Projeto 15

A **implementação de um Projeto de "Machine Learning" (Aplicação Web)** é um sistema abrangente que contém todos os estágios de um projeto de aprendizado de máquina, desde a coleta e o pré-processamento de dados até o desenvolvimento, a implementação e a manutenção contínua do modelo.

Se você quiser aprender a criar um modelo de aprendizado de máquina e implementá-lo em uma aplicação web, este projeto é para você.

Neste material, mostraremos como criar um modelo de aprendizado de máquina completo usando Python e Dash (uma estrutura de interface do usuário).

Implementando um Projeto de "Machine Learning": Passo a Passo

Para criar um modelo de aprendizado de máquina, primeiro você precisa treiná-lo, como é feito em todos para resolução dos problemas.

Depois de treinar o modelo, veja abaixo o processo que você pode seguir para transformá-lo em uma solução completa usando a estrutura dash:

> - Configure um novo aplicativo Dash;

> - Crie o layout do aplicativo usando o HTML e os componentes principais do Dash;
> - Inclua os campos de entrada, botões e áreas de exibição de saída;
> - Escreva as funções de retorno de chamada para definir a interatividade do aplicativo, como receber entradas do usuário, executar a previsão do modelo e exibir os resultados;
> - Carregar o modelo no aplicativo para usá-lo nas previsões.

Para economizar tempo, usaremos o modelo que treinamos para prever os preços dos imóveis.

Você pode encontrar esse artigo **aqui**.

Você pode baixar este conjunto de dados **aqui**.

Criando o Projeto Completo de "Machine Learning"

Iniciaremos essa tarefa lendo o conjunto de dados e treinando o modelo de aprendizado de máquina:

In [1]:
```
# Importando as Bibliotecas

import pandas as pd
import numpy as np
import dash
from dash import html, dcc, Input, Output, State
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [2]:
```
# Carregando o Arquivo CSV do Conjunto de Dados
real_estate_data = pd.read_csv("Real_Estate.csv")
```

In [3]:
```
# Visualizando a Base de Dados
real_estate_data.head(10)
```

Out[3]:

	Transaction date	House age	Distance to the nearest MRT station	Number of convenience stores	Latitude	Longitude	House price of unit area
0	2012-09-02 16:42:30.519336	13.3	4082.0150	8	25.007059	121.561694	6.488673
1	2012-09-04 22:52:29.919544	35.5	274.0144	2	25.012148	121.546990	24.970725
2	2012-09-05 01:10:52.349449	1.1	1978.6710	10	25.003850	121.528336	26.694267
3	2012-09-05 13:26:01.189083	22.2	1055.0670	5	24.962887	121.482178	38.091638
4	2012-09-06 08:29:47.910523	8.5	967.4000	6	25.011037	121.479946	21.654710
5	2012-09-06 14:18:34.142030	13.3	279.1726	2	24.994994	121.543823	36.972376
6	2012-09-06 15:07:23.396013	38.5	377.7956	3	25.009895	121.558955	27.637382
7	2012-09-07 07:57:25.291322	15.2	552.4371	5	24.997109	121.544377	44.116585
8	2012-09-07 10:06:48.384148	24.0	617.4424	3	24.987622	121.527841	49.071247
9	2012-09-07 13:21:33.254701	13.0	323.6550	8	24.978663	121.483457	43.114353

In [4]:
```
# Definição das variáveis independentes (features) e da variável dependente (target)
features = ['Distance to the nearest MRT station', 'Number of convenience stores', 'Latitude', 'Longitude']
target = 'House price of unit area'
```

```python
# Separação dos dados em X (entradas) e y (saída)
X = real_estate_data[features]
y = real_estate_data[target]

# Divisão dos dados em conjunto de treinamento (80%) e teste (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Lista de modelos para teste
modelos = {
    "Árvore de Decisão": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "XGBoost": XGBRegressor(n_estimators=100, random_state=42, objective="reg:squarederror"),
    "Regressão Linear": LinearRegression()
}

# Dicionário para armazenar os resultados
resultados = []

# Armazena o nome do último modelo treinado
ultimo_modelo_treinado = None

# Loop para treinar cada modelo e calcular as métricas
for nome, modelo in modelos.items():
    modelo.fit(X_train, y_train)  # Treinamento do modelo
    y_pred = modelo.predict(X_test)  # Previsão no conjunto de teste
    ultimo_modelo_treinado = nome  # Armazena o nome do último modelo

    # Cálculo das métricas de avaliação
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    # Armazena os resultados em uma lista
    resultados.append([nome, r2, mae, rmse])

# Criação do DataFrame com os resultados
df_resultados = pd.DataFrame(resultados, columns=["Modelo", "R²", "MAE", "RMSE"])

# Ordenar os modelos pelo melhor R² (quanto maior, melhor)
df_resultados = df_resultados.sort_values(by="R²", ascending=False).reset_index(drop=True)

# Exibição da tabela de resultados
print("\nResultados dos Modelos de Regressão:")
print(df_resultados)
print()
print(f"Último modelo treinado: {ultimo_modelo_treinado}")
```

```
Resultados dos Modelos de Regressão:
             Modelo        R²        MAE       RMSE
0   Regressão Linear  0.549673   9.518039  11.161514
1      Random Forest  0.516259   9.700825  11.568189
2            XGBoost  0.372561  10.506173  13.174822
3  Árvore de Decisão  0.006758  13.044347  16.576268

Último modelo treinado: Regressão Linear
```

Observação:

Este modelo preditivo foi desenvolvido com fins didáticos.

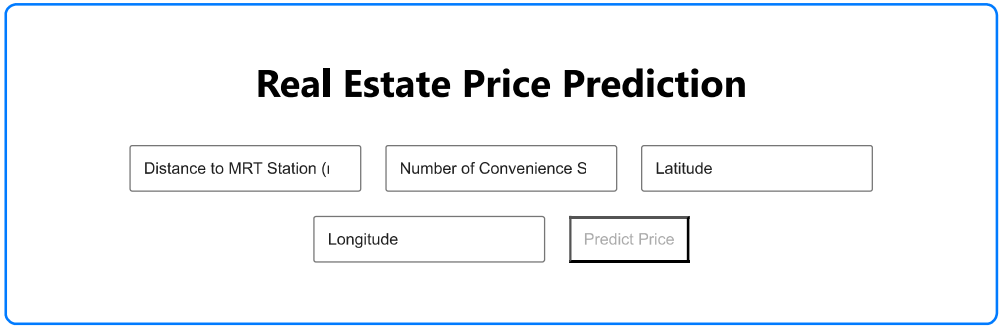
Em um projeto real, é essencial explorarmos diferentes abordagens e algoritmos para otimizar os indicadores de desempenho, tanto em termos de performance quanto na redução dos erros entre os valores reais e os valores preditos.

Agora, que acabamos de treinar o modelo de "Machine Learning" podemos dar continuidade no projeto de implementação da solução usando a Biblioteca Dash:

In [5]:

```python
# Importação das bibliotecas necessárias
import dash  # Framework para criar dashboards interativos
from dash import html, dcc, Input, Output, State  # Componentes para a interface
import pandas as pd  # Biblioteca para manipulação de dados

# Inicializa o aplicativo Dash
app = dash.Dash(__name__)

# Define o layout da aplicação
app.layout = html.Div([
    html.Div([
        # Título principal
        html.H1("Real Estate Price Prediction", style={'text-align': 'center'}),

        # Campos de entrada para as variáveis do modelo
        html.Div([
            dcc.Input(id='distance_to_mrt', type='number',
                      placeholder='Distance to MRT Station (meters)',
                      style={'margin': '10px', 'padding': '10px'}),

            dcc.Input(id='num_convenience_stores', type='number',
                      placeholder='Number of Convenience Stores',
                      style={'margin': '10px', 'padding': '10px'}),

            dcc.Input(id='latitude', type='number',
                      placeholder='Latitude',
                      style={'margin': '10px', 'padding': '10px'}),

            dcc.Input(id='longitude', type='number',
                      placeholder='Longitude',
                      style={'margin': '10px', 'padding': '10px'}),

            # Botão para realizar a previsão
            html.Button('Predict Price', id='predict_button', n_clicks=0,
                        style={'margin': '10px', 'padding': '10px', 'background-color': '#007BFF', 'color': 'white'}),
        ], style={'text-align': 'center'}),

        # Div para exibir a previsão do preço
        html.Div(id='prediction_output',
                 style={'text-align': 'center', 'font-size': '20px', 'margin-top': '20px'})
    ], style={'width': '50%', 'margin': '0 auto', 'border': '2px solid #007BFF',
              'padding': '20px', 'border-radius': '10px'})
])

# Define o callback para atualizar a previsão com base nos valores inseridos
@app.callback(
    Output('prediction_output', 'children'),  # Atualiza o texto exibido na tela
    [Input('predict_button', 'n_clicks')],  # O botão de previsão será o gatilho para a execução
    [State('distance_to_mrt', 'value'),  # Captura os valores inseridos nos campos de entrada
     State('num_convenience_stores', 'value'),
     State('latitude', 'value'),
     State('longitude', 'value')]
)

def update_output(n_clicks, distance_to_mrt, num_convenience_stores, latitude, longitude):
    """
    Função chamada quando o botão é pressionado.
    Verifica se os valores foram inseridos e realiza a previsão com o modelo treinado.
    """
    if n_clicks > 0 and all(v is not None for v in [distance_to_mrt, num_convenience_stores, latitude, longitude]):
        # Cria um DataFrame com os valores inseridos pelo usuário
        features = pd.DataFrame([[distance_to_mrt, num_convenience_stores, latitude, longitude]],
                                columns=['Distance to the nearest MRT station', 'Number of convenience stores', 'Latitude', 'Longitude'])

        # Faz a previsão do preço do imóvel
        prediction = model.predict(features)[0]
```

```
        # Retorna a previsão formatada
        return f'Predicted House Price of Unit Area: US$ {prediction:.2f}'

    elif n_clicks > 0:
        return 'Please enter all values to get a prediction'

    return ''  # Caso nenhum botão tenha sido pressionado ainda

# Executa o aplicativo
if __name__ == '__main__':
    app.run_server(debug=True)
```



Agora, vamos decompor esse código e entender cada parte do código:

importar dash
from dash import html, dcc, Input, Output, State
import pandas as pd

Nesse código, "dash" é a biblioteca principal do Dash.
O html e o dcc (Dash Core Components) são usados para criar HTML e componentes interativos.
Input, Output e State são usados para criar retornos de chamada no Dash (interatividade).

app = dash.Dash(__name__)

Essa linha inicializa um novo aplicativo Dash.

app.layout = html.Div([...])

Essa parte define a estrutura HTML do aplicativo usando os componentes HTML do Dash.
O layout inclui um título (html.H1), campos de entrada para a distância até a estação MRT, o número de lojas de conveniência, a latitude e a longitude (dcc.Input) e um botão para acionar a previsão (html.Button).

@app.callback(
Output('prediction_output', 'children'),
[Input('predict_button', 'n_clicks')],
[State('distance_to_mrt', 'value'),
State('num_convenience_stores', 'value'),
State('latitude', 'value'),
State('longitude', 'value')]
)

def update_output(n_clicks, distance_to_mrt, num_convenience_stores, latitude, longitude):

- É uma função de retorno de chamada que atualiza a saída (resultado da previsão) quando o botão "Predict Price" (Prever preço) é clicado;
- Output('prediction_output', 'children') indica que o conteúdo interno (filhos) do componente com id prediction_output será atualizado por essa chamada de retorno;
- A chamada de retorno usa o número de cliques no botão como Input e os valores dos quatro campos de entrada como State;
- A função update_output é executada quando o botão é clicado, usando os valores de entrada para gerar uma previsão;
- Dentro da função update_output, os inputs são verificados primeiro para garantir que os campos não estejam vazios;
- Em seguida, as entradas são organizadas em um Pandas DataFrame, de acordo com o formato esperado para o modelo. O método model.predict é chamado para gerar uma previsão. Isso pressupõe a existência de um modelo treinado denominado model, que pode ser acessado dentro deste script. A função retorna o preço previsto ou um prompt para inserir todos os valores.

if __name__ == '__main__':
app.run_server(debug=True)

- Essa parte executa o servidor do aplicativo quando o script é executado diretamente (**name** == '**main**'). debug=True ativa o modo de depuração, que fornece um depurador interativo no navegador e recarrega automaticamente o servidor quando o código é alterado;

Portanto este é um exemplo de como podemos colocar em produção um modelo de Machine Learning, fornecendo entradas e obtendo previsões.

Resumo

A criação de um modelo de "Machine Learning" e colocação do mesmo em produção é um sistema abrangente que envolve todos os estágios, desde a coleta e o pré-processamento de dados até o desenvolvimento, a implementação e a manutenção contínua do modelo.

Espero que este projeto sobre "Implementação de um Projeto de "Machine Learning" (Aplicação Web) usando Python tenha sido útil para seu aprendizado.

Referência do Projeto: https://thecleverprogrammer.com/2023/12/18/build-an-end-to-end-machine-learning-model/
Base de Dados: https://statso.io/real-estate-prediction-case-study/
Autor: Aman Kharwal - https://amankharwal.medium.com/
Projeto Estudado por: Giovani Aloísio da Luz
Disponível em: https://www.giovani-luz.com.br/