Documentação

Para realizar este exercício, for necessário utilizar algumas tecnologias e conceitos relacionados ao processamento de dados em tempo real, armazenamento em base de dados (neste caso uma base de dados não relacional), processamento em streaming e algoritmos de Machine Learning.

Este exercício foi dividido em várias etapas:

**Etapa 1:** Leitura dos dados e armazenamento numa base de dados (MongoDB):

Foi criado um script python, denominado de “csv\_to\_database\_mongo.py”, projetado para ler dados de um arquivo CSV, processá-los e inserir os dados processados numa base de dados MongoDB. O objetivo deste script é preparar o conjunto de dados para análises e consultas posteriores. O exercício envolve o tratamento de dados relacionados à sustentabilidade de cidades com base em fatores sociais, ambientais e econômicos.

Funcionalidade do código:

1. Importação das Bibliotecas:

O script começa por importar as bibliotecas necessárias:

* ***argparse***: Esta biblioteca é usada para analisar argumentos da linha de comandos, tornando fácil argumentos como entradas para o script.
* ***pandas***: Esta biblioteca é usada para manipulação de dados e, neste script, é utilizada para ler os dados do arquivo CSV e convertê-los num DataFrame do pandas.
* ***pymongo***: Esta biblioteca fornece uma interface para interagir com a base de dados MongoDB.

1. Definir a Conexão do MongoDB:

A função *connect\_to\_mongodb()* é definida para estabelecer uma conexão com a base de dados MongoDB em execução na máquina local. Utiliza a string de conexão padrão mongodb://localhost:27017/ para se conectar.

1. Inserir Dados no MongoDB:

A função *insert\_data\_to\_mongodb(csv\_file, collection\_name)* é responsável por processar os dados e inseri-los no MongoDB.

Funciona da seguinte forma:

a. Ler Dados do CSV: A função lê os dados do arquivo CSV especificado usando a função read\_csv() do pandas. Estes dados são armazenados num DataFrame do pandas (data\_df).

b. Converter DataFrame numa Lista de Dicionários: Para facilitar a inserção no MongoDB, o DataFrame é convertido numa lista de dicionários e, para isso, é utilizado o método to\_dict(orient='records'). Cada linha do DataFrame torna-se um dicionário na lista.

c. Conectar ao MongoDB: A função chama a função connect\_to\_mongodb() para estabelecer uma conexão com a base de dados MongoDB.

d. Inserir Dados na Coleção do MongoDB: Os dados são inseridos na coleção MongoDB especificada (collection\_name). É utilizado o método insert\_many() da coleção MongoDB para inserir vários documentos (dicionários) na coleção de uma só vez.

e. Imprimir o Número de Documentos Inseridos: O script imprime o número de documentos (linhas) que foram inseridos com sucesso na coleção.

1. Análise dos Argumentos da Linha de Comandos:

O script utiliza a biblioteca argparse para analisar os argumentos da linha de comandos. A classe argparse.ArgumentParser() é usada para capturar os argumentos.

Foram definidos dois argumentos:

**csv\_file\_path**: O caminho para o arquivo CSV que contem o dataset.

**collection\_name:** O nome da coleção MongoDB onde os dados serão armazenados.

O script, em seguida, utiliza parser.parse\_args() para analisar os argumentos fornecidos na linha de comandos e os atribui às variáveis csv\_file\_path e collection\_name.

**Uso**:

Para utilizar este script, será preciso executá-lo a partir da linha de comandos e fornecer o caminho para o arquivo CSV que contém o dataset e o nome desejado para a coleção MongoDB onde os dados serão armazenados.

Por exemplo: python nome\_do\_script.py caminho/para/dataset.csv best\_cities

O script lê os dados do CSV, conecta-se à base de dados MongoDB, insere os dados na coleção especificada e, por fim, imprime o número de documentos inseridos.

**Etapa 2:** Processamento em tempo real da informação em .avro

Nesta etapa temos uma implementação que utiliza o Kafka Producer para enviar dados de uma coleção MongoDB para um tópico Kafka, utilizando o esquema Avro para serializar os registos.

Funcionalidade do código (“real\_time\_processing.py”):

1. Importação das Bibliotecas:

O código começa por importar as bibliotecas necessárias:

* **KafkaProducer**: Esta biblioteca fornece uma interface para interagir com um produtor do Apache Kafka. permite enviar dados para tópicos no Kafka.
* **fastavro**: Esta biblioteca é utilizada para lidar com dados no formato Avro. Facilita a serialização e desserialização dos dados conforme o esquema Avro.
* **io:** Esta biblioteca é utilizada para lidar com operações de entrada e saída, e, neste caso, é usada para criar um buffer de bytes para a serialização do registo Avro.
* **json e bson.json\_util**: Estas bibliotecas são utilizadas para converter dados JSON para um formato que pode ser processado pelo MongoDB.
* **MongoClient e CursorType** do módulo **pymongo**: Estas bibliotecas são utilizadas para se conectar à base de dados MongoDB e configurar o tipo de cursor.

1. Definição do Esquema Avro:

O código define um esquema Avro para representar os dados de cada cidade, especificando os campos que compõem o registo. Este esquema será utilizado na serialização dos dados.

1. Conectando ao MongoDB e Removendo a Coleção:

A função *drop\_collection\_mongodb(db\_name, collection\_name)* estabelece uma conexão com o MongoDB e remove a coleção especificada, caso ela exista. Isto foi feito para que uma nova coleção possa ser criada posteriormente para armazenar os dados atualizados.

1. Função para Serializar o Registo:

A função *serialize\_city\_data(data)* recebe um dicionário que contém os dados da cidade e serializa-o no formato Avro usando o esquema definido anteriormente. O resultado é uma sequência de bytes que representa o registo Avro.

1. Configuração das Conexões com o MongoDB e Kafka:

A função *setup\_mongodb\_connection()* e a definição do produtor Kafka estabelecem as conexões com o MongoDB e o servidor Kafka, respectivamente.

1. Processamento e Envio dos Eventos:

O código utiliza uma estratégia de polling para capturar eventos (registos) no MongoDB e enviá-los para o tópico do Kafka em tempo real. A função *process\_event(event)* recebe um evento, converte-o para JSON, serializa-o usando o esquema Avro e, finalmente, envia o evento para o tópico Kafka.

1. Processamento de Todos os Eventos e Envio para o Kafka:

A função *process\_all\_events()* recupera todos os eventos existentes no MongoDB e envia-os para o tópico Kafka, garantindo que todos os dados sejam processados antes de encerrar o script.

1. Fecho do Produtor Kafka:

O produtor Kafka é fechado após o processamento de todos os eventos.

Também nesta etapa, foi criado um script com o intuito de consumir eventos em tempo real de um tópico do Kafka, onde são processados e armazenados numa base de dados MongoDB. Além disso também realiza cálculos de sustentabilidade média para continentes e países, e utiliza um modelo de machine learning pré-treinado para identificar os indicadores com maior impacto positivo e negativo na pontuação de sustentabilidade.

Funcionalidade do código (“event\_consumption.py”):

1. Importação das Bibliotecas:

O código começa por importar as bibliotecas necessárias, sendo estas Avro, JSON, MongoDB, Pandas e Joblib (para carregar o modelo de machine learning).

1. Definição do Esquema Avro:

O código define um esquema Avro para representar os dados de cada cidade, especificando os campos que compõem o registo. Este esquema será usado para desserializar os dados recebidos do tópico Kafka.

1. Funções para Manipulação do MongoDB:

O código inclui várias funções para se conectar ao MongoDB, inserir dados em coleções específicas, recuperar dados e limpar coleções existentes. Essas funções são usadas para interagir com a base de dados MongoDB.

1. Função para Desserializar os Dados:

A função *deserialize\_city\_data(data)* recebe uma sequência de bytes representando um registo Avro e desserializa os dados usando o esquema definido anteriormente. Retorna os dados em formato de dicionário Python.

1. Função para Cálculo da Sustentabilidade dos Continentes:

A função *calculate\_continent\_sustainability\_and\_insert\_on\_db(cities\_data)* recebe uma lista de dados de cidades e calcula a média da pontuação de sustentabilidade para cada continente. Em seguida, insere os dados calculados no MongoDB numa coleção específica.

1. Função para Cálculo da Sustentabilidade dos Países:

A *função calculate\_country\_sustainability\_and\_insert\_on\_db(cities\_data)* recebe uma lista de dados de cidades e calcula a média da pontuação de sustentabilidade para cada país. Em seguida, insere os dados calculados no MongoDB numa coleção específica.

1. Função para Previsão de Indicadores com Maior Impacto:

A função *predict\_most\_positive\_feature(data, model\_filename, collumns\_to\_drop)* recebe os dados de cidades, o nome do arquivo do modelo de machine learning pré-treinado e as colunas a serem excluídas do DataFrame. A função utiliza o modelo para fazer previsões e identifica os indicadores (características) com maior impacto positivo e negativo na pontuação de sustentabilidade.

1. Configuração do Consumer Kafka:

O código define as configurações necessárias para se conectar ao tópico do Kafka e consumir eventos em tempo real.

1. Loop para Consumir e Processar os Dados em Tempo Real:

O loop principal consome mensagens do tópico do Kafka usando o consumidor previamente configurado. Cada mensagem é desserializada usando a função *deserialize\_city\_data()* e, em seguida, é inserida numa collection do MongoDB usando a função *insert\_city\_data\_into\_mongodb()*. Em seguida, os dados são recuperados do MongoDB usando a função *get\_data\_from\_mongodb().*

1. Cálculo de Sustentabilidade para Continentes e Países:

Com os dados das cidades, são realizados os cálculos de sustentabilidade média para continentes e países usando as funções *calculate\_continent\_sustainability\_and\_insert\_on\_db()* e *calculate\_country\_sustainability\_and\_insert\_on\_db()* respectivamente. Os resultados são armazenados em coleções específicas no MongoDB.

1. Identificação dos Indicadores com Maior Impacto:

A função *predict\_most\_positive\_feature()* é chamada para identificar os indicadores com maior impacto positivo e negativo na pontuação de sustentabilidade. Os resultados são inseridos na coleção de cidades do MongoDB.

1. Fecho do Consumidor Kafka:

Após o processamento dos dados, o consumidor Kafka é fechado.

**Etapa 3**: Treino do modelo de machine learning para identificação do fator com maior impacto no cálculo global

Foi criado um modelo de regressão linear para prever a pontuação de sustentabilidade ("Overall") das cidades, com base em indicadores disponíveis nos dados armazenados no MongoDB. O modelo treinado é salvo para uso futuro (kafka consumer) e, em seguida, são calculadas algumas métricas de avaliação para medir o desempenho do modelo. Além disso, o código identifica os indicadores (características) que têm o maior impacto positivo e negativo na pontuação de sustentabilidade.

Funcionalidade do código (“train\_model.py”):

1. Importação das Bibliotecas:

O código começa por importar as bibliotecas necessárias, incluindo a biblioteca PyMongo para interagir com o MongoDB, a biblioteca Pandas para manipulação de dados em formato de DataFrame e a biblioteca Joblib para salvar o modelo de regressão linear treinado.

1. Função para Obter os Dados das Cidades do MongoDB:

A função *get\_city\_data\_from\_mongodb()* é utilizada para se conectar à base de dados MongoDB e capturar os dados das cidades armazenados na coleção "best\_cities".

1. Criação do DataFrame:

O código utiliza a função *get\_city\_data\_from\_mongodb()* para obter os dados das cidades e, em seguida, cria um DataFrame do Pandas para armazená-los e facilitar a manipulação.

1. Separação dos Dados para Treino:

Os dados do DataFrame são separados em duas partes: as *features* (indicadores) representadas por X e a variável de *target* (pontuação de sustentabilidade) representada por y.

1. Divisão dos Dados em Conjuntos de Treino e Teste:

O conjunto de dados é dividido em conjuntos de treino e teste usando a função *train\_test\_split()* do módulo sklearn.model\_selection. Permite avaliar o desempenho do modelo em dados não utilizados durante o treino.

1. Escolha e Treino do Modelo de Regressão Linear:

O modelo escolhido para prever a pontuação de sustentabilidade é uma regressão linear, representada pela classe LinearRegression do módulo sklearn.linear\_model. O modelo é treinado usando os dados de treino X\_train e y\_train.

1. Avaliação do Modelo:

São calculadas três métricas de avaliação para o modelo: Mean Absolute Error (MAE), Mean Squared Error (MSE) e R-squared (R2). Estas métricas fornecem uma medida da precisão e qualidade do modelo.

Resultado das métricas: Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

1. Identificação dos Indicadores com Maior Impacto:

O código obtém os coeficientes do modelo treinado através do atributo coef\_. Em seguida, cria-se um dicionário que associa cada coeficiente ao nome da característica correspondente. A partir disso, o código identifica os indicadores com o maior impacto positivo e negativo na pontuação de sustentabilidade.

1. Salvar o Modelo Treinado:

O modelo de regressão linear é salvo num arquivo usando a função *joblib.dump()*. Isso permite que o modelo seja carregado e reutilizado em outros contextos sem a necessidade de voltar a ser treinado.

Anexos

Coleção “Cities” da base de dados “sustainability”

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

Coleção “Continent\_metrics” da base de dados “sustainability”

Uma imagem com texto, captura de ecrã, número, software

Descrição gerada automaticamente

Coleção “country\_metrics” da base de dados “sustainability”

Uma imagem com texto, captura de ecrã, software, número

Descrição gerada automaticamente

Coleção “best\_cities” da base de dados “technical\_challenge\_ubiwhere”

Uma imagem com texto, captura de ecrã, software, número

Descrição gerada automaticamente

Foram criadas duas bases de dados:

**Technical\_challenge\_ubiwhere** – Base de dados que contém a coleção best cities onde estão armazenados os dados provenientes do dataset da Arcadis.

Sustainability – Base de dados que contém as coleções Cities, Continent\_metrics e Country\_metrics. A coleção Cities contém os dados das cidades processados em tempo real e na última linha é possível consultar os fatores que tiveram mais e menos impacto no Overall.

Uma imagem com texto, Tipo de letra, captura de ecrã, branco

Descrição gerada automaticamente

A coleção Continent\_metrics contém registos dos continentes e as suas médias em termos de Overall e restantes pontuações. Na última linha é possível consultar o melhor continente em termos de Ranking.

Uma imagem com texto, captura de ecrã, Tipo de letra, cartão de visita

Descrição gerada automaticamente

A coleção Country\_metrics contém registos dos países e as suas médias em termos de Overall e restantes pontuações.