Classificação - 9 categorias de sadt em ML

#### 1. Data Dependency (Dependência de Dados - DTD)

É a categoria mais comum, respondendo por mais de 30% dos débitos encontrados. Caracteriza-se por problemas relacionados aos dados e ao código que os processa, armazena ou configura. Como a funcionalidade do software de ML é "fortemente dependente da qualidade, estrutura e consistência dos dados de treinamento", qualquer atalho ou problema aqui gera um débito significativo.

* Exemplos: Tarefas de pré-processamento de dados que precisam ser melhoradas, problemas na forma como os dados são armazenados ou representados (ex: em um JSON), e configurações de qualidades dos dados de treinamento.

#### 2. Code Dependency (Dependência de Código - CDD)

Este débito ocorre quando um trecho de código depende de outro código, muitas vezes externo, para funcionar corretamente. A dívida se manifesta como uma "espera" por uma condição a ser atendida antes que uma melhoria possa ser feita. É mais comum em ferramentas de ML (frameworks) do que em aplicações.

* Exemplos: Esperar por uma atualização em uma biblioteca externa, código "cola" (*glue code*) que apenas serve para conectar sistemas, ou problemas de compatibilidade de versão de APIs.

#### 3. Awareness (Consciência/Conhecimento)

Esta categoria agrupa débitos que surgem da falta de conhecimento ou compreensão do desenvolvedor. Muitas vezes, aparecem na forma de perguntas nos comentários do código. A natureza de "caixa-preta" de muitos modelos de ML contribui para esse tipo de débito.

* Exemplos: Dúvidas sobre a correção de um algoritmo, incerteza sobre o comportamento de um modelo (interpretabilidade), ou falta de conhecimento de domínio específico para a aplicação.

#### 4. Modularity (Modularidade - MDL)

Refere-se a débitos de má estruturação do software, onde existem fronteiras fracas ou inexistentes entre os subsistemas. Isso torna a manutenção e a evolução do código difíceis, pois mudanças em um lugar causam efeitos cascata em outros. O artigo sugere que isso pode ser comum em aplicações de ML devido à transição de código de Jupyter Notebooks (interativos) para um design mais estruturado.

* Exemplos: Falta de abstrações (não dividir o código em funções/classes reutilizáveis) e "erosão de fronteira" (código de teste misturado com código de produção).

#### 5. Configurable Options (Opções Configuráveis - CFO)

Este é um débito muito específico de ML, relacionado a problemas no código que configura o próprio modelo. Os autores dividem essa categoria em três focos principais:

* Subtipos: Débitos na configuração dos pesos do modelo, na configuração das camadas do modelo e na configuração dos hiperparâmetros.

#### 6. Scalability (Escalabilidade - SCL)

Este débito diz respeito à capacidade do software de crescer e operar de forma robusta em um ambiente de produção completo. Envolve o risco de levar soluções de pequena escala para sistemas maiores.

* Exemplos: Usar um protótipo (código de pequena escala) diretamente em um sistema de produção, o que é considerado perigoso, e a falta de medidas para garantir a robustez do sistema.

#### 7. Readability (Legibilidade - RDB)

Caracteriza o débito onde o código é difícil de ler e entender, o que prejudica a manutenção. O artigo destaca que o código de modelos de ML pode ter problemas de legibilidade únicos, que não ocorrem em software tradicional.

* Exemplo: Variáveis de matriz temporárias com nomes ruins ou lógica confusa no código do modelo.

#### 8. Performance (Desempenho - PRF)

Este débito está diretamente ligado ao desempenho do fluxo de trabalho de ML. O artigo foca especificamente na "Qualidade da Predição", indicando que atalhos ou problemas no código podem afetar negativamente a performance preditiva do modelo.

#### 9. Duplicate Code Elimination (Eliminação de Código Duplicado - DCE)

É a categoria menos comum encontrada no estudo. Este débito é causado pela prática de copiar e colar código em vez de criar soluções reutilizáveis. O artigo destaca duas áreas específicas em ML onde isso ocorre com frequência:

* Onde ocorre: No próprio código do modelo e no código de extração de características (*feature extraction*).

# Exemplos de cada categoria 1. Data Dependency (Dependência de Dados)

# Exemplo de SATD: TODO: normalize true states (TODO: normalizar os estados verdadeiros).

# Característica: Este comentário admite que uma etapa crucial de pré-processamento de dados (normalização) está faltando ou precisa ser implementada. A performance e o comportamento do modelo dependem diretamente dessa normalização, criando uma dívida que reside na forma como os dados são tratados.

#### 2. Code Dependency (Dependência de Código)

# Exemplo de SATD: TODO add test for keypoints once their handling was improved in Convolve (TODO: adicionar teste para os pontos-chave assim que seu manuseio for melhorado no Convolve).

# Característica: O desenvolvedor está adiando a criação de um teste porque ele depende de uma melhoria que precisa ser feita em outra parte do sistema (o módulo Convolve). A tarefa está "em espera" (*on-hold*), dependendo de outro código ser alterado primeiro.

#### 3. Awareness (Consciência/Conhecimento)

# Exemplo de SATD: TODO: figure out what exactly size does (TODO: descobrir o que o parâmetro 'size' faz exatamente).

# Característica: Este comentário é uma pergunta direta. O desenvolvedor não tem certeza sobre a funcionalidade de um parâmetro (size). Essa falta de conhecimento ou "consciência" sobre a API ou o código gera uma dívida, pois o uso incorreto pode levar a bugs ou comportamento inesperado.

#### 4. Modularity (Modularidade)

# Exemplo de SATD: TODO: Split into separate functions (TODO: Dividir em funções separadas).

# Característica: O código provavelmente está em um bloco monolítico e difícil de manter. O desenvolvedor admite que a solução correta seria refatorá-lo, quebrando-o em funções menores e mais especializadas, melhorando a modularidade, o reuso e a manutenibilidade do código.

#### 5. Configurable Options (Opções Configuráveis)

# Exemplo de SATD: TODO: experiment with more layers (TODO: experimentar com mais camadas).

# Característica: Aqui, o desenvolvedor reconhece que a arquitetura atual do modelo (o número de camadas) é uma solução temporária ou não otimizada. A dívida consiste em não ter explorado configurações melhores para a estrutura do modelo, o que afeta diretamente seu potencial de performance.

#### 6. Scalability (Escalabilidade)

# Exemplo de SATD: TODO: This matching process is slow. Make it faster; avoid loops where possible. (TODO: Este processo de correspondência é lento. Torná-lo mais rápido; evitar loops onde possível).

# Característica: O comentário admite que o código atual é um protótipo ineficiente (lento, usando loops em vez de operações vetorizadas, algo comum em ML). Essa solução pode funcionar em pequena escala, mas não é escalável para um sistema de produção, gerando uma dívida de escalabilidade.

#### 7. Readability (Legibilidade)

# Exemplo de SATD: TODO: This next code is dense and confusing. Clean up at some point. (TODO: Este próximo código é denso e confuso. Limpar em algum momento).

# Característica: Este é um exemplo clássico. O desenvolvedor admite abertamente que o código é de difícil leitura. Embora funcional, sua complexidade torna a manutenção futura muito mais difícil e cara, caracterizando um débito de legibilidade.

#### 8. Performance (Desempenho)

# Exemplo de SATD: TODO: compare the performance! (TODO: comparar a performance!).

# Característica: O desenvolvedor implementou uma solução, mas ainda não avaliou ou comparou seu desempenho com outras alternativas. A dívida está na incerteza sobre a qualidade da predição do modelo. A solução pode não ser a ideal, afetando a performance final do sistema.

#### 9. Duplicate Code Elimination (Eliminação de Código Duplicado)

# Exemplo de SATD: XXX de-duplicate this with code from Montage somewhere? (XXX: remover a duplicação disso com o código de 'Montage' em algum lugar?).

# Característica: O comentário admite explicitamente que o trecho de código atual é uma cópia de outro código que existe em outra parte do projeto (Montage). Manter duas cópias do mesmo código é uma dívida, pois qualquer correção ou melhoria precisará ser feita em ambos os lugares.

# 