Parte 1: Best Subset Selection

Objetivo:

Os alunos irão:

- Aplicar o método de Best Subset Selection para selecionar as melhores variáveis
- Entender como escolher o melhor subconjunto

Descrição do Exercício:

1. Carregamento dos Dados

Carregue os dados do dataset Boston Housing:

```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_openml
# Instale bibliotecas necessárias no Google Colab
!pip install pandas numpy matplotlib scikit-learn statsmodels
# Instale bibliotecas necessárias no Google Colab
!pip install pandas numpy matplotlib scikit-learn statsmodels
# Carregar o dataset Boston Housing
data = fetch_openml(name="Boston", version=1, as_frame=True)
df = data.frame
# Variável resposta e preditores
X = df.drop(columns=["MEDV"])  # Todas as colunas, exceto a variável alvo
y = df["MEDV"]  # Variável alvo (preço das casas)
```

2. Implementação do Best Subset Selection

Utilize o Best Subset Selection para testar todos os conjuntos possíveis de variáveis.

```
import itertools
import time
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

def best_subset_selection(X, y):
    n_features = X.shape[1]
    subsets = []
    r2_values = []
    times = []
```

```
    start_time = time.time()
    # Testar todos os possíveis subconjuntos
    for k in range(1, n_features + 1):
        best_r2 = -np.inf
        best_subset = None
        for subset in itertools.combinations(range(n_features), k):
            X_subset = np.array(X.iloc[:, list(subset)])
            model = LinearRegression().fit(X_subset, y)
            r2 = r2_score(y, model.predict(X_subset))
            if r2 > best_r2:
                best_r2 = r2
                best_subset = subset
        subsets.append(best_subset)
        r2_values.append(best_r2)
        times.append(time.time() - start_time)
    return subsets, r2_values, times
# Aplicar Best Subset Selection
subsets, r2_values, times = best_subset_selection(X, y)
# Gráfico de R² vs Número de Variáveis
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.plot(range(1, X.shape[1] + 1), r2_values, marker="o")
plt.xlabel("Número de Variáveis")
plt.ylabel("R²")
plt.title("Número de Variáveis vs R² (Best Subset Selection)")
plt.grid(True)
plt.show()
print(f"Tempo total de execução: {times[-1]:.2f} segundos")
```

3. Validação Cruzada para Subconjuntos Selecionados

Nesta etapa, avaliaremos os subconjuntos selecionados por validação cruzada, utilizando o erro médio quadrático (MSE) como métrica de avaliação.

```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

def cross_validate_best_subset(X, y, subsets, k_folds=5):
    mse_values = []
    kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)
    for subset in subsets:
        mse_fold = []
        for train_index, test_index in kf.split(X):
```

```
            X_train = np.array(X.iloc[train_index, list(subset)])
            X_test = np.array(X.iloc[test_index, list(subset)])
            y_train = np.array(y.iloc[train_index])
            y_test = np.array(y.iloc[test_index])
            model = LinearRegression().fit(X_train, y_train)
            y_pred = model.predict(X_test)
            mse_fold.append(mean_squared_error(y_test, y_pred))
        mse_values.append(np.mean(mse_fold))
    return mse_values
mse_values = cross_validate_best_subset(X, y, subsets)
plt.figure(figsize=(8, 6))
plt.plot(range(1, X.shape[1] + 1), mse_values, marker="o")
plt.xlabel("Número de Variáveis")
plt.ylabel("MSE (Validação Cruzada)")
plt.title("Número de Variáveis vs MSE (Validação Cruzada)")
plt.grid(True)
plt.show()
```

4. Análise

1. Levando em conta a simplicidade e eficiência, você escolheria um subconjunto com quantas variáveis? Por que?
2. Observe o tempo de execução do algoritmo. Em que casos você usaria ele?

Parte 2: Regularização em Modelos de Regressão

1. Preparação dos Dados:

Carregue os dados do dataset California Housing:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing(as_frame=True)
df = data.frame
X = df.drop('MedHouseVal', axis=1)
y = df['MedHouseVal']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Função para plotar os gráficos
def plot_real_vs_pred(y_test, y_pred, title):
    plt.figure(figsize=(6,6))
```

```
    plt.scatter(y_test, y_pred, alpha=0.5, label='Valores Preditos')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label='Linha Ideal')
    plt.xlabel('Valores Reais')
    plt.ylabel('Valores Preditos')
    plt.title(title)
    plt.show()
```

2. Aplicação da Regressão Ridge

Aplique o modelo Ridge Regression:

```
alpha_r = 0.1
# Treinando o modelo Ridge
ridge_model = Ridge(alpha_r)
ridge_model.fit(X_train, y_train)
# Previsões no conjunto de teste
y_pred_ridge = ridge_model.predict(X_test)
# Plot para Ridge
plot_real_vs_pred(y_test, y_pred_ridge, 'Regressão Ridge')
# Cálculo do MSE
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print(f"MSE Ridge: {mse_ridge:.4f}")
```

3. Aplicação da Regressão Lasso:

Aplique o modelo Lasso Regression:

```
alpha_l = 0.1
# Treinando o modelo Lasso
lasso_model = Lasso(alpha_l)
lasso_model.fit(X_train, y_train)
# Previsões no conjunto de teste
y_pred_lasso = lasso_model.predict(X_test)
# Plot para Lasso
plot_real_vs_pred(y_test, y_pred_lasso, 'Regressão Lasso')
# Cálculo do MSE
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print(f"MSE Lasso: {mse_lasso:.4f}")
```

5. Impacto da Multicolinearidade

Nesta etapa, avaliaremos o desempenho de Ridge e Lasso em datasets sintéticos com diferentes graus de correlação entre as variáveis.

```
from sklearn.datasets import make_regression

def calculate_mse(effective_ranks):
    mse_ridge, mse_lasso = [], []
    for rank in effective_ranks:
        X, y = make_regression(n_samples=1000, n_features=50, n_informative=10, noise=0.1,
effective_rank=rank, random_state=42)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        for model, mse_list in [(Ridge(alpha=1.0), mse_ridge), (Lasso(alpha=0.1,
max_iter=10000), mse_lasso)]:
            model.fit(X_train, y_train)
            mse_list.append(mean_squared_error(y_test, model.predict(X_test)))
    return mse_ridge, mse_lasso

effective_ranks = np.linspace(1, 50, 10).astype(int)
mse_ridge, mse_lasso = calculate_mse(effective_ranks)
plt.figure(figsize=(8, 6))
plt.plot(effective_ranks, mse_ridge, label='Ridge', marker='o', linestyle='-', linewidth=2)
plt.plot(effective_ranks, mse_lasso, label='Lasso', marker='o', linestyle='-', linewidth=2)
plt.xlabel('Grau de Correlação (effective_rank)')
plt.ylabel('Erro Quadrático Médio (MSE)')
plt.title('Impacto da Multicolinearidade no MSE')
plt.legend()
plt.grid(True)
plt.show()
```

6. Discussão

1. Qual dos dois modelos se saiu melhor no dataset California Housing?
2. Altere os valores de `alpha` nos dois modelos para 1 e depois para 10. O que acontece?
3. Após analisar o desempenho dos dois métodos com o aumento da colinearidade, a que conclusão você chega?