Geração de Dados Sintéticos

Escola de Verão 2025

Alexandre Teles <alexandre.teles@inctdd.org>

Índice

O que esperar deste curso?

Neste curso, vamos aprender a gerar dados sintéticos para treinar modelos de aprendizado de máquina. Vamos aprender a criar dados estruturados, gerar dados sintéticos e validar esses dados com modelos de linguagem.

O que preciso saber?

Para acompanhar este curso, é necessário ter conhecimentos básicos de Python. Além disso, é importante ter conhecimentos básicos de estatística. Também é desejável ter familiaridade com conceitos de aprendizado de máquina e com processamento de dados.

Não sei nada de Python. Posso acompanhar?

Sim! Este curso foi pensado para ser acessível a todos os públicos. Se você não tem conhecimentos de Python, não se preocupe. Vamos aprender juntos!

Dados não estruturados

- **Definição:**
  - Não possuem formato ou organização predefinida.
  - Difíceis de analisar diretamente; exigem pré-processamento.
- **Exemplos:**
  - Texto livre (documentos, e-mails, posts).
  - Imagens, áudio, vídeo.
- **Desafios:**
  - Complexidade na extração de informações.
  - Requerem técnicas como Processamento de Linguagem Natural (PLN).

Dados estruturados

- **Definição:**
  - Informações altamente organizadas (ex. em formato tabular).
  - Esquema predefinido: cada atributo possui um tipo de dado específico (número, texto, data, etc.).
- **Exemplos:**
  - Tabelas de Bancos de Dados Relacionais (SQL).
  - Planilhas (Excel, CSV).
  - *Arquivos JSON (com estrutura rígida).*
- **Vantagens:**
  - Fácil pesquisa e análise por máquinas.
  - Simplicidade na modelagem e consulta.
  - Integração eficiente com várias ferramentas.

Dados estruturados vs. não estruturados

Característica	Dados Estruturados	Dados Não Estruturados
Organização	Formato tabular (linhas/colunas)	Sem formato predefinido
Esquema	Predefinido (tipos de dados)	Ausente ou flexível
Pesquisa/Análise	Fácil e direta	Complexa, requer pré-processamento
Exemplos	Bancos de dados SQL, planilhas	Texto, imagens, áudio, vídeo

Por que dados estruturados são cruciais para geração sintética?

- **Esquemas como "receitas" para LLMs:**
  - Estruturas JSON/SQL funcionam como "instruções precisas" para modelos de linguagem
  - Garantem que dados gerados sigam exatamente o formato necessário
  - Permitem validação e conformidade com regras de negócio
- **Benefícios práticos:**
  - ✓ Consistência na geração
  - ✓ Dados prontos para uso em sistemas
  - ✓ Redução de pós-processamento
  - ✓ Maior controle sobre o output

Introdução ao Formato JSON

O que é JSON?

- **JSON** (JavaScript Object Notation) é um formato de dados leve e fácil de ler.
- É amplamente utilizado para troca de dados entre sistemas.
- É baseado em pares de chave-valor, organizados em objetos e arrays.
- Podemos representar objetos, arrays, strings, números, booleanos e valores nulos.
- É suportado por várias linguagens de programação.
- Dicionários em Python são equivalentes a objetos JSON.

Exemplo de JSON

```
{
  "nome": "João",
  "idade": 30,
  "cidade": "São Paulo",
  "interesses": ["Python", "Data Science"]
}
```

🟦 Comparação entre JSON e outros formatos

- **JSON vs XML:**
  - JSON é mais leve e fácil de ler.
  - XML suporta dados mais complexos, mas com maior sobrecarga.
- **JSON vs CSV:**
  - JSON suporta hierarquia de dados, enquanto CSV é plano.
  - CSV é mais eficiente para dados tabulares simples.
- **JSON vs YAML:**
  - JSON é mais rígido em sintaxe, enquanto YAML é mais flexível e legível.
  - YAML é frequentemente usado para configuração, enquanto JSON é popular para APIs.

JSON Schema

- **JSON Schema** é uma especificação para validar dados JSON.
- Permite definir regras para tipos de dados, formatos, valores mínimos/máximos, etc.
- É útil para garantir a conformidade de dados gerados sinteticamente.
- Podemos definir esquemas complexos com validações detalhadas.
- Ferramentas como `jsonschema` em Python permitem validar dados contra um schema.
- No entanto, JSON Schema não é tão flexível quanto `Pydantic` ou ``msgspec`, e por isso é menos usado em geração de dados sintéticos em Python.

Exemplo de JSON Schema

```
{
  "type": "object",
  "properties": {
    "nome": { "type": "string" },
    "idade": { "type": "integer" },
    "cidade": { "type": "string" },
    "interesses": {
      "type": "array",
      "items": { "type": "string" }
    }
  },
  "required": ["nome", "idade"]
}
```

Dicas de Tipos (Type Hints) em Python

Python é uma linguagem dinâmica

Considere a função abaixo:

```python
def media(*numeros):
  if not numeros:
    return 0
  return sum(numeros) / len(numeros)
```

- **Problema:** Quem chama a não sabe quais tipos de argumentos são aceitos e retornados, mas a função aceita qualquer tipo de argumento. Se um argumento não for numérico, a função falhará.

Type Hints

São anotações de tipo que permitem indicar explicitamente os tipos esperados em variáveis, parâmetros e retornos de funções. Embora opcionais, são valiosas para análise estática, autocomplete em IDEs e documentação de código. Por exemplo:

```python
def media(*numeros: float) -> float:
  if not numeros:
    return 0
  return sum(numeros) / len(numeros)
```

IMPORTANTE: Type hints são apenas anotações e não afetam o comportamento do código em runtime.

🔷 Porque Type Hints são cruciais para geração de dados sintéticos?

O uso de dicas de tipo oferece um "contrato explícito" que LLMs podem seguir, reduzindo ambiguidades na geração de dados.

- **Benefícios na Geração Sintética:**
  - ✓ Validação automática de tipos
  - ✓ Constraints explícitos (ranges, formatos)
  - ✓ Melhor qualidade dos dados gerados
  - ✓ Redução de erros em runtime
  - ✓ Aumento da consistência entre diferentes fontes de dados

Resultado: Dados sintéticos mais consistentes, realistas e prontos para uso.

TypedDicts

- Permitem definir dicionários com chaves e valores de tipos específicos.
- TypedDicts são *exatamente* iguais aos dicionários normais.
- A verificação de tipos ocorre apenas durante a análise estática.
- É importante ter cuidado com a adição de chaves não tipadas para evitar erros posteriores.

Exemplo de TypedDict

```python
from typing import TypedDict

class Pessoa(TypedDict):
  nome: str
  idade: int
  cidade: str
  interesses: list[str]

joao: Pessoa = {
  "nome": "João",
  "idade": 30,
  "cidade": "São Paulo",
  "interesses": ["Python", "Data Science"]
}
```

Dataclasses

- **Dataclasses** são uma forma concisa de definir classes de dados em Python.
- Permitem definir classes com atributos tipados de forma simples e eficiente.
- É possível implementar validações e lógica de negócios dentro das Dataclasses.
- Dataclasses são mais verbosas que TypedDicts, mas oferecem mais flexibilidade.
- A verificação de tipos também ocorre apenas durante a análise estática.

Exemplo de Dataclass

```python
from dataclasses import dataclass

@dataclass
class Pessoa:
  nome: str
  idade: int
  cidade: str
  interesses: list[str]

joao = Pessoa(
  nome="João",
  idade=30,
  cidade="São Paulo",
  interesses=["Python", "Data Science"]
)
```

Pydantic

- **Pydantic** é uma biblioteca Python para validação de dados e serialização.
- Permite definir modelos de dados com validações detalhadas.
- É amplamente utilizado em geração de dados sintéticos para garantir a conformidade dos dados gerados.
- Oferece validação de dados em tempo de execução, ao contrário de TypedDicts e Dataclasses.
- Pydantic é mais lento que TypedDicts e Dataclasses.

Exemplo de modelo Pydantic

```python
from pydantic import BaseModel

class Pessoa(BaseModel):
  nome: str
  idade: int
  cidade: str
  interesses: list[str]

joao = Pessoa(
  nome="João",
  idade=30,
  cidade="São Paulo",
  interesses=["Python", "Data Science"]
)
```

Engenharia de Prompts

- **Prompts** são instruções que guiam modelos de linguagem na geração de texto.
- Permitem controlar o conteúdo, estilo e formato dos dados gerados.
- Podem ser usados para gerar dados estruturados, como JSON.
- Prompts bem projetados são essenciais para a geração de dados sintéticos de alta qualidade.

Vamos analisar a estrutura de um prompt?

▒▒ Prompt para análise de sentimentos

```
Você é um especialista em análise de sentimentos. Dado um texto, você deve classificá-lo
como positivo, negativo ou neutro. Suas respostas devem ser estruturadas em JSON, com os
campos "texto", "sentimento" e "razão". Ao escrever a razão, explique por que você
classificou o texto daquela maneira. Lembre-se de ser claro e objetivo. Aqui está um
exemplo de entrada:
{
  "texto": "Eu amo esse filme!",
}
E aqui um exemplo de saída:
{
  "texto": "Eu amo esse filme!",
  "sentimento": "positivo",
  "razão": "O texto contém a palavra 'amo', que é um indicativo de sentimento positivo."
}
```

- Função: descreva de forma clara e concisa o papel a ser adoptado pelo modelo.
- Entrada: Explique o tipo de entrada que o modelo deve esperar.
- Processamento: Para cada entrada, descreva o que o modelo deve fazer.
- Saída: Descreva o formato da saída esperada.
- Exemplos: Forneça exemplos de entrada e saída para ilustrar o comportamento esperado.

Engenharia de Prompts: Dicas Práticas

- **Seja claro e objetivo:** Evite ambiguidades e instruções vagas.
- **Forneça exemplos:** Ilustre o comportamento esperado com exemplos claros.
- **Use linguagem simples:** Evite jargões técnicos e termos complexos.
- **Seja consistente:** Evite mudanças de direção ou formato durante os prompts.
- **Revise e teste:** Verifique se os prompts são compreensíveis e eficazes.
- **Gerencie a complexidade:** Simplifique fatores que possam complicar a compreensão do modelo.
- **Utilize LLMs:** Modelos de linguagem podem ser usados para gerar ou melhorar prompts automaticamente.
- **Por vezes, instrua em inglês:** A maioria dos modelos de linguagem são treinados em inglês. Instruções em inglês podem melhorar a compreensão e desempenho do modelo, ainda que o texto gerado seja em outro idioma.
- **Formate o texto:** Use formatação adequada para destacar partes importantes do prompt. Sempre em Markdown.
- **Evite espaços em branco:** Evite espaços em branco desnecessários, que podem confundir o modelo.
- **Instrua em pequenos passos:** Divida instruções complexas em passos menores e mais simples.
- **Vamos pensar passo a passo:** Instrua o modelo a pensar passo a passo, descrevendo cada etapa do processo.
- **Solicite razões e justificativas:** Instrua o modelo a fornecer razões e justificativas para suas respostas, quando necessário.

Geração de Dados com Instructor

- **Instructor** é uma biblioteca Python para geração de dados estruturados com modelos de linguagem. Construído sobre a biblioteca Pydantic.
- Permite definir esquemas de dados com validações detalhadas.
- Oferece uma API simples e intuitiva para gerar dados sintéticos.
- Pode ser usado para gerar dados de teste, treinamento e validação.
- Instructor é altamente customizável e extensível.
- Tem suporte a múltiplos provedores de modelos de linguagem (OpenAI, Cohere, Mistral, etc).

--> Notebook!

Obrigado por participar

Perguntas? Comentários? Dúvidas?

Entre em contato: **alexandre.teles@inctdd.org** (*mailto:alexandre.teles@inctdd.org*)