**Documento Explicativo do Projeto de Classificação de Modelos de Moto da Mottu**

Este documento detalha o código Python fornecido, que implementa um fluxo completo de Machine Learning, desde a geração de dados sintéticos até o treinamento, avaliação e visualização de um modelo de classificação. O objetivo principal é prever o Modelo de uma moto com base em sua Condição e Pátio de localização.

**1. Visão Geral do Projeto**

O projeto consiste em um pipeline de Machine Learning para classificar modelos de motos (Mottu Sport, Mottu E, Mottu Pop) a partir de características categóricas como Condição (Nova, Usada, Manutenção) e Pátio (localização da unidade). O processo envolve as seguintes etapas:

1. **Geração de Dados Sintéticos:** Criação de um dataset simulado para treinamento e teste do modelo, incluindo a identificação do chaveiro Bluetooth.
2. **Pré-processamento de Dados:** Transformação de dados categóricos em numéricos.
3. **Divisão de Dados:** Separação do dataset em conjuntos de treino e teste.
4. **Treinamento do Modelo:** Utilização do algoritmo XGBoost para aprender os padrões nos dados.
5. **Avaliação do Modelo:** Medição da acurácia e análise da matriz de confusão para entender o desempenho.

**2. Geração de Dados Sintéticos**

**Frameworks e Técnicas:**

* **pandas:** Utilizado para criar e manipular o DataFrame, além de salvar os dados em formato CSV.
* **random:** Usado para introduzir aleatoriedade na geração das características, simulando uma distribuição não uniforme.

**Justificativas:**

A geração de dados sintéticos é crucial quando não se tem acesso a um dataset real ou quando se precisa de um grande volume de dados com características específicas para testar um algoritmo. A função gerar\_dados\_modelo simula uma lógica de negócio onde a probabilidade de um Modelo de moto, sua Condição e Pátio de origem são ponderadas. Por exemplo, Mottu Sport tem maior probabilidade de ser Nova, enquanto Mottu Pop é mais frequentemente Usada. Além disso, a distribuição de pátios é específica para cada modelo, refletindo possíveis especializações ou volumes de estoque por unidade. A inclusão do Dispositivo Bluetooth (chaveiro) simula um identificador único para cada moto, que pode ser útil para rastreamento e outras funcionalidades.

**Implementação:**

A função gerar\_dados\_modelo utiliza random.random() para sortear o modelo principal e, em seguida, random.choices() com pesos (weights) para determinar a condição e o pátio, garantindo a distribuição desejada. Um identificador único para o Dispositivo Bluetooth (chaveiro) é gerado para cada registro. Um DataFrame é construído com 5000 registros e salvo como motos\_dataset.csv.

**3. Pré-processamento dos Dados**

**Frameworks e Técnicas:**

* **pandas:** Para carregar o dataset (.read\_csv()) e manipular as colunas.
* **sklearn.preprocessing.LabelEncoder:** Uma técnica de codificação para transformar variáveis categóricas (texto) em representações numéricas inteiras.

**Justificativas:**

A maioria dos algoritmos de Machine Learning, incluindo o XGBoost, opera com dados numéricos. Portanto, as colunas categóricas (Condição, Pátio, Modelo) precisam ser convertidas. LabelEncoder é uma escolha simples e eficaz para este cenário, onde a ordem das categorias não implica em uma relação ordinal significativa. Cada categoria única recebe um número inteiro. O Dispositivo Bluetooth, sendo um identificador único, não é codificado para o modelo de classificação neste projeto, mas sua presença no dataset é mantida.

**Implementação:**

Três instâncias de LabelEncoder são criadas: le\_cond, le\_patio, e le\_modelo. Elas são aplicadas individualmente às colunas Condição, Pátio e Modelo, criando novas colunas codificadas (Cond\_Enc, Patio\_Enc) e a variável y\_encoded para o target.

**4. Divisão dos Dados**

**Frameworks e Técnicas:**

* **sklearn.model\_selection.train\_test\_split:** Uma função padrão para dividir um dataset em conjuntos de treinamento e teste.

**Justificativas:**

A divisão dos dados é fundamental para avaliar a capacidade de generalização do modelo. O conjunto de treinamento é usado para o modelo aprender os padrões, enquanto o conjunto de teste (dados "nunca vistos") é usado para simular o desempenho do modelo em dados novos e reais. Uma proporção de 70% para treinamento e 30% para teste (test\_size=0.3) é uma prática comum. random\_state=42 garante a reprodutibilidade da divisão.

**Implementação:**

As *features* (X) são definidas como Cond\_Enc e Patio\_Enc, e o *target* (y) como y\_encoded. A função train\_test\_split é então utilizada para criar X\_train, X\_test, y\_train, y\_test.

**5. Modelo de Machine Learning (XGBoost)**

**Frameworks e Técnicas:**

* **xgboost.XGBClassifier:** Uma implementação otimizada de árvores de decisão baseadas em gradiente (Gradient Boosting).

**Algoritmo e Justificativas:**

**XGBoost (Extreme Gradient Boosting)** é um algoritmo de *ensemble* que constrói uma série de árvores de decisão de forma sequencial, onde cada nova árvore tenta corrigir os erros das árvores anteriores. É conhecido por sua alta performance, velocidade e capacidade de lidar com diversos tipos de dados, incluindo dados categóricos (após codificação).

**Justificativas para a escolha do XGBoost:**

* **Performance:** Frequentemente vence competições de Machine Learning devido à sua robustez e otimização.
* **Lida bem com dados categóricos:** Embora exija codificação prévia, ele é eficaz na identificação de padrões em features discretas.
* **Regularização:** Possui mecanismos de regularização embutidos para evitar *overfitting*.
* **Hiperparâmetros Otimizados:** O código já inclui hiperparâmetros ajustados manualmente (max\_depth=6, learning\_rate=0.1, n\_estimators=150).
  + max\_depth: Limita a profundidade das árvores, controlando a complexidade do modelo.
  + learning\_rate: Reduz a contribuição de cada árvore, ajudando a prevenir *overfitting*.
  + n\_estimators: Número de árvores a serem construídas.
  + eval\_metric='mlogloss': Métrica de avaliação para problemas de classificação multi-classe.

**Implementação:**

Uma instância de XGBClassifier é criada com os hiperparâmetros definidos. O modelo é então treinado usando model.fit(X\_train, y\_train).

**6. Treinamento e Avaliação do Modelo**

**Frameworks e Técnicas:**

* **sklearn.metrics.accuracy\_score:** Para calcular a acurácia do modelo.

**Justificativas:**

Após o treinamento, é essencial avaliar o desempenho do modelo. A acurácia é uma métrica simples que indica a proporção de previsões corretas.

**Implementação:**

As previsões são geradas no conjunto de teste (y\_pred = model.predict(X\_test)), e a acurácia é calculada e impressa.

**7. Análise da Matriz de Confusão**

**Frameworks e Técnicas:**

* **sklearn.metrics.confusion\_matrix:** Para gerar a matriz de confusão.
* **seaborn:** Para criar visualizações estatísticas, especificamente um *heatmap*.
* **matplotlib.pyplot:** Para configurar e exibir o gráfico.

**Justificativas:**

A matriz de confusão oferece uma visão mais detalhada do desempenho do classificador do que apenas a acurácia. Ela mostra o número de previsões corretas e incorretas para cada classe, permitindo identificar onde o modelo está acertando e onde está errando (por exemplo, quais classes estão sendo confundidas). O *heatmap* facilita a interpretação visual da matriz.

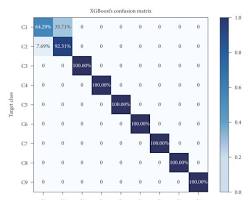
**Implementação:**

A matriz de confusão (cm) é gerada usando y\_test (valores reais) e y\_pred (valores previstos). As classes originais (labels = le\_modelo.classes\_) são usadas para rotular os eixos do *heatmap*. O gráfico é configurado com matplotlib e visualizado com seaborn.heatmap.

**8. Arquitetura Desenvolvida**

A arquitetura da solução segue um padrão comum em projetos de Machine Learning:

1. **Módulo de Geração de Dados:** Responsável por criar o dataset inicial, simulando dados do mundo real com distribuições específicas, incluindo o identificador do chaveiro Bluetooth.
2. **Módulo de Pré-processamento:** Prepara os dados brutos para o consumo do modelo, realizando codificação de variáveis categóricas.
3. **Módulo de Treinamento do Modelo:** Engloba a divisão dos dados e o treinamento do algoritmo de Machine Learning (XGBoost).
4. **Módulo de Avaliação e Visualização:** Analisa o desempenho do modelo através de métricas como acurácia e ferramentas visuais como a matriz de confusão.

[[](https://www.researchgate.net/figure/The-confusion-matrix-of-the-XGBoost-model_fig10_351140988)www.researchgate.net](https://www.researchgate.net/figure/The-confusion-matrix-of-the-XGBoost-model_fig10_351140988)

Essa estrutura modular facilita a manutenção, o teste e a escalabilidade do projeto, permitindo que cada etapa seja desenvolvida e aprimorada de forma independente.