados que foram coletados dessas bases públicas.

Nesta etapa, diferencie os tipos de variáveis e quais tratamentos fazer para elas. Para poder alcançar o objetivo do desafio, conecte os dados das duas fontes, e explore descritivamente a sua base criada.

II. Prever:

Nesta etapa, escolha algum modelo, implemente e avalie os resultados.

Introdução

Estudando a base de dados de todas as bicicletas alugadas na cidade canadense de Montreal, no período de 15 de março a 15 de novembro de 2015 e 2016, assim como os dados climáticos da região, escolheu-se criar um modelo que juntasse as duas bases de informações e conseguisse prever o número de bicicletas alugadas na região por hora.

Através de uma primeira análise, foi possível concluir que se trata de um problema de modelagem supervisionada, no qual a partir de uma base de dados tratada deve-se ensinar o modelo a prever resultados de uma variável que dependa de outras totalmente independentes, presentes na nossa base de dados. Além disso, como, para o problema, a variável chave é o número de bicicletas alugadas por hora, a melhor estratégia para estimar esse valor é realizar um arquétipo de regressão que represente a dependência entre as variáveis.

Figura 1 – Exemplos de estratégias adotadas para criação de modelos.

Para a criação de um modelo preliminar, escolheu-se variáveis que levassem em conta aspectos temporais da rotina da cidade, como dia da semana ou hora e climáticos como a temperatura ou a umidade relativa do ar, de modo que as entradas escolhidas foram: ano, mês, data, hora, respectivo dia da semana - variando de 0 para segunda feira e 6 para domingo -, temperatura (ºC), ponto de orvalho(ºC), umidade relativa do ar, velocidade do vento e visibilidade.

Pré-processamento dos dados

Feita essa estratégia de resolução do problema, percebeu-se que se tratam de dados que precisaram ser manipulados. Os arquivos estavam separados por meses e para uma melhor manipulação escolheu-se agrupar esses arquivos em dois– um climático e um com todas as bicicletas alugadas. Para isso foi utilizado o seguinte código do arquivo *merging.py*, obtidos em um site[1], capaz de agrupar vários arquivos .csv.

```
import pandas as pd
import os

files = [f for f in os.listdir('.') if os.path.isfile(f)]

merged = []

for f in files:
    filename, ext = os.path.splitext(f)
    if ext == '.csv':
        read = pd.read_csv(f)
        merged.append(read)

result = pd.concat(merged)

result.to_csv('merged.csv')
```

Figura 2 – Imagem do Código capaz de agrupar vários arquivos .csv.

No entanto, algumas variáveis precisavam ser transformadas e outras criadas, como por exemplo o dia da semana e o número de bicicletas alugadas por hora. Em um arquivo chamado Pre*processamento.py* realizou-se essa etapa de transformação dos dados, assim como a integração entre as duas bases dados, para no final ter-se uma base de dados pronta para ser utilizada em um modelo.

Nessa etapa, como primeiro passo, importou-se as bibliotecas e as duas bases de dados: *datasetbike* referente aos aluguéis de bicicletas e o *datasetclimate* referente aos dados climáticos; criando-se em seguida a base de dados finais, representada por *datasetfinal*. Fonte de informação que apresenta os dados da lista *legenda*, como mostrado na figura 3.

Após importada as bases de dados, transferiu-se diretamente as informações de ano, mês, dia, hora, temperatura, ponto de orvalho, umidade relativa, velocidade do vento e visibilidade do *datasetclimate* para o *datasetfinal.* Em seguida, através da biblioteca *datetime*, utilizou-se a função *weekday()*, com os parâmetros de ano, mês e dia, para se achar o dia da semana correspondente e armazenando assim no banco de dados, também mostrada na figura 3.

```
# Importando as bibliotecas
import numpy as np
import pandas as pd
from datetime import date
import collections


# Importando os dados de aluguel de bicicleta e os climaticos
datasetbike = pd.read_csv('Bixi_time.csv', encoding = 'ISO-8859-1')
datasetclimate = pd.read_csv('Climate2.csv', encoding = 'ISO-8859-1')

#Criando o dataset final
legenda=['Ano','Mês','Dia', 'Hora','Dia da Semana', 'Temperatura(ºC)',
         'Ponto de Orvalho(ºC)', 'Umidade Relativa', 'Velocidade do Vento(Km/h)',
         'Visibilidade', 'Nº de Alugueis por hora']
datasetfinal=pd.DataFrame(index= datasetclimate.index, columns=legenda)

#Importando as variaveis de tempo e clima para a base de dados
datasetfinal.iloc[:,0]=datasetclimate.iloc[:,2] #Ano
datasetfinal.iloc[:,1]=datasetclimate.iloc[:,3] #Mês
datasetfinal.iloc[:,2]=datasetclimate.iloc[:,4] #Dia
datasetfinal.iloc[:,5]=datasetclimate.iloc[:,7] #Temperatura
datasetfinal.iloc[:,6]=datasetclimate.iloc[:,9] #Ponto de Orvalho
datasetfinal.iloc[:,7]=datasetclimate.iloc[:,11] #Umidade Relativa
datasetfinal.iloc[:,8]=datasetclimate.iloc[:,15] #Velocidade do Vento
datasetfinal.iloc[:,9]=datasetclimate.iloc[:,17] #Visibilidade
```

Figura 3 – Passo 1 do pré-processamento de dados.

Por último, era necessário criar a variável de saída do número de bicicletas alugadas. Para isso criou-se uma lista auxiliar chamada *databike*, na qual todos os dados temporais de cada aluguel de bicicleta – do ano até a hora do aluguel - foram armazenados, para depois criar uma variável *counter* que junta todos os aluguéis que ocorreram na mesma hora. Vale notar, que para o problema foi escolhido levar em consideração a hora do aluguel da bicicleta e não sua devolução, uma vez que apenas 0,7% dos aluguéis usavam a bicicleta por mais de uma hora.

Tendo, o número de aluguéis que aconteceram na mesma hora, bastou-se apenas percorrer a base de dados do *datasetfinal,* já separada por dados de hora em hora, de modo que cada número de aluguéis por hora foi armazenado em sua determinada hora correspondente, criando assim a variável de saída no datasetfinal, convertendo-o depois para um arquivo .csv. Processo que pode ser inteiro visto na figura 4.

```
#Somando o nº de bicicletas alugadas em cada hora
databike = ["" for x in range(len(datasetbike))]
for i in range(len(datasetbike)):
    databike[i]=datasetbike.iloc[i,1][0:13]
counter=collections.Counter(databike)


# Preenchendo no dataset a variável do nº de bicicletas alugadas por hora
valores = list(counter.values())
datas = list(counter.keys())
for i in range(len(datasetfinal)):
    data=[datasetfinal.iloc[i,0], datasetfinal.iloc[i,1],datasetfinal.iloc[i,2],
    datasetfinal.iloc[i,3]]
    for k in range(len(datas)):
        datacomp=[int(datas[k][0:4]),int(datas[k][5:7]),int(datas[k][8:10]),
                  int(datas[k][11:13])]
        ano=datacomp[0]
        mes=datacomp[1]
        dia=datacomp[2]
        hora=datacomp[3]
        if(ano==data[0] and mes ==int(data[1]) and dia ==int(data[2]) and hora==int(data[3])):
            datasetfinal.iloc[i,10]=valores[k]

#Exportando para CSV:
datasetfinal.to_csv('DatasetFinal.csv', encoding='utf-8')
```

Figura 4 – Achando a variável de número de aluguéis de bicicleta por hora.

Criação e Aperfeiçoamento do Modelo

Com a base de dados disponível e todas as variáveis já manipuladas e criadas, passou-se para a etapa de criação do modelo, no arquivo chamado *CriaçãodoModelo.py*. Vale notar que foi necessário remover as linhas referentes aos períodos do começo de abril de 2015 e 2016 e final de novembro de 2015 e 2016, uma vez que na base de dados disponibilizada de aluguéis de bicicleta não havia dados referentes a esse período.

Nesse arquivo, após importação do banco de dados, realizou-se um passo que teve como intuito localizar casos de dados faltando, adotando-se a estratégia de substituí-los pela média, no caso da matriz de variáveis independentes X e por 0 para a variável independente Y (uma vez que um NaN no número de aluguéis de bicicletas afirma que não foi encontrada nenhuma locação de bicicleta no processo de contagem do pré-processamento de dados para aquela hora).

Realizado essa etapa, dividiu-se aleatoriamente a base de dados em uma parte referente ao treino o modelo (80% das informações) e outra para o teste (20% das informações), denominados respectivamente *training_set e test_set*. Processos que podem ser todos vistos na figura 5.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.formula.api as sm
from datetime import datetime

# Importando o Dataset
dataset = pd.read_csv('DatasetFinal.csv')
dataset['Horario'] = (dataset.apply(lambda row: datetime(
                              int(row['Ano']), int(row['Mês']), int(row['Dia']),
                              int(row['Hora'])), axis=1))

#Lidando com a possibilidade de nenhuma bicicleta ter sido alugada em um horário
dataset.iloc[:,11]=dataset.iloc[:,11].fillna(0)
np.argwhere(np.isnan(dataset.iloc[:,11]))

#Criando os vetores de entrada e saída
X = dataset.iloc[:, 1:11].values
y = dataset.iloc[:, 11].values

# Trabalhando com a possibilidade de falta de dados em X
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(X[:, 1:11])
X[:, 1:11] = imputer.transform(X[:, 1:11])


#Dividindo os dados em Training set e Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Figura 5 – Etapa anterior a criação do modelo.

Realizado isso, criou-se um modelo inicial de regressão (*regressor*) com parâmetros genéricos para o nosso modelo utilizando o algoritmo de “Random Forest Regression”[2], modelo robusto que já foi capaz de lidar muito bem com a nossa base de dados, como pode ser visto na tabela 2. Vale notar que foram testados outros processos de regressão como o SVR e o Polinomial, porém acabaram mostrando resultados inferiores pelo algoritmo escolhido para o problema.

Com o intuito de refinar o modelo, determinou-se as principais variáveis que afetam o número de aluguéis de bicicletas realizados por hora. Para isso foi aplicado manualmente os passos do algoritmo de “Backward Elimination”[3], no qual remove-se iterativamente todas as variáveis que apresentam um valor P maior que um nível de significância escolhido – no caso deste problema: 0,05. Com isso verificou-se que as variáveis do Ponto de Orvalho, dia do mês, ano não tinham muita correlação nosso vetor de saída, assim como a velocidade do vento que afeta o modelo, porém não o suficiente de acordo com os critérios estabelecidos (valor P de 0,07>0,05).

Resultando em um modelo reduzido as variáveis do mês, hora, dia da semana, temperatura, umidade relativa e visibilidade, que foram armazenadas em uma nova matriz denominada *X_opt,* utilizada novamente para a criação de um novo modelo regressão. Processos que são todos representados na figura 6.

```
# Ajustando o modelo de Random Forest Regression
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor()
regressor.fit(X_train, y_train)


#Olhando os valores de Adjusted R² e P-Value
#Busca pelas variáveis de entrada mais importantes(P-Value<0.05)
X_opt= X_train[:,[0,1,2,3,4,5,6,7,8,9]]
regressor_OLS=sm.OLS(endog=y_train, exog = X_opt).fit()
regressor_OLS.summary()

#Remoção do Ponto de Orvalho
X_opt= X_train[:,[0,1,2,3,4,5,7,8,9]]
regressor_OLS=sm.OLS(endog=y_train, exog = X_opt).fit()
regressor_OLS.summary()

#Remoção do dia
X_opt= X_train[:,[0,1,3,4,5,7,8,9]]
regressor_OLS=sm.OLS(endog=y_train, exog = X_opt).fit()
regressor_OLS.summary()

#Remoção do ano
X_opt= X_train[:,[1,3,4,5,7,8,9]]
regressor_OLS=sm.OLS(endog=y_train, exog = X_opt).fit()
regressor_OLS.summary()

#Remoção da velocidade do vento
X_opt= X_train[:,[1,3,4,5,7,9]]
regressor_OLS=sm.OLS(endog=y_train, exog = X_opt).fit()
regressor_OLS.summary()
X_test=X_test[:,[1,3,4,5,7,9]]

# Ajustando o novo modelo
regressor.fit(X_opt, y_train)
```

Figura 6 – Criação do modelo e procura pelas variáveis mais importantes.

Por último, na figura 7, observa-se o código que procurou otimizar um pouco mais o modelo, olhando-se para seus hiperparâmetros, através da técnica de “Grid Search”[4]. Foram realizados três testes, um com parâmetros padrões e dois que percorreram uma rede de valores, representados pela lista *parameters*, resultados que podem ser vistos na tabela 2. Para então recriar o modelo de regressão novamente, dessa vez com os resultados dos melhores hiperparâmetros obtidos nos testes.

```
#Remoção da velocidade do vento
X_opt= X_train[:,[1,3,4,5,7,9]]
regressor_OLS=sm.OLS(endog=y_train, exog = X_opt).fit()
regressor_OLS.summary()
X_test=X_test[:,[1,3,4,5,7,9]]

# Ajustando o novo modelo
regressor.fit(X_opt, y_train)

#Otimizando os Hiperparâmetros do modelo
from sklearn.model_selection import GridSearchCV
parameters = [{}]#Teste 1 - Valores Padrões - Resultado:0.9158
parameters = [{ #Teste 2
    'bootstrap': [True],
    'max_depth': [None, 10, 50],
    'max_features': [2, 3],
    'min_samples_leaf': [1,3],
    'min_samples_split': [2,3],
    'n_estimators': [10,50, 100] #Resultado:0,9206
    }]
parameters = [{ #Teste 3
    'bootstrap': [True],
    'max_depth': [40,50,60], #40
    'max_features': [2,3,4],#4
    'min_samples_leaf': [1],#1
    'min_samples_split': [3,4],#3
    'n_estimators': [100,200,500] #500
    }]#Resultado:0,9266
grid_search = GridSearchCV(estimator = regressor, param_grid = parameters,
                           cv =3, n_jobs=-1, )
grid_search = grid_search.fit(X_opt,y_train)
melhorresultado = grid_search.best_score_
melhoresparametros = grid_search.best_params_

regressor = RandomForestRegressor(n_estimators=500,bootstrap=True,max_depth=40,
                                  max_features=4,min_samples_leaf=3)
regressor.fit(X_opt, y_train)
```

Figura 7 – Aplicação do Algoritmo de Grid Search

Para uma análise inicial do modelo, utilizou-se a função *.predict* do nosso regressor para o base de dados de teste X_test. Foi realizada uma comparação direta de alguns valores entre os resultados previstos e o que de fato aconteceu com o vetor y_test, dados disponíveis na tabela 1. Análise que apesar de simples, já consegue nos dar uma indicação da qualidade do modelo.

Tabela 1 – Comparação entre os resultados do modelo e a realidade no número de bicicletas alugadas por hora

Linha	Valor Real (y_test)	Previsão (y_pred)
0	74	62
500	1421	1163
912	447	652,5
1188	437	418,3
1573	88	188,7
2004	1282	1305
2063	1539	1579,1

A tabela 2 consegue fornecer uma análise mais detalhada dos modelos, nela foram utilizados os parâmetros de R^2 Ajustado, que é capaz de nos mostrar a relação entre a qualidade do modelo e o número de variáveis utilizadas, penalizando valores mais altos de número de variáveis escolhidas; além do resultado obtido pelo parâmetro *.best_score* da nossa função do Grid Search.

Tabela 2 – Índices de Performance para os modelos

Tipo de Modelo	Valor R^2 Ajustado	Nota Grid Search
Modelo Inicial com parâmetros padrões	0,799	0,9171
Modelo reduzido com parâmetros padrões	0,799	0,9158
Modelo reduzido e otimizado com Grid Search	0,799	0,9266

Por último, através do código da figura 8 foi realizada uma comparação visual através de dois gráficos que comparam os valores estimados com a realidade na variável de interesse do problema, representados na figura 9 e 10. Foram pegos pontos em intervalos de 100 e 100 linhas para a base de dados inteira na figura 9 e intervalos de 5 em 5 para o gráfico da figura 10, que pega o intervalo inteiro do mês de Junho de 2016. Em azul estão os valores previstos pelo modelo e em vermelho os valores reais.

```
# Gráfico - Análise Geral
eixoX=np.arange(0,10320,100)
eixoy=X[range(0,10320,100),:]
eixoy=regressor.predict(eixoy[:,[1,3,4,5,7,9]])
plt.scatter(eixoX, y[range(0,10320,100)], color = 'red')
plt.plot(eixoX, eixoy, color = 'blue')
plt.title('Comparação entre os resultados reais e o modelo - Dataset Inteiro
plt.xlabel('Número da linha')
plt.ylabel('Nº de Bicicletas Alugadas')
plt.show()

# Gráfico2 - Análise do mês 6 de 2016
eixoX=np.arange(6288,7007,5)
eixoy=X[range(6288,7007,5),:]
eixoy=regressor.predict(eixoy[:,[1,3,4,5,7,9]])
plt.scatter(eixoX, y[range(6288,7007,5)], color = 'red')
plt.plot(eixoX, eixoy, color = 'blue')
plt.title('Comparação entre os resultados reais e o modelo - Junho de 2016')
plt.xlabel('Número da linha')
plt.ylabel('Nº de Bicicletas Alugadas')
plt.show()
```

Figura 8 – Código da montagem dos gráficos e previsão dos resultados

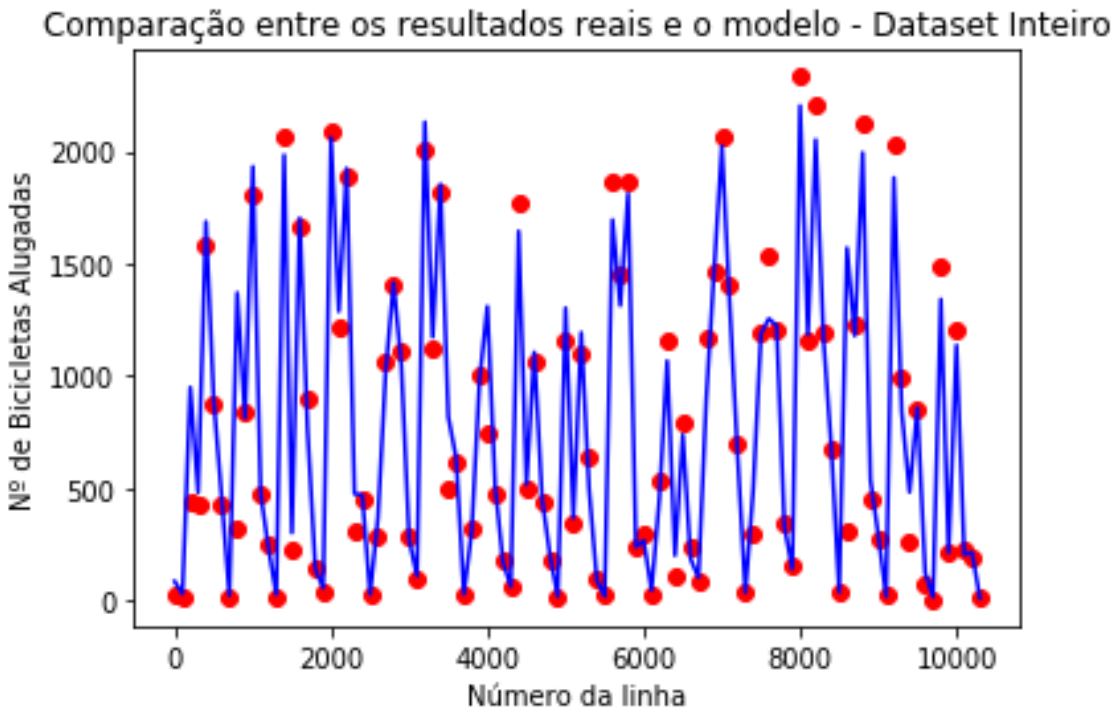

Figura 9 – Comparação entre a realidade em vermelho e o previsto em azul.

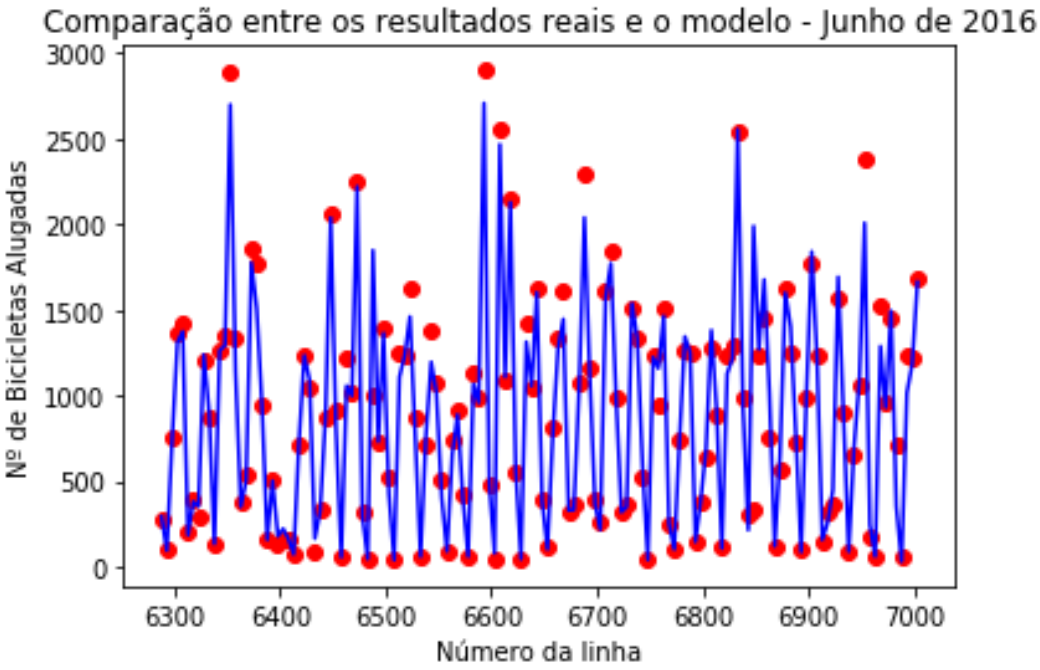

Figura 10 – Comparação para o mês de Junho de 2016 entre a realidade em vermelho e o previsto em azul.

Conclusão e Análise dos Resultados

A partir dos resultados obtidos através das diferentes análises, é possível concluir que o modelo ainda consegue ser melhorado, através de uma melhor escolha ou manipulação das variáveis, uma vez que foram obtidos valores de R^2 ajustado da ordem de 0,8.

Ou por uma pequena melhor predição dos valores da realidade, através de uma varredura mais profunda dos hiperparâmetros, já que foram obtidas notas no Grid Search da ordem de 90%. Contudo melhorias significativas nesse valore irá fazer com que o modelo sofra com “overfitting”, algo que ocorre com uma certa frequência para o algoritmo de “Random Forest” utilizado.

Porém, de maneira geral, o modelo de regressão utilizado foi capaz de se adequar bem aos valores da realidade, fornecendo respostas satisfatórias aos valores do número de aluguéis realizados por hora na cidade de Montreal, sendo capaz também de pegar o comportamento cíclico em relação à madrugada e aos horários de pico do dia a dia no aluguel de bicicletas de uma cidade, como é possível de ser visualizado nos gráficos da figura 9 e 10.

Referências

1. Leon Lovett. Merge multiple csv files. https://www.linkedin.com/pulse/merge-multiple-csv-files-python-leon-lovett/

2. Towardsdatascience. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

3. Stat. Backward Elimination https://www.stat.ubc.ca/~rollin/teach/643w04/lec/node42.html

4. Scikit http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html