Universidade Estadual de Londrina

JOÃO GABRIEL RODRIGUES SILVA

PROPOSTA DE UM MÉTODO PARA REPRESENTAÇÃO E ARMAZENAMENTO DE PROVENIÊNCIA EM FEATURE STORES

LONDRINA
2023

JOÃO GABRIEL RODRIGUES SILVA

PROPOSTA DE UM MÉTODO PARA REPRESENTAÇÃO E ARMAZENAMENTO DE PROVENIÊNCIA EM FEATURE STORES

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Daniel dos Santos Kaster

LONDRINA
2023

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Sobrenome, Nome.
Título do Trabalho : Subtitulo do Trabalho / Nome Sobrenome. - Londrina, 2017.
100 f. : il.

Orientador: Nome do Orientador Sobrenome do Orientador.
Coorientador: Nome Coorientador Sobrenome Coorientador.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2017.
Inclui bibliografia.

1. Assunto 1 - Tese. 2. Assunto 2 - Tese. 3. Assunto 3 - Tese. 4. Assunto 4 - Tese. I. Sobrenome do Orientador, Nome do Orientador. II. Sobrenome Coorientador, Nome Coorientador. III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

JOÃO GABRIEL RODRIGUES SILVA

PROPOSTA DE UM MÉTODO PARA REPRESENTAÇÃO E ARMAZENAMENTO DE PROVENIÊNCIA EM FEATURE STORES

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina

Prof. Dr. Segundo Membro da Banca
Universidade/Instituição do Segundo Membro da Banca – Sigla instituição

---

Prof. Dr. Terceiro Membro da Banca
Universidade/Instituição do Terceiro Membro da Banca – Sigla instituição

Londrina, 03 de maio de 2023.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Adriana e João, que sempre me incentivaram a escolher o curso a partir apenas da minha decisão, e que me ajudaram muito emocionalmente nesse último ano. Agradeço também ao meu orientador Daniel Kaster, por ter acompanhado de perto o desenvolvimento deste trabalho e providenciado auxílios em momentos difíceis do desenvolvimento e escrita. Por fim, agradeço aos amigos feitos durante o curso, em especial Gabriel, Lucca e Victor Hugo, aos quais sempre nos ajudamos durante todos esses anos e sei que continuará assim após a universidade.

SILVA, JOÃO GABRIEL RODRIGUES.. **Proposta de um método para representação e armazenamento de proveniência em Feature Stores**. 2023. 60f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

Aprendizado de máquina é uma área da inteligência artificial que busca desenvolver sistemas de aprendizado capazes de adquirir conhecimento de forma independente através de experiências passadas. O preparo de dados é uma etapa essencial para o aprendizado de máquina, sendo muito comum a utilização de diversas técnicas para a criação de características que melhor se adéquem ao problema, sendo importante o registro das técnicas utilizadas no processamento de cada característica. *Feature Store* é uma tecnologia recente que auxilia em diversas etapas no desenvolvimento de aprendizado de máquina, tendo como sua maior utilidade o armazenamento e reúso de características sem a necessidade de refazer as técnicas aplicadas durante suas criações. Para garantir um reúso eficaz e seguro de características é fundamental utilizar do conceito de proveniência, um conceito que consiste em armazenar a origem e as transformações ao longo do tempo de um dado. O objetivo deste trabalho é o desenvolvimento de métodos para obtenção de proveniência das características armazenadas em uma *Feature Store*, tornando possível saber detalhes do processo de formação de cada característica. Este trabalho propoe um diagrama utilizado para a representação de conjunto de dados e suas modificações ao longo do tempo no contexto de aprendizado de máquina. Este trabalho também apresenta um sistema desenvolvido para potencializar a proveniência das características armazenadas na *Feature Store* Hopsworks. Por fim, foi realizado um estudo de caso utilizando o diagrama e sistema desenvolvidos, buscando verificar a capacidade da proveniência disponibilizada pela utilização dessas ferramentas. As contribuições deste trabalho permitem a representação gráfica de informações de proveniência para conjuntos de dados em aprendizado de máquina, assim como maneiras para a obtenção de proveniência de características especificamente em Feature Stores.

Palavras-chave: Aprendizado de máquina. Características. Proveniência. Feature Stores.

SILVA, JOÃO GABRIEL RODRIGUES.. **Proposal for a method for representing and storing provenance in a Feature Store**. 2023. 60p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2023.

ABSTRACT

Machine learning is an area of artificial intelligence that seeks to develop learning systems capable of independently acquiring knowledge through past experiences. Data preparation is an essential step for machine learning, and it is very common to use different techniques to create features that best fit the problem, and it is important to record the techniques used in the processing of each feature. *Feature Store* is a recent technology that assists in several stages in the development of machine learning, having as its greatest utility the storage and reuse of features without the need to redo the techniques applied during their creations. To ensure an effective and safe reuse of features, it is essential to use the concept of provenance, a concept that consists of storing the origin and transformations over time of data. The objective of this work is the development of methods for obtaining the provenance of features stored in a *Feature Store*, making it possible to know details of the formation process of each feature. This work proposes a diagram used to represent a dataset and its modifications over time in the context of machine learning. This work also presents a system developed to leverage the provenance of features stored in *Feature Store* Hopsworks. Finally, a case study was carried out using the diagram and system developed, seeking to verify the provenance capacity made available by using these tools. The contributions of this work allow the graphical representation of provenance information for datasets in machine learning, as well as ways to obtain provenance of features specifically in Feature Stores.

Keywords: Machine Learning. Features. Provenance. Feature Stores.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ML	*Machine Learning*
MLOps	*Machine Learning Operations*
FS	*Feature Store*
API	*Application Programming Interface*
JSON	*JavaScript Object Notation*
REST	*Representational State Transfer*

SUMÁRIO

1 INTRODUÇÃO

Aprendizado de máquina (*Machine Learning* – ML), é uma área da inteligência artificial que busca métodos computacionais capazes de aprender de acordo com experiências passadas[5]. Esse tópico se mostrou extremamente importante para diversas áreas e cresceu muito com o passar do tempo, crescendo também sua complexidade e necessidade de melhores ferramentas e tecnologias[6]

Conforme o uso do aprendizado de máquina se dissemina, surgem mais ferramentas focadas no desenvolvimento e manutenção de sistemas de aprendizado de máquina. O conceito de MLOps[6] engloba um conjunto de práticas e ferramentas para suportar o desenvolvimento, implantação e execução de sistemas baseados em ML. MLOps é inspirado na noção de DevOps, termo usado para englobar o conjunto, as práticas e ferramentas para suportar o ciclo de vida de desenvolvimento, execução e manutenção de sistemas. MLOps visa simplificar o ciclo de vida de sistemas de ML, acelerar o desenvolvimento e mitigar diversos problemas que surgem devido a características específicas dessa área[7].

Um modelo de ML tipicamente recebe um vetor de características e devolve um ou mais valores indicando uma predição. Um vetor de características pode ser visualizado como uma linha de uma tabela, onde cada coluna representa uma característica distinta. Cada característica possui uma relevância específica no resultado final, sendo que modelos tendem a perder desempenho com grandes quantidades de características que pouco contribuem para o resultado[8]. Buscando uma melhor eficiência dos modelos de aprendizado de máquina, é comum a utilização de técnicas para a criação de novas características e seleção das características mais relevantes. Tais técnicas fazem parte de um processo mais amplo, popularmente conhecido como engenharia de características.

Devido à natureza do aprendizado de máquina, grandes quantidades de dados são necessárias, onde muitas vezes esses dados passam por transformações para melhor desempenho do modelo. A falta de cuidado com o registro das transformações utilizadas pode causar problemas de confiabilidade e reprodutibilidade[9]. Para auxiliar nesse aspecto, é comum utilizar-se o conceito de proveniência.

Proveniência, ou linhagem, é um conceito que surgiu inicialmente na comunidade de banco de dados, que consiste basicamente em armazenar a origem de um dado e suas modificações ao longo do tempo[3]. No aprendizado de máquina, proveniência pode ser usada em diferentes etapas do ciclo de vida, sendo mais comum a utilização no preparo de dados e aprendizado de modelos. Devido à importância de proveniência em ML, tem-se buscado a incorporação deste conceito nas ferramentas, seja de maneira manual ou automática.

Feature Store é uma tecnologia recente na comunidade de aprendizado de máquina, que consiste em um repositório centralizado para ML, provendo a capacidade de armazenamento, versionamento e compartilhamento de características[10]. Uma *Feature Store* auxilia em diversas etapas do ciclo de vida de ML, tendo como sua maior vantagem a possibilidade de reutilização de características já processadas, evitando trabalho duplicado durante o preparo de dados. Devido a isso, a utilização de *Feature Stores* vem sendo cada vez mais adotada.

Por ser um conceito recente que é cada vez mais adotado, as capacidades de uma *Feature Store* estão evoluindo rapidamente, porém, não existem muitos artigos que estudam o tema. Proveniência em *Feature Stores*, especificamente, é um tópico muito pouco abordado na literatura, fazendo-se necessário o desenvolvimento de trabalhos que explorem como *Feature Stores* utilizam e se beneficiam de proveniência.

O objetivo deste trabalho é o desenvolvimento de maneiras para a obtenção de proveniências das características armazenadas em uma Feature Store, almejando saber em detalhes a origem e o processo de formação das características armazenadas em uma Feature Store, potencializando as principais vantagens da utilização de *Feature Stores*. Com as contribuições apresentadas por este trabalho, é possível a total compreensão de características armazenadas na *Feature Store* Hopsworks, assim como a visualização ampla de diferentes conjuntos de dados e suas origens.

Este documento está organizado da seguinte maneira. O capítulo apresenta os fundamentos necessários para o entendimento deste trabalho. O capítulo 3 mostra trabalhos na literatura relacionados com este trabalho. O capítulo 4 descreve um diagrama para o gerenciamento de dados de aprendizado de máquina, capaz de descrever a linhagem e as modicações de um conjunto ao longo do tempo. Além disso, o capítulo 4 também apresenta um sistema de obtenção de proveniência de características, designado para ser utilizada em conjunto com a *Feature Store* Hopsworks. O capítulo 5 apresenta um estudo de caso, utilizando o diagrama e sistema desenvolvidos para processamento de diversas características, verificando a proveniência disponível para as características armazenadas na *Feature Store*.

2 FUNDAMENTAÇÃO TEÓRICO-METODOLÓGICA

2.1 Aprendizado de máquina

Aprendizado de máquina é uma área da inteligência artificial que busca desenvolver sistemas de aprendizado capazes de adquirir conhecimento de forma independente através de experiências passadas[5].

Existem diferentes categorias de aprendizado de máquina, sendo a mais comum a de aprendizado supervisionado[5]. Durante o aprendizado supervisionado, um modelo recebe um vetor de características (também chamadas de *features*) como entrada e retorna uma ou mais predições como saída. Após a predição, a saída do modelo é comparada com a saída correta, também fornecida pela entrada, e os parâmetros do modelo são ajustados para próximas predições. Quando o conjunto de valores de saída é discreto, o problema é chamado de classificação, sendo comum buscar pela maior taxa de acerto possível. Quando o conjunto de valores de saída é contínuo, o problema é chamado de regressão, onde busca-se minimizar o erro obtido.

Outra categoria existente é a de aprendizado não supervisionado. Nesse caso, durante o aprendizado, apenas o vetor de características é fornecido, e o modelo tenta agrupar entradas semelhantes em diferentes conjuntos. Essa categoria é normalmente utilizada para descobrir novas observações sobre os dados ou para o agrupamento de dados sem a necessidade de observação humana[5].

Existe ainda a categoria de aprendizado semi-supervisionado. Combinando entradas rotuladas (com uma saída correta definida) e não rotuladas, essa categoria é um meio-termo entre aprendizado supervisionado e não supervisionado, sendo normalmente utilizada em situações de grande quantidade de dados de entrada, porém com poucos dados rotulados[11].

Aprendizado de máquina ganhou muita adesão nos últimos anos, causando uma expansão de diversas sub-áreas de pesquisas. Uma dessas sub-áreas mais famosas é a de redes neurais. Redes neurais consiste em um tipo de aprendizado de máquina inspirado em redes neurais humanas.

Uma rede neural é formada por neurônios, podendo ser *perceptrons* ou *sigmoid neutrons*, que recebem valores de entrada e retornam um valor de saída. *Perceptrons* retornam valores discretos, enquanto que *sigmoid neutrons* retornam valores contínuos. A estrutura de uma rede neural é formada por camadas. A primeira camada é chamada de camada de entrada, que recebe as características como entrada, a última camada é chamada de camada de saída, cujos valores retornados pelos neurônios representam as predições feitas

pelo modelo. Camadas intermediarias são chamadas de camadas ocultas[12].

Uma camada de uma rede neural se conecta com a próxima camada através dos valores retornados pelo seus neurônios. Para cada neurônio de uma camada, o valor de saída desse neurônio é utilizada como entrada para os neurônios da próxima camada, possuindo um peso associado que difere para cada neurônio[12].

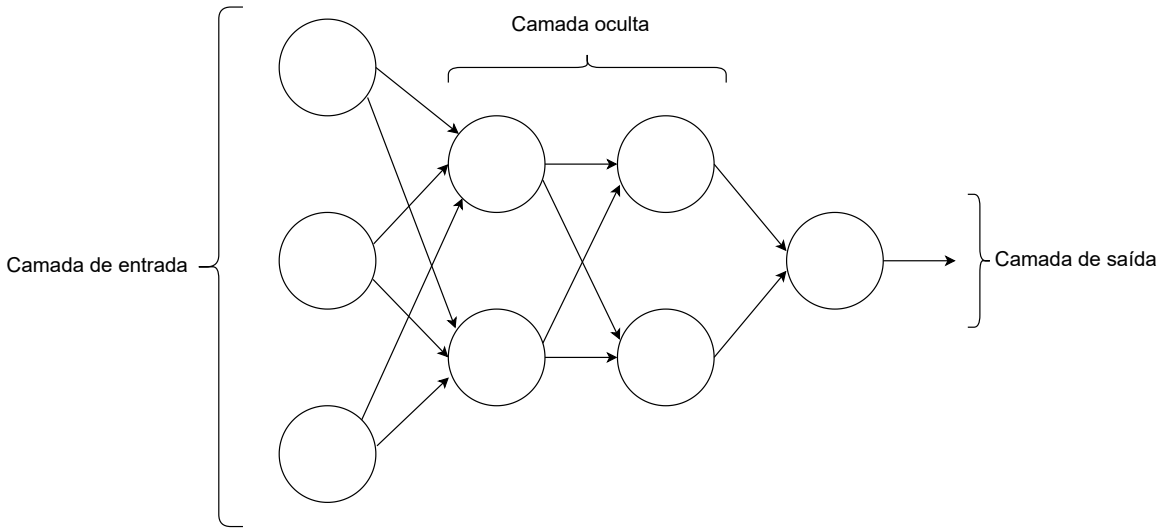

Figura 1 – Exemplo de rede neural. Fonte: Autor

O processo de aprendizado de redes neurais é iterativo. A rede neural consome os dados de treinamento, ajusta o peso dos neurônios e repete o processo, com cada iteração sendo chamada de época[12]. Para o ajuste dos pesos são utilizados algoritmos de otimização, sendo comum o uso do algoritmo Gradient Descent[13].

Atualmente, os modelos mais robustos de aprendizado de máquina são baseados em redes neurais, sendo utilizados em várias aplicações, como reconhecimento de imagens[14], redes adversárias generativas[15], entre outros.

O processo de aprendizado de um modelo é um processo iterativo onde os dados utilizados para o aprendizado são fundamentais. Durante o aprendizado de um modelo, são realizados diversos ajustes com a finalidade de melhorar a precisão de um modelo, sendo comum a utilização de técnicas de engenharia de características e estratégias de gerenciamento das características obtidas.

2.2 Engenharia de Características

O preparo de dados é essencial para o aprendizado de máquina, sendo muitas vezes a etapa mais demorada no desenvolvimento de um sistema de aprendizado de máquina[10]. Um modelo tem sua precisão diretamente relacionada com as características utilizadas em

sua entrada, quanto mais características relevantes para o modelo, mais eficiente o modelo fica[8]. Buscando obter características relevantes para um modelo específico, existem diversas tipos de técnicas que podem ser utilizadas para expandir ou reduzir o total de características, de forma a utilizar os dados disponíveis de maneira mais eficiente.

Construção de características se refere ao processo de obter novas características relacionando características já existentes. Construção de características aumenta o poder de expressão das características originais e é normalmente utilizada para expandir a dimensionalidade das características[16].

Assim como construção de características expande a dimensionalidade do conjunto de dados, exitem técnicas utilizadas para reduzir a dimensionalidade. Extração de características é uma técnica que agrupa características existentes, criando novas características enquanto tenta manter os dados existentes nas características antigas[17]. *Principal Component Analysis* e *Linear Discriminant Analysis* são exemplos clássicos de métodos de extração de características[1].

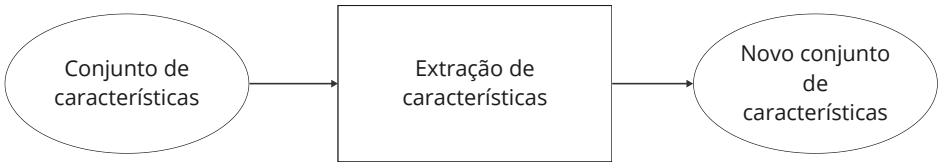

Figura 2 – Processo de extração de características. Fonte: [1]

Seleção de características é outra técnica utilizada para redução de dimensionalidade. Diferente da extração, a seleção não cria nenhuma nova característica, buscando apenas selecionar as características mais relevantes de acordo com um determinado critério. Exitem 3 tipos de seleções de características: supervisionada, não supervisionada e semi-supervisionada[18].

Existem ainda transformações que apenas modificam as características, sem aumentar ou reduzir a dimensionalidade do conjunto. Tais transformações são muito utilizadas no preparo de dados para aprendizado de máquina, seja para enriquecer os dados ou para adequá-los ao algoritmo de aprendizado de máquina utilizado.

Diversos algoritmos lidam apenas com valores numéricos, sendo necessário um tratamento de características categóricas. Nesses casos, é comum a utilização de transformações de codificação. A transformação mais básica de codificação é chamada de *Label Encoding*, que atribui um valor numérico distinto para cada valor categórico[19].

Ainda sobre transformações para características categóricas, existe também a codificação *One-Hot Encoding*[19]. Essa transformação se difere de transformações tradicionais, sendo uma exceção que aumenta a dimensionalidade do conjunto. Para cada valor distinto presente na característica categórica original, é criada uma característica booleana

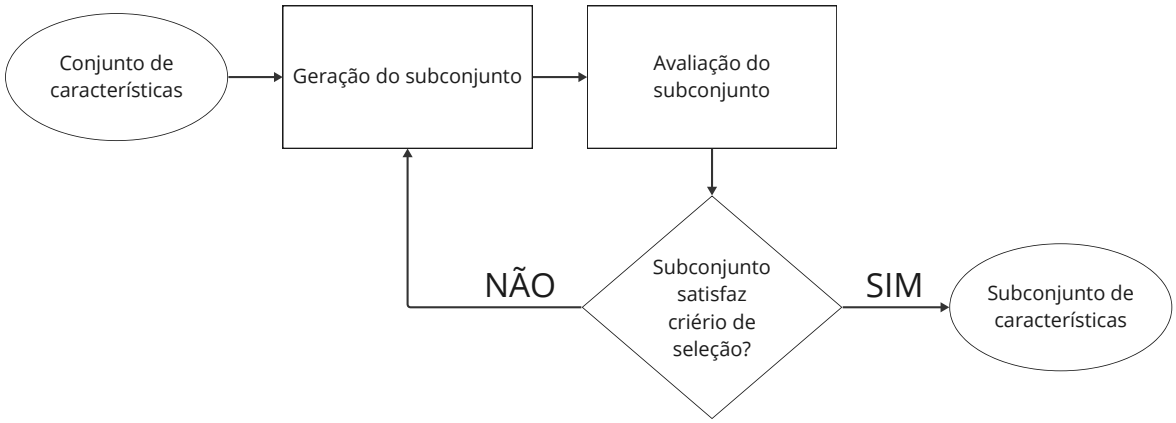

Figura 3 – Processo de seleção de características. Fonte: [1]

que representa aquele valor. A característica criada pode possuir o valor de um, indicando que o valor da característica categórica equivale ao valor correspondente da característica numérica, ou zero, indicando que não equivale ao valor correspondente. Em um vetor de características, entre todas as características numéricas criadas pela codificação *One-Hot Encoding*, apenas uma possui valor igual a um, com todas as outras possuindo valor zero.

Característica categórica

	Cor
1	Vermelho
2	Verde
3	Azul

One-Hot Encoding

	Vermelho	Verde	Azul
1	1	0	0
2	0	1	0
3	0	0	1

Figura 4 – Exemplo de *One-Hot Encoding*. Fonte: Autor

Alguns algorítimos de aprendizado de máquina são sensíveis a escala dos valores, podendo ter uma perda de desempenho quando existem valores muito discrepantes. Para isso, existem transformações para características numéricas que buscam trazer os valores todos para uma mesma escala, como Normalização e Padronização[19].

Apesar de engenharia de características ser um processo manual, existem estudos que exploram técnicas para automatização do processo[20]. Seleção de características, em específico, é uma técnica comum de ser utilizada de forma automática, utilizando algorítimos de otimização como algorítimos genéticos[21][22].

No preparo de dados, é comum a combinação de várias técnicas para a criação de

características adequadas. Durante essa etapa, é importante o armazenamento das características obtidas e a documentação das técnicas utilizadas, de forma a garantir um reúso seguro dessas características sem a necessidade de refazer todo o processo de preparação de dados e engenharia de características, o que frequentemente acontece quando não há a adoção de ferramentas adequadas para o gerenciamento de bases de características.

2.3 Feature Stores

Feature Store (FS) é um conceito recente na área de aprendizado de máquina, ganhando destaque após sua utilização na plataforma de aprendizado de máquina da Uber[10]. Desde então, a utilização de *Feature Stores* vêm crescendo rapidamente, com grandes empresas como Amazon, Facebook, Google e diversas outras incorporando suas próprias *Feature Stores* em suas plataformas de aprendizado de máquina, além do desenvolvimento de *Feature Stores* de código aberto como Feast[1] e Hopsworks[2].

Uma *Feature Store* é um repositório centralizado de dados curados para aprendizado de máquina, possuindo três principais componentes[10]:

a) armazenamento: pode armazenar características de diversas maneiras, como banco de dados SQL, NoSQL, armazenamento em nuvem, sistemas distribuídos, entre outros;

b) interface: possui uma interface utilizada para acesso das características, definindo como deve ser feita a leitura e escrita de novas características na FS;

c) registro de metadados: armazena todas as informações importantes relacionadas à características, como versões e origem, possibilitando a pesquisa e compartilhamento de características por parte de usuários.

As figuras 5 e 6 mostram uma das vantagens da utilização de *Feature Stores*. Na figura 5, características de diferentes fontes são utilizadas em diferentes modelos, sendo necessário o processamento dessas características para cada modelo que as utiliza. Na figura 6, as características são processadas apenas uma vez, em seguida são armazenadas na *Feature Store*, ficando disponíveis para serem utilizadas por qualquer modelo, sem a necessidade de processá-las novamente.

Uma *Feature Store* normalmente é divida em duas partes: online e offline. A parte offline armazena grandes quantidades de dados, possuindo uma alta taxa de transferência e baixo custo, sendo utilizada para definir conjunto de dados de treinamento e teste que são utilizados por modelos durante seu aprendizado. A parte online armazena apenas os valores mais recentes de uma característica, possuindo um custo maior e latência menor quando comparado a parte offline. Essa parte é normalmente utilizada para modelos em

[1] https://feast.dev/
[2] https://www.hopsworks.ai/

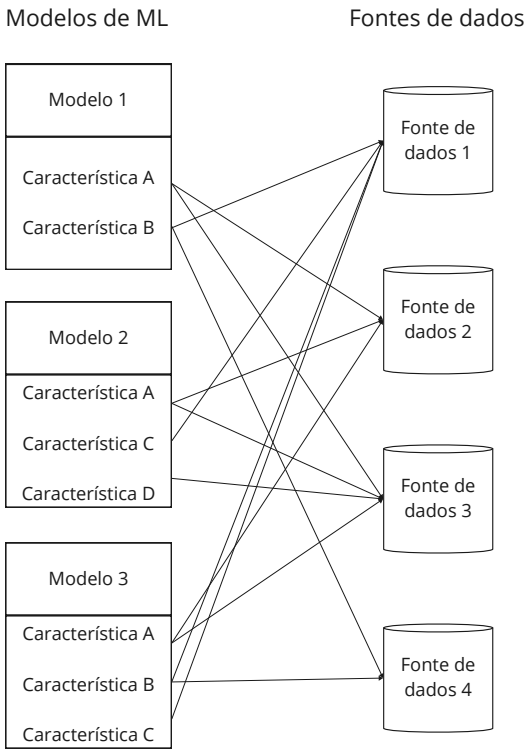

Figura 5 – Aprendizado de máquina tradicional. Fonte: [2]

inferência, com o objetivo de buscar características que possam enriquecer as informações disponíveis. Por exemplo, em um sistema de comércio eletrônico, durante uma pesquisa por parte de um usuário, é possível enriquecer as informações originais (texto digitado) buscando características relacionadas aquele usuário, por exemplo histórico de compras.

Feature Stores também auxiliam em diversas etapas do desenvolvimento de sistemas de aprendizado de máquina, disponibilizando características para o aprendizado de modelos ou predição de modelos em inferência e fornecendo métricas de qualidade que auxiliam na detecção de erros e análise de modelos e características[2].

Feature Stores aceleram uma das etapas mais trabalhosas no desenvolvimento de aprendizado de máquina, o preparo de dados. Em uma *Feature Store*, características podem ser reutilizadas sem nenhum trabalho necessário para processá-las novamente, e também é possível procurar características relacionadas a alguma característica específica. Para potencializar essa vantagem, é necessário um registro preciso de como as características foram geradas, sendo importante a utilização de proveniência para garantir a reutilização de características de maneira segura.

2.3.1 Feast

Feast é uma *Feature Store* simples, consistindo em um pacote da linguagem Python. Feast não armazena nenhum dado por si, servindo como uma camada de abstração entre

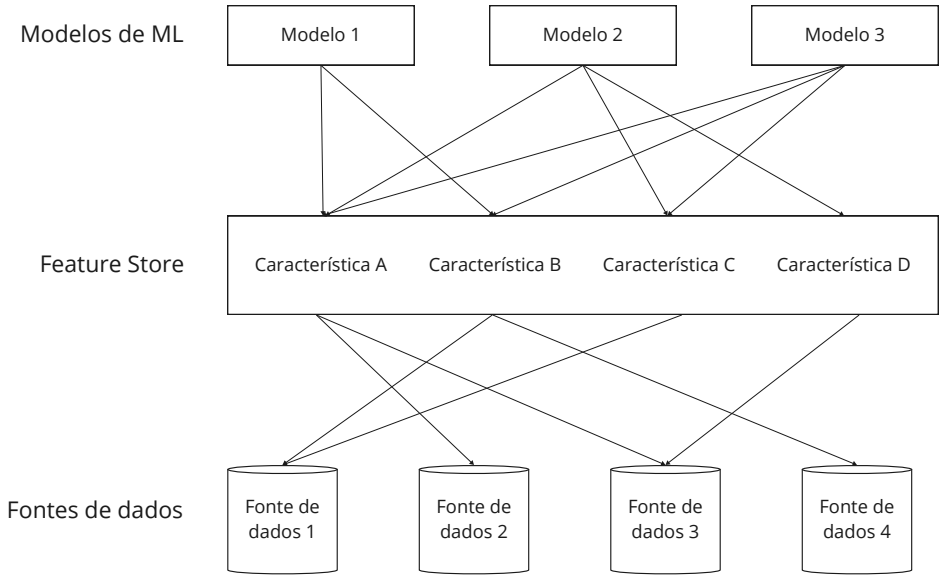

Figura 6 – Aprendizado de máquina com o uso de *Feature Store*. Fonte: [2]

o armazenamento e acesso de características. A localização de armazenamento dos dados deve ser indicada, tendo suporte para diversos tipo como arquivos locais, *Data Lakes* e *Data Warehouses*.

O conceito mais básico do Feast é o de Entidade[23], representando um conjunto de características relacionadas entre si. Entidades por si só não definem nenhuma característica, consistindo apenas de uma chave e um selo de tempo, sendo utilizadas em conjunto com o próximo conceito a ser apresentado.

O próximo conceito é o de *Feature view*[23]. Um *feature view* consiste em um grupo de características com zero ou mais entidades associadas, juntamente com uma fonte de dados. Um *feature view* necessita da localização da fonte de dados e um nome único. Apesar de recomendado, não é necessário definir um esquema para o *feature view*. Nesse caso, o esquema é inferido a partir da fonte de dados. Um detalhe importante sobre *feature views* é que todos os vetores de características presentes na fonte de dados do *feature view* precisam conter um campo com um selo de tempo e campos para as chaves das entidades do *feature view*, que são utilizados para a leitura de valores.

Outro ponto importante do Feast é o registro[23]. O registro é onde são armazenadas todas as informações sobre entidades e *feature views*, sendo utilizado para saber quais características fazem parte de uma determinada *feature view* e onde essa *feature view* está armazenada. Toda vez que ocorre uma mudança no esquema da *Feature Store*, seja ela inserção ou atualização de novas entidades e *feature views*, o registro é atualizado.

As figuras 7 e 8 apresentam exemplos de código para definição de entidade e *feature views*, assim como o acesso de valores para determinadas características de um

feature view.

```
from datetime import timedelta
from feast import (
    Entity,
    FeatureView,
    Field,
    FileSource,
)
from feast.types import Float32, Float64, Int64

# Declaração da entidade
entidade = Entity(name="exemplo_entidade", join_keys=
["clientid"],description="descrição")

# Declaração da fonte de dados
f_source1 = FileSource(
    name="fonte_dados_1",
    path="/caminho/do/arquivo",
    timestamp_field="event_timestamp" # Campo com selo de tempo
)

# Cricação do feature view
example_fv= FeatureView(
    name="examplo_fv",
    ttl=timedelta(days=1),
    entities=[entidade],
    schema=[
        Field(name="feature1", dtype=Float64, tags={"description":"descrição1"}),
        Field(name="feature2", dtype=Float64, tags={"description":"descrição2"}),
        Field(name="feature3", dtype=Float64, tags={"description":"descrição3"}),
        ],
    source=f_source1
)
```

Figura 7 – Exemplo de código para definição de entidades e *feature views*. Fonte: Autor

```
import pandas as pd
from datetime import datetime
from feast import FeatureStore

store = FeatureStore(repo_path="feature_repo/")

# Valores das entidades
valores_entidade = [
    {"clientid":1,"event_timestamp":datetime(2017, 5, 6, 12, 59, 21, 941097)},
    {"clientid":2,"event_timestamp":datetime(2017, 5, 7, 12, 59, 21, 941097)},
    {"clientid":3,"event_timestamp":datetime(2017, 5, 8, 12, 59, 21, 941097)}
]
entity_df = pd.DataFrame(valores_entidade)

# Leitura das características da feature view.
caracteristicas = store.get_historical_features(
    entity_df=entity_df,
    features=[
        "exemplo_fv:feature1",
        "exemplo_fv:feature2",
        "exemplo_fv:feature3",
        "exemplo_fv:feature4"
    ]
)
dataset = caracteristicas.to_df()
```

Figura 8 – Exemplo de código para leitura *feature views*. Fonte: Autor

2.3.2 Hopsworks

Hopsworks é uma *Feature Store* de código aberto desenvolvido para uso geral pela Logical Clocks. Diferente da Feast, Hopsworks possui um sistema de armazenamento próprio. Para a *Feature Store* offline, é utilizado um sistema de arquivos HopsFS, um sistema baseado em HDFS desenvolvido pela própria Logical Clocks, sendo possível também de-

finir fontes de dados externas para a parte offline. A *Feature Store* online utiliza RonDB para o acesso rápido dos dados.

O conceito inicial da Hopsworks é o de *Feature Group*[24]. Um *feature group* é similar a uma tabela em um banco de dados, onde as colunas dessa tabela representam características. Um *feature group* necessita de um nome único e um campo que indica qual ou quais características compõem a chave primária da tabela.

Existe também o conceito de *Feature View*[24]. Assim como um *feature group* se assemelha a uma tabela de banco de dados, um *feature view* se assemelha a uma visão. Um *feature view* consiste em um conjunto de características que podem vir de diferentes *feature groups*. Assim como em uma visões tradicionais em bancos de dados, *feature views* não armazenam dados, realizando uma consulta para acessar as características toda vez que o *feature view* é acessado. Também é possível definir transformações para cada característica presente na *feature view*, como padronização, codificações, entre outros.

Existe ainda o conceito de conjunto de dados de treinamento[24], que está diretamente atrelado ao conceito de *feature views*. Um conjunto de dados de treinamento é criado a partir de um *feature view*, utilizando todas suas características e transformações associadas. conjunto de dados de treinamento possuem dois subconjuntos, treinamento e teste.

As figuras 9 e 10 apresentam exemplos de código para leitura e escrita de *feature groups*, *feature views* e conjunto de dados de treinamento.

```python
import hopsworks

# Realizar login na feature store
project = hopsworks.login()
fs = project.get_feature_store()

# Criação do feature group
example_fg = fs.create_feature_group(
    name="example_fg",
    version=1,
    description="Descrição",
    primary_key=["chave-primaria"]
)

# Insere os dados no feature group
example_fg.insert(dados)

# Obtém feature group da feature store
example_fg = fs.get_feature_group(name="example_fg", version=1)

# Ler valores de todas características armazenadas
dados = example_fg.read()

# Selecionar características específicas de um feature group
query = example_fg.select(["feature1", "feature2"])
query.read()
```

Figura 9 – Escrita e leitura de *feature groups*. Fonte: Autor

```python
import hopsworks

# Realizar login na feature store
project = hopsworks.login()
fs = project.get_feature_store()
example_fg = fs.get_feature_group(name="example_fg", version=1)

# Definição da consulta utilizada pela feature view
query = example_fg.select(["feature1", "feature2"])

# Definição de funções de transformações para as características da feature view
min_max_scaler = fs.get_transformation_function(name="min_max_scaler")
label_encoder = fs.get_trasnformation_function(name="label_encoder")
transformations = {
    "feature1": min_max_scaler,
    "feature2": label_encoder
}

# Criação do feature view
example_fv = fs.create_feature_view(
    name="example_fv",
    version=1,
    query=query,
    transformation_function = transformations
)

# Leitura de uma feature view
example_fv = fs.get_feature_view(name="example_fv", version=1)

# Criação de conjunto de dados de treinamento
td_version, td_job = example_fv.create_train_validation_test_split(
    validation_size = 0.2,
    test_size = 0.1
)

# Leitura de um conjunto de dados de treinamento
x_train, x_val, x_test, y_train, y_val, y_test =
example_fv.get_train_validation_test_split(version=1)
```

Figura 10 – Escrita e leitura de *feature views*. Fonte: Autor

2.4 Proveniência

O termo proveniência, ou linhagem, descreve, de maneira geral, de onde um dado se originou e quais mudanças foram feitas ao longo do tempo[3], sendo uma maneira de descrever o processo de produção de um dado do começo ao fim[25].

Um exemplo simples de proveniência pode ser visto na figura 11, que mostra um grafo de transformações[3]. Nesse grafo, transformações T_i são aplicadas em conjuntos de dados de entrada I_j resultando em conjuntos de dados de saída O. A partir do grafo, é possível formular duas questões que podem ser respondidas pela proveniência:

a) dada uma saída O_k, de quais entradas I o conjunto de dados O se originou;

b) dada uma saída O_k, quais transformações T geraram o conjunto de dados O.

O conceito de proveniência surgiu como assunto de pesquisas na comunidade de banco de dados. Inicialmente, proveniência era utilizada para explicar o resultado de consultas realizadas a um banco de dados, com uma granularidade tipicamente no nível de tuplas. Nesse contexto, existiam 3 tipos de proveniência: *Why*, *How*, *Where*[26].

A proveniência *Why* determina para cada elemento no resultado de uma consulta, quais tuplas foram responsáveis por inserir aquele elemento no resultado[26]. A proveniência *How* explica como as tuplas apontadas pela proveniência *Why* contribuíram para o

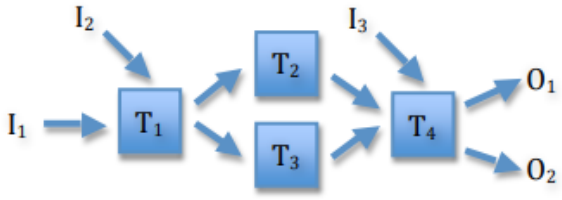

Figura 11 – Grafo de transformação. Fonte: [3]

resultado final[26]. A proveniência *Where* determina a localização no banco de dados das tuplas apontadas pela proveniência *Why*[26].

Enquanto que as proveniências *Why*, *How*, *Where* explicam os resultado existentes, existem proveniências com o objetivo de explicar resultados faltantes, isto é, explicar porque algumas tuplas que poderiam ser esperadas no resultado final não estão presentes. Existem três tipos de proveniência desse tipo: *Instance-based explanations*, *Query-based explanations* e *Refinement-based explanations*[25].

Instance-based explanations busca determinar quais tuplas foram responsáveis pela exclusão de uma certa tupla do resultado final, enquanto que *Query-based explanations* determina quais operadores da consulta foram responsáveis pela exclusão. *Refinement-based explanations* busca uma maneira de reformular a consulta, de maneira que os valores faltantes sejam incluídos no resultado da nova consulta[25].

Apesar de surgir na área de banco de dados, proveniência se mostrou útil em outras áreas, o que levou a uma generalização do conceito e utilização em diferentes áreas como segurança e aprendizado de máquina[27].

Oferecer recursos que permitam a captura de proveniência potencializa a confiança em utilizar-se um conjunto de dados coletado e transformado anteriormente em novas tarefas, além de ser fundamental para a interpretação dos dados e de resultados de operações sobre esses dados.

2.5 Proveniência em aprendizado de máquina

Um campo de pesquisa que se beneficiou com o conceito de proveniência é o aprendizado de máquina[27]. Ao incluir informações intrínsecas do aprendizado de máquina, como características e hiper-parâmetros de modelos, a proveniência auxilia no entendimento de resultados, reprodução de modelos e análise de qualidade de dados[27][9][25], demonstrando ser importante para diferentes etapas do ciclo de vida de aprendizado de máquina[28].

Proveniência é normalmente alcançada por meio de duas maneiras. A primeira re-

quer que o usuário especifique explicitamente quando deseja capturar proveniência, possibilitando um uso mais flexível. Essa maneira é normalmente alcançada por meio inserções no código-fonte[28]. A segunda maneira disponibiliza uma proveniência mais genérica, com a vantagem de poder ser feita de maneira automática sem alteração de código-fonte[29].

Durante o preparo dos dados para consumo por um modelo, é comum uma ampla utilização de diferentes técnicas de engenharia de características, incluindo junção de dados de diferentes conjuntos de dados, criação de novas características e transformações em características existentes. A proveniência de características[9] é responsável por manter um registro das técnicas utilizadas para o processamento de uma característica e qualquer atualização que venha a ser feita posteriormente.

Proveniência de modelos[9] é, junto com a proveniência de características, um dos principais tipos de proveniência para o aprendizado de máquina. Ela é a responsável por capturar informações durante o processo de aprendizado de um modelo, como as características e hiper-parâmetros utilizados para o aprendizado, sendo essencial para a reprodução de modelos.

Proveniência é um conceito muito importante para o aprendizado de máquina, com diversas tecnologias fornecendo mecanismos para a utilização deste conceito. *Feature Stores* especificamente beneficiam-se de proveniência, pois além de necessária para auxiliar na reprodução de modelos, proveniência é fundamental para garantir o reúso das características armazenadas em uma *Feature Store*. Entretanto a adoção de proveniência em MLOps ainda é um assunto incipiente, demandando trabalhos que explorem de forma ampla proveniência neste contexto, particularmente em *Feature Stores*.

3 TRABALHOS CORRELATOS

A seguir são apresentados trabalhos relacionados ao tema desse trabalho. Devido a escassez de artigos que abordam proveniência em *Feature Stores*, foi necessário expandir o escopo dos trabalhos selecionados. Os trabalhos selecionados foram divididos nas seguintes categorias:

a) *Feature Stores*: artigos que explicam o conceito e aspectos de *Feature Stores* ou utilizam *Feature Stores* como uma ferramenta;

b) Proveniência: artigos relacionados a proveniência, tanto em contexto geral, quanto no contexto específico de aprendizado de máquina;

c) Captura de proveniência em processamento de dados: artigos que apresentam especificamente sistemas com o objetivo de capturar proveniência no processamento de dados.

3.1 *Feature Stores*

Feature Store é um conceito recente, tendo poucos estudos que exploram aspectos específicos do funcionamento dessas tecnologias. O artigo de Patel[10] apresenta um ciclo de desenvolvimento de aplicações de aprendizado de máquina e demonstra como o preparo de dados é uma etapa demorada e ineficiente, muitas vezes gerando trabalho repetido devido a falta de um recurso que permita armazenar e compartilhar seguramente características já processadas. O artigo apresenta duas soluções para esse problema, sendo uma delas a utilização de *Feature Stores*.

Patel explica os principais componentes de uma *Feature Store* e exemplifica como *Feature Stores* resolvem o problema de trabalho repetido. Ao final, Patel levanta pontos positivos e negativos das duas soluções abordadas, apontando como positivo para *Feature Stores* a centralização da disponibilização de características, a facilidade de manter metadados e estatísticas e a facilidade no reúso de características. Como ponto negativo, Patel aponta o alto custo de manutenção devido à complexidade das *Feature Stores*.

O artigo de Orr et al.[2] também explica o conceito de *Feature Stores* e demonstra como *Feature Stores* auxiliam em diversas etapas de um *pipeline* de aprendizado de máquina, como gerenciamento de características, treinamento e inferência de modelos e depuração de erros. Também é apresentado uma nova técnica de aprendizado de modelos que vem ganhando adoção, utilizando *embeddings* pré-treinados como características de entrada em um modelo. O artigo aponta como *Feature Stores* atuais, que são desenvolvidas com o intuito de armazenar características como colunas de uma tabela, não são capazes

de lidar com esse método de aprendizado. Por fim, o artigo apresenta possíveis direções futuras para o desenvolvimento de *Feature Stores*, sendo uma delas o desenvolvimento de *Feature Stores* com suporte para *embeddings*.

O artigo de Kakantousis et al.[30] apresenta um pipeline de aprendizado de máquina fim a fim, onde a *Feature Store* Hopsworks é utilizada para simplificar a comunicação entre engenheiros de dados responsáveis por definir características e cientistas de dados responsáveis por utilizar as características para o aprendizado de modelos. O artigo de Cerar et al.[31] apesar de não tratar de *Feature Stores* como foco principal, apresenta diferentes *Feature Stores* de código aberto disponíveis atualmente e seus aspectos técnicos como opções de armazenamento, ingestão de dados e *deploy*.

Apesar de todos os artigos acima abordarem *Feature Stores*, nenhum explora o foco deste trabalho: a utilização de proveniência em *Feature Stores*.

3.2 Proveniência

Proveniência é um conceito bem explorado na literatura, com diversos artigos abordando o tema. O artigo de Herschel et al.[25] explora proveniência em um contexto geral. O artigo detalha os principais usos de proveniência, entre eles, a compreensão dos dados, a reprodutibilidade dos dados e a análise de qualidade dos dados. O artigo também apresenta quatro tipos de proveniência: proveniência de dados, proveniência de fluxo de trabalho, proveniência de sistemas de informação e proveniência de metadados, explicando com mais detalhes os dois primeiros. Este artigo, porém, não entra em detalhes sobre a proveniência no contexto de aprendizado de máquina.

O artigo de Souza et al.[28] explora proveniência em um contexto de aprendizado de máquina. O artigo inicia com uma visão geral sobre ciclo de desenvolvimento de aplicações de aprendizado de máquina. São definidas as etapas do ciclo, iniciando com a curadoria de dados, onde dados brutos são tratados para ficarem adequados para serem utilizados por modelos de aprendizado de máquina. A seguir vem a preparação de dados para aprendizado, que utiliza os dados tratados para montar diferentes conjuntos de dados de treinamento. A última etapa consiste no aprendizado, onde os conjuntos de dados de treinamento criados na etapa anterior são utilizados com diversos algoritmos e hiper-parâmetros, a fim de otimizar a precisão do modelo final.

Também é definido um sistema de captura de proveniência, com foco na captura de proveniências para os dados brutos, modelos gerados, conjuntos de treinamento utilizados e dados de execução, como tempo de execução e métricas de performance. O artigo apresenta uma esquema de representação de dados da proveniência chamado de PROV-ML, baseado nos esquema W3C PROV[32] e ML-Schema[33]. Por fim, o artigo apresenta uma infraestrutura para a captura dos dados de proveniência definidos. Apesar de apresentar

um sistema de captura de proveniência, o artigo não aborda a utilização de *Feature Stores* como opção de armazenamento de características.

O artigo de Ormenisan et al.[29] cita brevemente uma maneira tipicamente utilizada para capturar proveniência em aprendizado de máquina, onde é necessário o usuário explicitar quando é desejado obter proveniência, apelidada de proveniência explícita. O artigo também apresenta uma maneira automática de captura de proveniência, sem a necessidade de interação por parte do usuário, chamada de proveniência implícita, utilizada no sistema de arquivos da *Feature Store* Hopsworks. Esta proveniência porém não captura informações relevantes para o usuário no que tange a compreensão do processamento de características, sendo melhor utilizada para detectar dependências entre arquivos e governança de arquivos.

O único trabalho encontrado que aborda proveniência em *Feature Stores* é o artigo de Ormenisan et al.[34]. Ese artigo fala sobre proveniência na *Feature Store* Hopsworks, como ela é utilizada para reproduzir a execução de *pipelines* e também ajudar na depuração de erros que podem acontecer durante a execução de um *pipeline*. Apesar de abordar o tema proveniência especificamente em *Feature Stores*, o artigo explica somente para uma *Feature Store* específica, e não apresenta nenhum caso de uso real. Alem disso, a proveniência descrita não faz menção a captura de informações sobre o processamento das características armazenadas. Diferentemente desse artigo, a presente proposta de TCC tem como foco uma análise mais aprofundada sobre proveniência em diferentes *Feature Stores* e também a utilização prática de *Feature Stores* de código aberto adotando estratégias de proveniência para as características armazenadas.

3.3 Captura de proveniência em processamento de dados

Os artigos anteriores apresentavam uma proveniência em aprendizado de máquina mais voltada para modelos, buscando explicar o processo de aprendizado de um modelo através de quais conjunto de dados foram utilizados. Apesar de importante, esse tipo de proveniência não é o mais bem aproveitado quando se trata de *Feature Stores*. Em uma *Feature Store*, é importante saber, além de informações descritivas que ajudam a visualizar as características, como as características armazenadas foram geradas. Tais informações são obtidas durante a etapa de processamento de dados, que é tipicamente realizada através de *scripts*.

O artigo de Chapman et al.[4] apresenta uma definição formal para operadores comuns no processamento de dados para aprendizado de máquina. Chapman classifica tais operadores em três principais categorias:

a) Redução de dados: operadores que diminuem a quantidade de linhas (vetores de características) ou colunas (características). Exemplo: seleção e extração de

características;

b) Aumento de dados: operadores que aumentam a quantidade de linhas ou colunas. Exemplo: construção de características, One-Hot Encoding

c) Transformação de dados: operadores que modificam dados sem alterar a quantidade de linhas e colunas. Exemplo: Normalização.

Chapman utiliza um modelo de proveniência baseado no modelo PROV, como demonstrado na figura 12. Neste modelo, uma entidade representa um elemento do conjunto de dados, podendo ser uma linha ou coluna, enquanto que uma atividade representa qualquer técnica de processamento aplicada sobre o conjunto de dados. Com essas informações definidas, são mantidos relacionamentos para indicar que: uma entidade foi gerada ou invalidada por uma atividade, uma entidade foi utilizada por uma atividade e uma entidade foi derivada a partir de outra entidade.

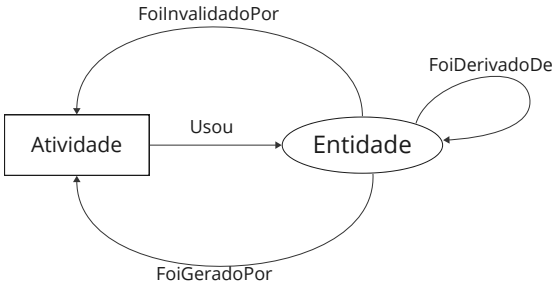

Figura 12 – Modelo de proveniência. Fonte: [4]

Para cada operador, é definido uma função de captura de proveniência, que analisa o conjunto de dados antes e depois da execução do operador e gera arquivos de proveniência de acordo com o modelo para cada elemento afetado, permitindo uma captura de proveniência com uma granularidade em nível de instância. Os arquivos de proveniência podem ser todos utilizados para formar um grafo, disponibilizando uma proveniência com uma granularidade maior.

O artigo de Namaki et al.[35] também apresenta um sistema de captura de proveniência para *scripts* de processamento de dados de aprendizado de máquina, denominado Vamsa. Diferente do artigo anterior, Namaki apresenta uma captura automática de proveniência, porém ligada com um modelo de aprendizado de máquina. A proveniência fornecida em Vamsa é restrita a dizer quais campos de um conjunto de dados originaram as características utilizadas no treinamento de um modelo.

O sistema é divido em 3 partes. A primeira parte é responsável por realizar uma análise estática do *script*, montando um grafo direcionado que descreve o fluxo das operações realizadas. A segunda parte é responsável por adicionar informação semântica ao grafo, como tipos das variáveis presentes no *script*. Para isso, o sistema necessita de uma

base de dados externa, que contém informações sobre as funções das bibliotecas mais comuns, como parâmetros recebidos e tipo de retorno. A última parte utiliza o grafo e a base de dados externa para encontrar as colunas do conjunto de dados que foram utilizadas como características pelo modelo, ou que geraram características utilizadas pelo modelo.

Ambos os trabalhos apresentam o tipo de proveniência na qual este trabalho se concentra, a proveniência de características. Entretanto, os trabalhos não abordam a utilização dessa proveniência em conjunto com *Feature Stores*.

3.4 Recursos de Proveniência em Feature Stores de Código Aberto

A seguir são analisadas funcionalidades disponibilizadas pelas *Feature Stores* Feast e Hopsworks, assim como suas capacidades no quesito proveniência. Ao final, são disponibilizadas as tabelas 1 e 2. A primeira tabela ressalta pontos positivos e negativos das *Feature Stores* quando comparadas entre si, enquanto que a segunda tabela indica a existência ou não de informações que colaboram para a proveniência de características.

3.4.1 Feast

Feast possui uma funcionalidade chamada de *feature server*, onde é possível ler e escrever características fazendo requisições HTTP para uma API, que recebe JSON como entrada e saida. Utilizando *feature server*, não há a necessidade de utilizar uma linguagem específica para acessar a *Feature Store*, bastando apenas uma linguagem que possa realizar requisições HTTP, realizar serializações de dados para JSON e vice-versa, todos requisitos facilmente atendidos por qualquer linguagem utilizada atualmente.

Feast também disponibiliza uma interface web que ajuda na busca e visualização dos *feature views* presentes na *Feature Store*. Para busca, são suportados busca por texto livre, e busca por *tags*. Na visualização de um *feature view* específico, são disponibilizadas informações das características armazenadas naquele *feature view*, como tipo do dado e descrição.

Feast não aparenta fornecer muitos mecanismos que possam ser utilizados para captura de proveniência. Não existe controle de versão em Feast, o que quer dizer que uma mudança no esquema do *feature view* resulta na perda do esquema anterior. Uma alternativa para contornar esse problema, mesmo que de forma precária, poderia consistir na criação de um novo *feature view* em vez de atualizar um *feature view* existente. Essa alternativa porém, é inviabilizada pelo fato de não possuir maneira de relacionar dois *feature views*.

Também não existe nada que possibilite o registro das técnicas de engenharia utilizada para o processamento das características inseridas na *Feature Store*. O melhor que se pode obter de informações de proveniência consiste nas descrições de *feature views*

e de características individuais, sendo longe do ideal para compreender o processo de engenharia das características armazenadas.

3.4.2 Hopsworks

Assim como Feast, Hopsworks também disponibiliza uma interface web que possibilita a busca e visualização de *feature groups* e *feature views*. A visualização é um pouco mais robusta quando comparado com Feast, sendo possível, para cada característica, inspecionar uma amostra dos valores armazenados e visualizar estatísticas como média, quantidade de valores distintos, completude, entre outros. A interface também possui algumas outras funcionalidades úteis, como um histórico de ingestão de dados e a possibilidade de iniciar um servidor Jupyter conectado diretamente à *Feature Store*, possibilitando um rápido acesso à *Feature Store*.

Apesar de possuir requisitos de sistemas mais pesados e instalação mais complexa quando comparada com Feast, Hopsworks possui uma alternativa *Serverless* de fácil acesso, bastando apenas o login com uma conta Google ou GitHub, porém, possuindo alguns limites como armazenamento e quantidade de *Feature Groups*. Embora para casos de grande quantidade de dados seu uso não seja adequado, a versão Serverless possibilita um rápido e fácil acesso para cenários mais simples.

No assunto de proveniência, Hopsworks apresenta alguns mecanismos que colaboram com a captura de proveniência. Ambos *feature groups* e *feature views* possuem controle de versão. Para *feature groups*, mudanças que não alteram o esquema como adição de novos valores são realizadas como *commits* em uma versão específica. Mudanças que alteram o esquema, como inclusão ou remoção de características e mudanças no tipo de uma característica, necessitam que seja criada uma nova versão do *feature group*. De maneira similar, mudanças que alteram o esquema de um *feature view* necessitam de uma nova versão daquele *feature view*.

Hopsworks também apresenta uma maneira de fornecer proveniência para suas entidades (*feature groups*, *feature views* e conjunto de dados de treinamento). Durante a criação de alguma entidade, é possível especificar quais as entidades utilizadas em sua criação. Com isso, é mantido um grafo de proveniência que possibilita descobrir, a partir de determinada entidade, quais entidades foram usadas em sua criação, e quais entidades foram criadas a partir dela. Entretanto, tal funcionalidade só é disponível para clientes *Enterprise* com pelo menos a versão 3 da Hopsworks.

Apesar de possuir estatísticas que ajudam na compreensão dos dados e até mesmo uma proveniência que demonstra as dependências entre diferentes entidades, tais mecanismos não são o suficiente para alcançar o nível de proveniência desejado por este trabalho. Com a proveniência fornecida nativamente por Hopsworks, é possível determinar as origens das características. Entretanto, é necessário além disso, ser capaz de determinar quais

as técnicas utilizadas no processamento das características, o que não é possível apenas com os mecanismos fornecidos pela Hopsworks.

Tabela 1 – Pontos positivos e negativos das Feature Stores

	Pontos positivos	Pontos negativos
Feast	Requisitos de sistema leves Fácil instalação Acesso por requisições HTTP	Não possui armazenamento próprio Não possui controle de versão Sem captura de proveniência nativa Necessidade de dados datados
Hopsworks	Armazenamento próprio Interface web robusta Estatísticas de características Captura de proveniência nativa	Requisitos de sistemas pesados

Tabela 2 – Informações descritivas disponíveis nativamente nas Feature Stores

	Descrição	Estatísticas	Versionamento	Relacionamento entre entidades	Armazenamento de técnicas utilizadas
Feast	Sim	Não	Não	Não	Não
Hopsworks	Sim	Sim	Sim	Sim*	Não

4 DESENVOLVIMENTO DE MÉTODOS PARA OBTENÇÃO DE PROVENIÊNCIA DE CARACTERÍSTICAS

O preparo de dados no aprendizado de máquina é um processo iterativo, onde muitas vezes é desejável salvar, além do estado mais recente, estados intermediários, possibilitando o teste de diferentes técnicas de engenharia de característica para um mesmo objetivo. Um mesmo conjunto de dados pode passar por diversas transformações, gerando diversos conjuntos intermediários cujo registro é útil e desejável. Sem uma estratégia de gerenciamento adequada para tais conjuntos, que possibilite a visualização da linhagem e as técnicas de processamento utilizadas, a compreensão dos dados e de suas origens é dificultada.

A seguir são apresentadas as duas principais contribuições deste trabalho, um diagrama para representação de proveniência de conjuntos de dados no contexto de aprendizado de máquina e um sistema de obtenção de proveniência para características armazenadas em uma *Feature Store*.

4.1 Diagrama de gerenciamento de dados

A seguir é apresentado um diagrama de gerenciamento de características, que busca categorizar conjuntos de dados de acordo com uma série de operações tipicamente utilizadas no preparo de dados. São definidas entidades para a representação dos conjuntos e relacionamentos para a representação das operações, gerando um diagrama que fornece uma visão geral sobre a linhagem de um conjunto de dados, como suas origens e operações aplicadas sobre ele.

4.1.1 Conceitos básicos

Para este diagrama, é presumido que os conjuntos de dados são organizados no formato de uma tabela, onde as colunas representam características e as linhas instâncias de dados. A seguir são explicados as entidades consideradas no diagrama e os relacionamentos que ligam tais entidades.

4.1.1.1 Arquivos de dados brutos

A primeira entidade é chamada de Arquivos de dados brutos, consistindo em arquivo com dados, sem nenhum tipo de processamento para aprendizado de máquina, que são utilizados para gerar características. Como exemplo de arquivos de dados brutos, podem ser citados arquivos CSV e tabelas de bancos de dados.

4.1.1.2 Conjunto de características

Conjunto de características é a segunda entidade do diagrama, representando um grupo de características, relacionadas entre si, extraídas de algum arquivo de dados bruto ou de outro conjunto de características. Conjunto de características possuem três categorias: versão, ramificação e amostra. Tais categorias são utilizadas para melhor definição de diferentes cenários de modificação no conjunto.

Versão é a categoria padrão dos conjuntos de características. Uma nova versão de um conjunto de características representa um aprimoramento ou correção daquele conjunto.

Ramificação é a segunda categoria possível. Ramificações são criadas a partir de modificações em algum conjunto de características que não necessariamente representam uma melhoria ou correção. Ramificações também possuem versionamento, melhorias ou correções feitas em uma ramificação geram uma nova versão daquela ramificação.

Amostra é a última categoria possível, consistindo em um conjunto específico de vetores de características de algum conjunto de características. Amostras não passam por modificações, sendo utilizadas apenas para o armazenamento de seleções de instâncias específicas, como por exemplo conjuntos para treinamento e teste de modelos.

Conjunto de características necessitam de ao menos um campo que indica o propósito do conjunto.Um conjunto de características só pode ser utilizado com outro conjunto de características caso ambos os conjuntos compartilhem ao menos um proposito, possibilitando a junção das instâncias de ambos os conjuntos utilizando o propósito em comum como chave de junção.

4.1.1.3 Relacionamentos

Relacionamentos são utilizados para ligar entidades, representando as operações realizadas sobre conjuntos de dados. São definidos 6 tipos de relacionamentos diferentes, cada um com sua representação gráfica. A seguir são listados os tipos de relacionamento:

a) Seleção total de características

b) Seleção parcial de características

c) Transformação total de características

d) Transformação parcial de características

e) Adição de instâncias de dados

f) Remoção de instâncias de dados

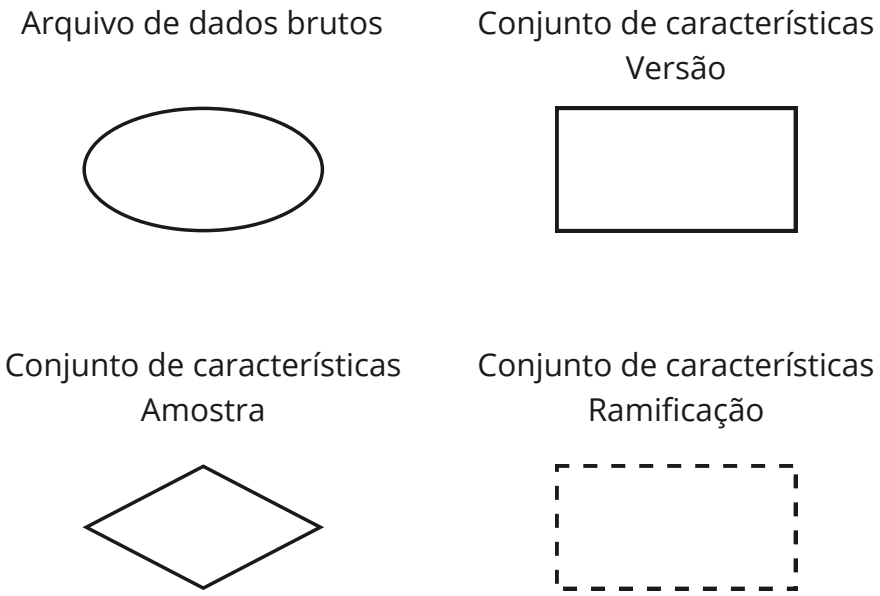

Figura 13 – Representação gráfica das entidades.

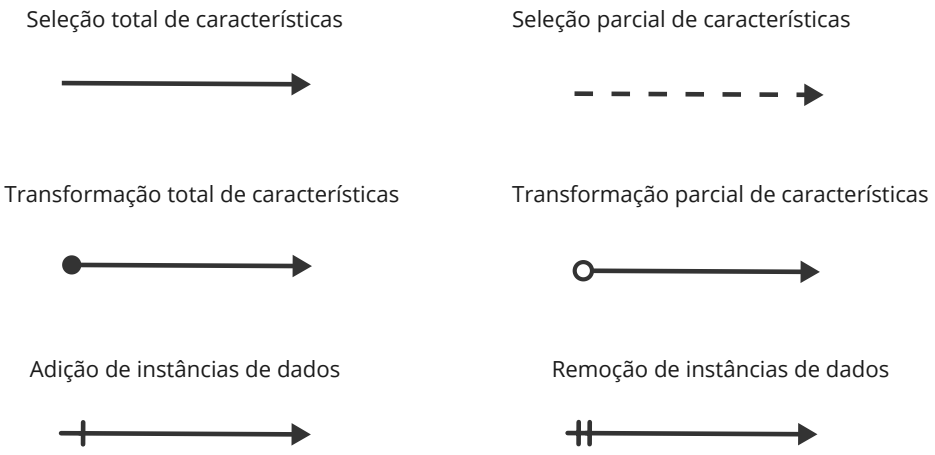

Figura 14 – Representação gráfica dos relacionamentos.

4.1.2 Premissas

A seguir são detalhadas as premissas utilizadas para a definição da entidade resultante após uma operação aplicada sobre um conjunto de dados.

Qualquer operação realizada sobre um arquivo de dados brutos deve obrigatoriamente gerar um novo conjunto de características. Operações sobre um conjunto de

características podem gerar um novo conjunto de características ou uma nova versão, ramificação ou amostra do mesmo conjunto, com cada cenário sendo definido pelas premissas seguintes.

Caso o conjunto resultante não possua todas as características do conjunto original, tal conjunto é considerado um novo conjunto de características. Criação de novas características, mantendo todas as características originais, geram uma nova versão do conjunto.

No cenário de operações envolvendo múltiplos conjuntos, caso nenhum conjunto original possua todas suas características presentes no conjunto resultante, o resultado é considerado um novo conjunto de características, caso contrário é considerado uma nova versão dos conjuntos que mantiveram todas suas características

Adição de novas instâncias e operações de limpeza de dados, como remoção de instâncias com valores nulos ou inconsistentes, são considerados melhorias, sendo assim, o resultado é considerado uma nova versão do conjunto de características. Remoção de instâncias com o proposito de criação de conjunto de dados de treinamento e teste são consideradas amostras.

Por fim, transformações específicas de pré-processamento, como codificação e escalonamento, geram uma ramificação do conjunto de características, visto que tais transformações não necessariamente representam uma melhoria no conjunto, apenas buscam adequar os dados a determinados algoritmos de aprendizado de máquina.

4.1.2.1 Encadeamento de operações

As premissas definem uma ação a ser tomada para cada tipo de operação realizada, definindo qual entidade sera utilizada para representar o conjunto resultante da operação. Entretanto, no preparo de dados, é muito comum a utilização de operações encadeadas, até que se chegue a um estado onde é desejável armazenar os dados resultantes. Para esses casos, a entidade do conjunto resultante é definida a partir das prioridades de cada ação. A seguir são listadas as ações possíveis em ordem de prioridade decrescente:

a) Novo conjunto de características

b) Nova versão do conjunto

c) Nova ramificação do conjunto

d) Nova amostra do conjunto

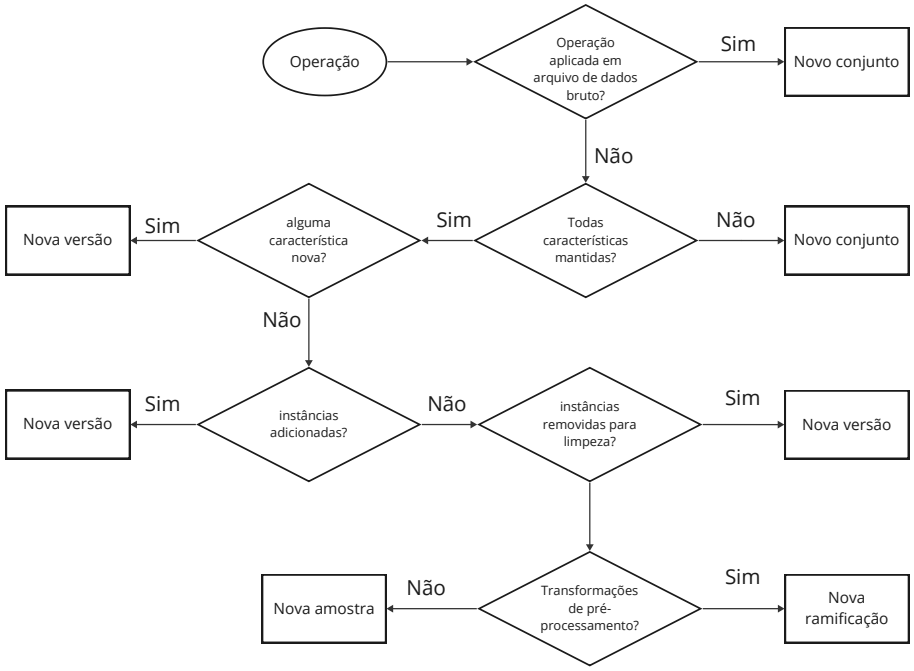

Figura 15 – Fluxograma para definição da entidade.

Relacionamentos podem ser combinados para representar operações encadeadas, a figura 16 mostra exemplos de operações encadeadas e a representação gráfica dos relacionamentos correspondentes.

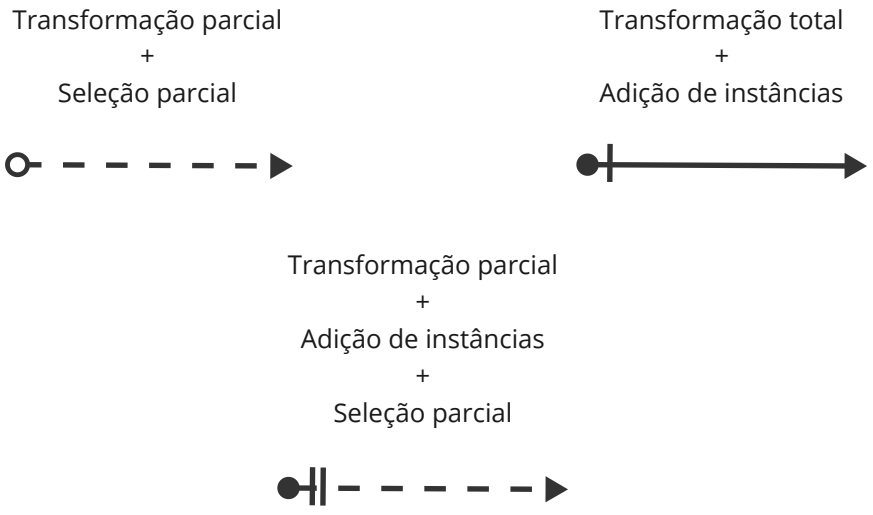

Figura 16 – Exemplos de relacionamentos combinados.

4.1.3 Exemplos

A seguir são apresentados exemplos que demonstram utilização do diagrama em situações cotidianas no preparo de dados de aprendizado de máquina. Em todos os exemplos, assume-se que a característica indicadora do propósito do conjunto não é afetada pelas operações realizadas.

4.1.3.1 Seleção de características

A figura 17 apresenta dois exemplos de operações de seleção de características aplicadas sobre conjuntos de características.

No exemplo a, são selecionadas todas as características do conjunto 'exemplo-1' e parte das características do conjunto 'exemplo-2'. Como o conjunto 'exemplo-1' possui todas suas características no conjunto resultante, o resultado é salvo como uma nova versão do conjunto 'exemplo-1'. Já no exemplo b, é selecionada parte das características do conjunto 'exemplo-1'. Como o conjunto resultante não possui todas as características de 'exemplo-1', o resultado é salvo como um novo conjunto.

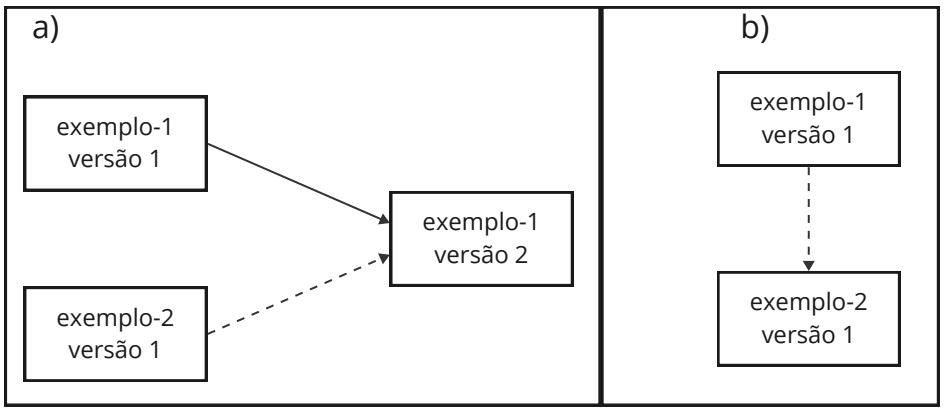

Figura 17 – Exemplos de seleções de características

4.1.3.2 Transformações

A figura 18 apresenta quatro exemplos de transformações aplicadas sobre conjuntos de características.

No exemplo a, é aplicada uma PCA sobre todas as características do conjunto 'exemplo-1', reduzindo sua dimensionalidade. Nesse caso, o resultado é armazenado como um novo conjunto de características.

No exemplo b, são aplicadas transformações de pré-processamento em todas as características do conjunto 'exemplo-1'. Nesse caso, o resultado é armazenado como uma ramificação do conjunto 'exemplo-1'.

Ambos os exemplos c e d aplicam uma transformação parcial para a criação de uma nova característica, por exemplo, a utilização das características 'latitude' e 'longitude' para a computação da característica 'coordenada'. A diferença entre os exemplos se da pelo fato do exemplo c manter as características 'latitude' e 'longitude', resultando em uma nova versão do conjunto, enquanto que o exemplo d descarta as características 'latitude' e 'longitude', resultando em um novo conjunto.

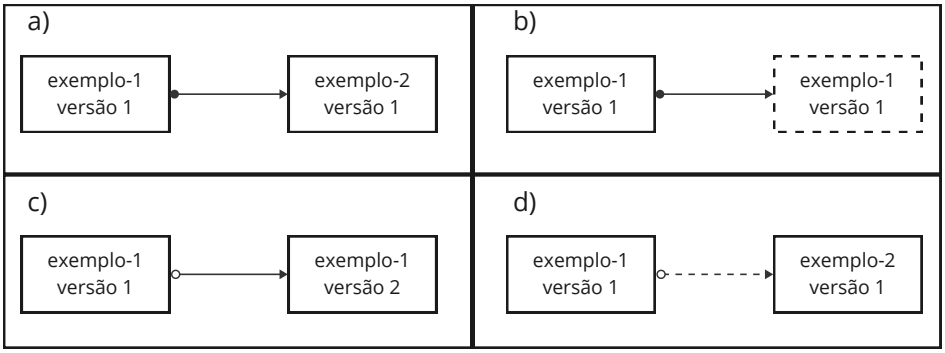

Figura 18 – Exemplos de transformações em características.

4.1.3.3 Adição e remoção de instâncias

A figura 19 apresenta três exemplos de adição e remoção de instâncias de dados.

No exemplo a, o conjunto 'exemplo-1' é enriquecido com novas instâncias de dados, resultando em um nova versão do conjunto. No exemplo b, o conjunto passar por um limpeza de dados, removendo instâncias que possuem valores nulos, novamente resultando em uma nova versão do conjunto.

O exemplo c apresenta um caso de divisão dos dados em treinamento e teste. Nesse caso, ambos os conjuntos resultantes de treinamento e teste são armazenados como amostras do conjunto 'exemplo-1'

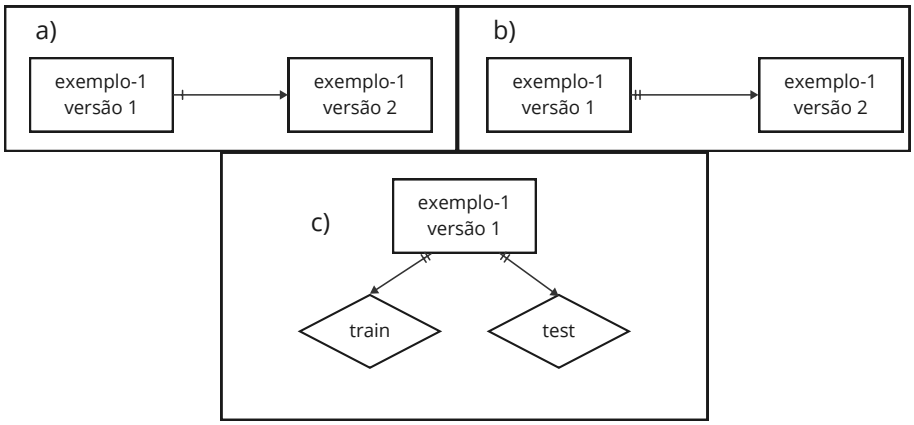

Figura 19 – Exemplos de adição e remoção de instâncias.

4.1.3.4 Exemplo geral

A figura 20 apresenta um diagrama completo de conjuntos processados a partir de cinco arquivos de dados brutos. Os conjunto em ciano são conjuntos cujo número do cartão de crédito é o campo indicador do propósito. Conjuntos em amarelo possuem o cpf da pessoa como propósito.

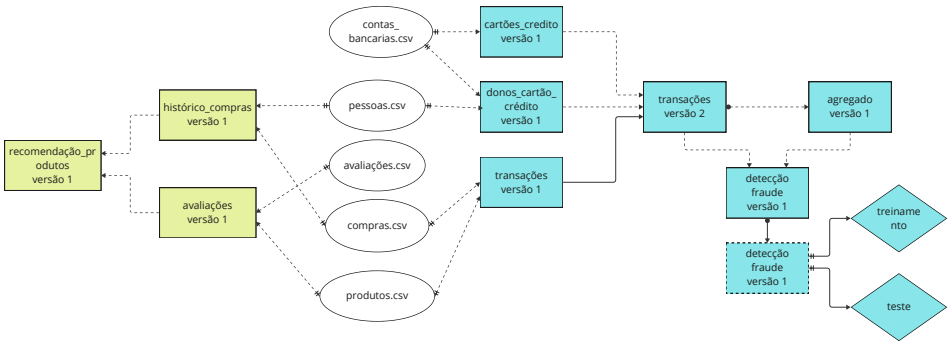

Figura 20 – Exemplo geral do diagrama.

4.2 Sistema de obtenção de proveniência

Apesar das *Feature Stores* avaliadas possuírem mecanismos que possam ajudar no entendimento do processamento das características, como descrições e controle de versão, tais mecanismos não são o suficiente para atingir o nível de proveniência desejado neste trabalho. Vendo que nenhuma *Feature Store* apresenta uma proveniência específica das características, que é o foco deste trabalho, foi desenvolvido um sistema que fornece maneiras de alcançar essa proveniência em conjunto com a utilização da *Feature Store*.

Tendo isso em mente, foi proposto um sistema que realiza uma integração entre a *Feature Store* e um repositório com controle de versão usando Git. A ideia principal consiste em armazenar no repositório os scripts utilizados no processamento das características. Algumas informações básicas são mantidas diretamente, como a origem das características processadas, enquanto que informações complexas, como transformações aplicadas, podem ser visualizadas inspecionando o conteúdo do script de processamento utilizado.

O sistema de integração também disponibiliza uma API, desenvolvida com o intuito de ser um incremento a API da própria *Feature Store*. Nessa API, são disponibilizadas funções que listam os scripts de processamento disponíveis, o conteúdo de tais scripts e também informações sobre a origem das características, disponibilizadas através de um grafo.

Apesar de simples, através do armazenamento de scripts é possível obter qualquer informação necessária sobre o processamento das características, fornecendo a proveniência necessária para garantir uma utilização segura das características armazenadas na *Feature Store*. Essa solução também apresenta um impacto mínimo no desempenho quando comparado a outras soluções para proveniência no processamento de dados.

4.2.1 Detalhamento técnico

A seguir é apresentado detalhes do funcionamento do sistema projetado para funcionar em conjunto com a *Feature Store* Hopsworks. Apesar de desenvolvido especialmente para a Hopsworks, é possível com poucas modificações adaptar o sistema para ser utilizado em conjunto com outras *Feature Stores*.

Para armazenamento dos scripts e de outras informações de proveniência, é utilizado um repositório na plataforma GitHub. Cada *Feature Group* possui um diretório próprio no repositório e cada versão de um *Feature Group* possui associado um script contendo o processamento das características daquela versão, que é armazenado no diretório do *Feature Group*. Também é armazenado no repositório um grafo de proveniência que disponibiliza as conexões entre os *Feature Groups*, como os ancestrais e descendentes de determinados *Feature Group*.

As funcionalidades do sistema são todas disponibilizadas através de uma biblioteca Python. A seguir são detalhadas as bibliotecas auxiliares utilizadas pelo sistema, a classe disponibilizada e seus métodos

O acesso à Hopsworks é feito utilizando a biblioteca hsfs, na qual disponibiliza funções para leitura e escrita na Feature Store. A biblioteca PyGithub é utilizada para o acesso ao repositório, disponibilizando funções de leitura e escrita que realizam chamadas à API REST do GitHub. O grafo de proveniência é feito utilizando a biblioteca NetworkX,

na qual disponibiliza diversos tipos de grafos e algoritmos para grafos.

A biblioteca desenvolvida disponibiliza a classe *IntegrationApi*, que realiza toda a integração entre a Hopsworks e o repositório GitHub. Para acesso dos métodos da classe é necessário instanciar um objeto, passando como parâmetro para o construtor as informações para conexão com a *Feature Store* e com o repositório GitHub.

O método *write_feature_group* é responsável pela escrita dos dados, desenvolvido com o intuito de ser utilizado no lugar da função de escrita disponibilizada pela biblioteca hsfs da Hopsworks. O método recebe como parâmetros obrigatórios os dados e o nome do *Feature Group* a serem inseridos, uma lista dos campos que atuam como chave primária para os dados, e o caminho do arquivo de script. Como parâmetro opcional, é possível especificar a fonte dos dados inseridos. O método cria o *Feature Group* e insere os dados na *Feature Store*, lê e escreve o conteúdo do script de processamento no GitHub, e atualiza o grafo de proveniência, inserindo no grafo o *Feature Group* adicionado na Feature Store e seus conjuntos de dados utilizados para gerar o *Feature Group*, podendo ser outros *Feature Groups* ou conjuntos de dados externos.

O método *list_feature_groups* busca na *Feature Store* todos os *Feature Groups* que possuem o script de processamento disponível, retornando um lista com os *Feature Groups* e o caminho de armazenamento do script no repositório.

O método *get_feature_group_script* obtém o conteúdo do script de processamento para um *Feature Group* específico. É necessário passar como parâmetro o nome e a versão do *Feature Group* desejado.

O método *get_ancestors_subgraph* obtém um subgrafo do grafo de proveniência contendo apenas os ancestrais de determinado *Feature Group*, isto é, as fontes de dados utilizadas para gerar o *Feature Group*. O método *get_descendants_subgraph* é similar ao método anterior, retornando um subgrafo com os descendentes de determinado *Feature Group*. Ambos os métodos necessitam do nome e versão do *Feature Group* como parâmetro.

Além dos métodos, a classe também possui as propriedades *feature_store* e *lineage_graph*. A primeira propriedade é utilizada para acesso à Feature Store, possibilitando o uso das função da API da Hopsworks através dela. A segunda propriedade é uma cópia do grafo de proveniência, que possibilita a visualização do grafo de maneira mais customizada.

A tabela 3 apresenta uma visão geral dos métodos disponibilizados pela biblioteca, especificando seus parâemtros e tipo de retorno, caso exista.

Tabela 3 – Documentação da Classe IntegrationApi e seus métodos

Método	Parâmetros obrigatórios	Parâmetros opcionais	Tipo de Retorno
Construtor	fs_project, fs_token, github_token, github_repo	fs_host	IntegrationApi
write_feature_group	name, primary_key, dataframe, script_path	parents, fs_additional_args	–
list_feature_groups	–	–	List[Dict[FeatureGroup,str]]
get_feature_group_script	fg_name, fg_version	–	str
get_ancestors_subgraph	fg_name, fg_version	–	networkx.DiGraph
get_descendants_subgraph	fg_name, fg_version	–	networkx.DiGraph

5 ESTUDO DE CASO

A seguir é realizado um estudo de caso com o propósito de detecção de fraudes em cartões de créditos. A inserção de dados na *Feature Store* é feita através do sistema de integração desenvolvido, utilizando os conceitos de entidade, relacionamento e premissas do diagrama proposto para a definição dos conjuntos salvos. A seguir são explicadas as bases de dados utilizadas, e o processamento realizado em diferentes etapas, evidenciando como são armazenadas informações de proveniência que facilitam a compreensão dos dados e seu processamento.

Para este estudo, foram utilizadas três base de dados. A primeira contém informações sobre cartões de crédito, incluindo número, provedor e data de expiração do cartão. A segunda contém informações pessoais sobre os donos dos cartões, incluindo nome, sexo, e-mail, data de nascimento, cidade e país. A terceira contém informações sobre transações realizada por cartões de crédito, incluindo data, coordenadas, categoria, valor, cidade, país, e uma indicador se a transação é fraudulenta ou não. Como todas as bases de dados compartilham uma chave em comum (número do cartão), é possível a junção das instâncias das bases de dados, expandindo as possibilidades de técnicas de engenharia de características a serem aplicadas.

5.1 Primeira etapa: Leitura e filtragem dos dados brutos

Inicialmente, foram importados os dados das três base de dados. Os dados passaram por um limpeza e filtragem antes de serem inseridos na *Feature store*. As bases de dados utilizadas se configuram como arquivos de dados brutos, sendo assim, de acordo com as premissas do diagrama proposto, os conjuntos resultantes foram todos obrigatoriamente armazenados como conjunto de características.

As figuras 21, 22 e 23 demonstram, respectivamente, o grafo de proveniência nesse ponto, um exemplo da página de detalhes de um *Feature Group* na interface da Hopsworks e o script de processamento disponível no repositório Git.

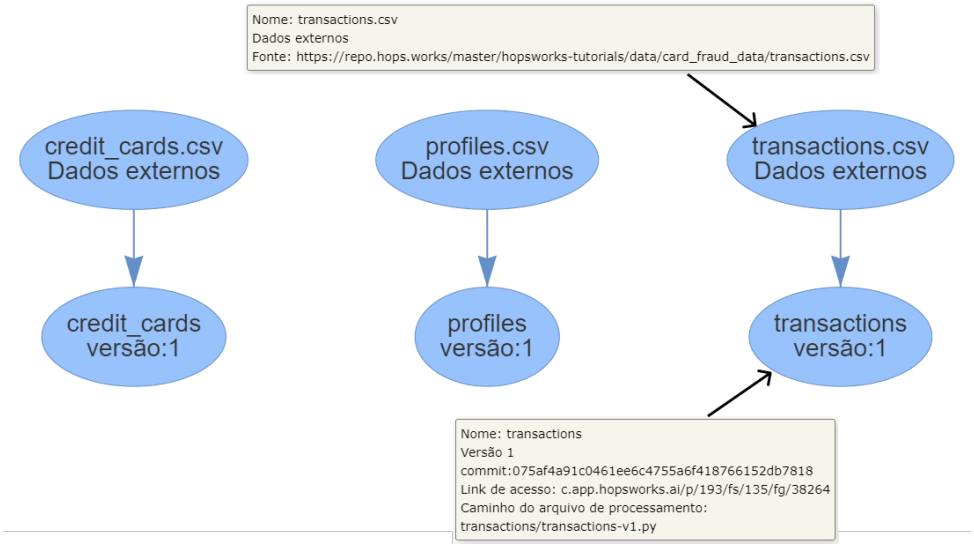

Figura 21 – Visualização do grafo de proveniência após a primeira etapa

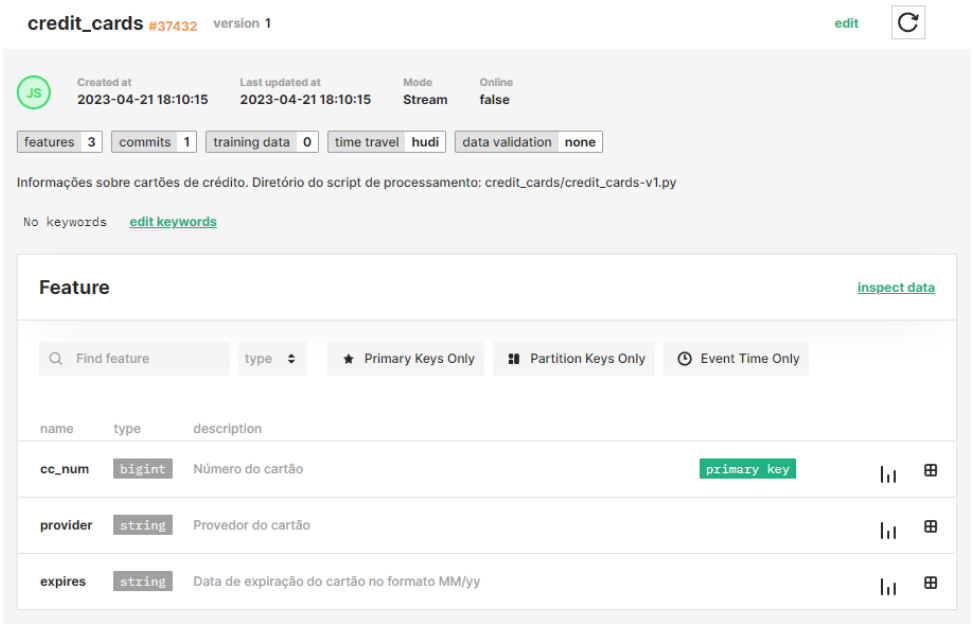

Figura 22 – Detalhes sobre o Feature Group 'credit_cards'

```
11  credit_cards = pd.read_csv(CREDIT_URL)
12  profiles = pd.read_csv(PROFILE_URL, parse_dates=["birthdate"])
13  transactions = pd.read_csv(TRANSACTION_URL, parse_dates=["datetime"])
14
15  # Removendo instâncias com valores nulos
16  credit_cards.dropna()
17  profiles.dropna()
18  transactions.dropna()
19
20  # Removendo pessoas fora dos estados unidos
21  profiles = profiles.loc[profiles.Country == 'US']
22
23  # Removendo transações fora dos estados unidos
24  transactions = transactions.loc[transactions.country == 'US']
25
26  api = IntegrationApi(FS_PROJECT,FS_KEY,GITHUB_TOKEN,GITHUB_REPO)
27
28  parents_cc = [{"name":"credit_cards.csv","source":CREDIT_URL}]
29  parents_prof = [{"name":"profiles.csv","source":PROFILE_URL}]
30  parents_trans = [{"name":"transactions.csv","source":TRANSACTION_URL}]
31
32  api.write_feature_group("credit_cards",["cc_num"],credit_cards,"exemplo1.py",parents_cc)
33  api.write_feature_group("profiles",["cc_num"],profiles,"exemplo1.py",parents_prof)
34  api.write_feature_group("transactions",["tid"],transactions,"exemplo1.py",parents_trans)
```

Figura 23 – Trecho do script de processamento dos conjuntos de características iniciais

Através do grafo é possível obter informações sobre um conjunto de características e seus antecessores. Ná página da interface da Hopsworks estão disponíveis metadados que ajudam a compreender os dados, como descrições e tipos das características. Algumas informações de proveniência também estão disponíveis, como data de criação, data de atualização e o perfil que criou o conjunto.

Analisando o script no repositório, é possível identificar o processamento realizado. As linhas 16, 17 e 18 demonstram que os três conjuntos de dados tiveram instâncias que possuíam ao menos uma coluna com valor nulo removidas. A linha 21 indica que para o conjunto de perfis, apenas instâncias de pessoas residentes nos Estados Unidos foram mantidas. A linha 24 indica que apenas transações realizadas nos Estado Unidos foram mantidas no conjunto de transações.

5.2 Segunda etapa: Computação de novas características para transações

Após a limpeza e filtragem inicial dos dados, foram computadas novas características a partir das características existentes. O conjunto de transações foi enriquecido com três novas características, a idade do dono do cartão durante a transação, a quantidade de dias restantes à expiração do cartão e a distância física entre a transação atual e a ultima transação realizada pelo mesmo cartão. O conjunto final foi salvo como uma nova versão do conjunto de transações por ser uma melhoria do conjunto original.

Também foi criado um novo conjunto de características sobre transações, contendo apenas transações da categoria de saque de dinheiro. Tal conjunto foi conjunto foi armazenado com um novo conjunto de características por ter tido a característica 'category' removida.

A figura 24 demonstra o grafo de proveniência neste momento, destacando as informações dos novos *Feature Groups* inseridos. As figuras 25 e 26 exibem o processamento realizado para os conjuntos de características 'transactions' e 'transactions_withdrawal_only'.

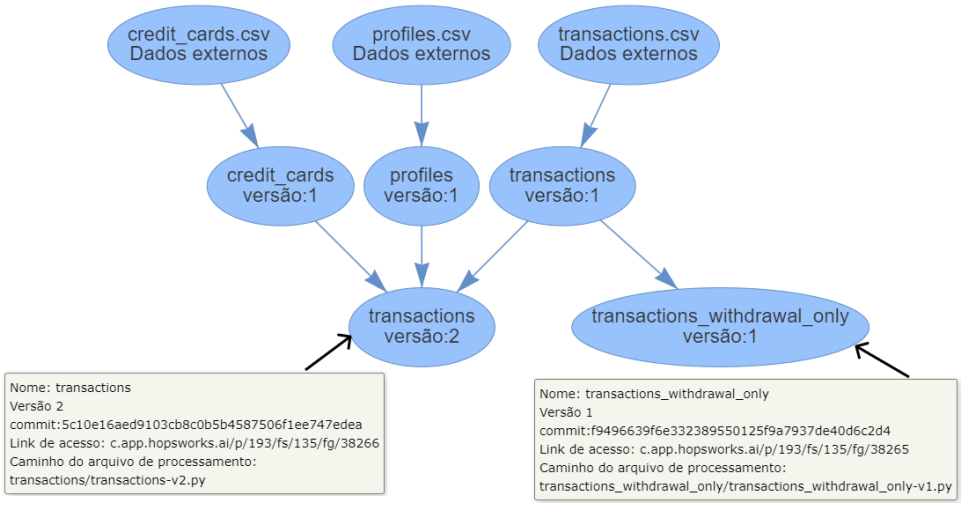

Figura 24 – Visualização do grafo de proveniência após a segunda etapa

```
# Calculando idade do dono do cartão durante a transação
age = transactions.merge(profiles, on="cc_num", how="left")
transactions["age_at_transaction"] = (age["datetime"] - age["birthdate"]) / np.timedelta64(1, "Y")

# Calculando dias restantes do cartão durante a transação.
card_expiry = transactions.merge(credit_cards, on="cc_num", how="left")
card_expiry["expires"] = pd.to_datetime(card_expiry["expires"], format="%m/%y")
transactions["days_until_card_expires"] = (card_expiry["expires"] - card_expiry["datetime"]) / np.timedelta64(1, "D")


# Pré-processamento para calcular distância entre transação atual e anterior.
transactions.sort_values("datetime", inplace=True)
transactions[["longitude", "latitude"]] = transactions[["longitude", "latitude"]].applymap(radians)

# Função auxiliar que calcula a distância entre cada coordenada
def haversine(long, lat):
    long_shifted = long.shift()
    lat_shifted = lat.shift()
    long_diff = long_shifted - long
    lat_diff = lat_shifted - lat

    a = np.sin(lat_diff/2.0)**2
    b = np.cos(lat) * np.cos(lat_shifted) * np.sin(long_diff/2.0)**2
    c = 2*np.arcsin(np.sqrt(a + b))

    return c

transactions["loc_delta"] = transactions.groupby("cc_num")\
    .apply(lambda x : haversine(x["longitude"], x["latitude"]))\
    .reset_index(level=0, drop=True)\
    .fillna(0)
```

Figura 25 – Trecho do script de processamento do conjunto de característica 'transaction'

```
9   fg_transactions = api.feature_store.get_feature_group("transactions",1)
10  data = fg_transactions.read()
11
12  # Deixando apenas instâncias com a categoria 'Cash Withdrawal'
13  data = data.loc[data.category == 'Cash Withdrawal']
14  data = data.drop(columns=['category'])
15
16  parents = [{"name":"transactions","version":1}]
17  api.write_feature_group("transactions_withdrawal_only",["tid"],data,"exemplo2.py",parents)
```

Figura 26 – Trecho do script de processamento do conjunto de característica 'transaction_withdrawal_only'

5.3 Terceira etapa: Computação de estatísticas agregadas e seleção de características

Para o terceiro passo, foi criado um novo conjunto de características a partir do conjunto de transações, contendo como características estatísticas de transações do mesmo cartão nas 4 horas anteriores a ocorrência de uma transação, entre elas, média móvel e desvio padrão dos valores das transações, média móvel da frequência de transações e média móvel da distância física entre transações. Apesar de ter se originado do conjunto de transações, o conjunto resultante foi salvo como um novo conjunto de características por não mantar todas as características originais do conjunto de transações.

Após isso, foram selecionadas, a partir dos conjuntos de transações e agregado de transações, as características mais relevantes para detecção de fraude. Como nenhum dos conjuntos teve todas suas características selecionadas, o conjunto resultante foi armazenada como um novo conjunto de características, denominado de 'fraud_detection'.

Através do script exibidos nas figuras 28 e 29 é possível determinar como foram computadas as estatísticas agregadas sobre transações. O script da figura 30 demonstra quais características foram selecionadas para a criação do conjunto de características 'fraud_detection'.

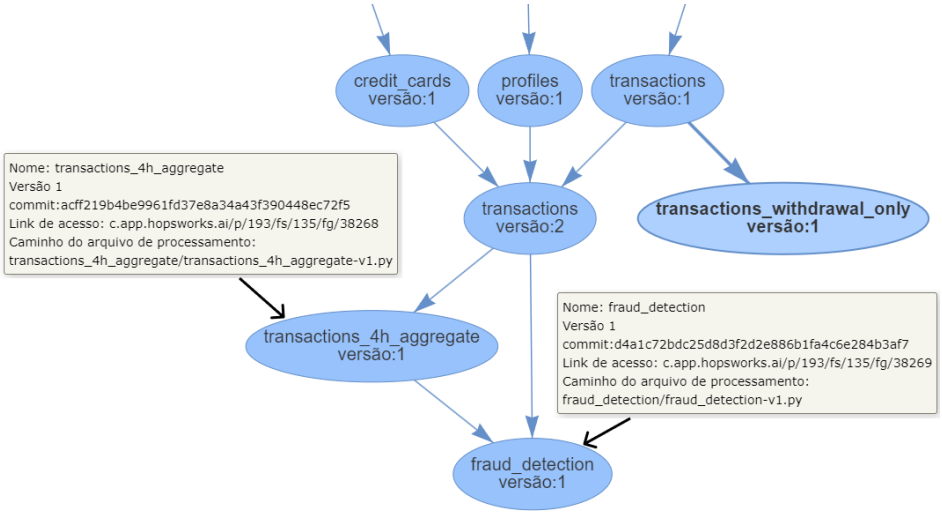

Figura 27 – Visualização do grafo de proveniência após a terceira etapa

```
api = IntegrationApi(FS_PROJECT,FS_KEY,GITHUB_TOKEN,GITHUB_REPO)
fg = api.feature_store.get_feature_group("transactions",2)
transactions = fg.read()

window_len = "4h"
transactions.sort_values("datetime", inplace=True)
cc_group = transactions[["cc_num", "amount", "datetime"]].groupby("cc_num").rolling(window_len, on="datetime")

# Média móvel do volume de transações.
df_4h_mavg = pd.DataFrame(cc_group.mean())
df_4h_mavg.columns = ["trans_volume_mavg", "datetime"]
df_4h_mavg = df_4h_mavg.reset_index(level=["cc_num"])
df_4h_mavg = df_4h_mavg.drop(columns=["cc_num", "datetime"])
df_4h_mavg = df_4h_mavg.sort_index()

# Desvio padrão móvel do volume de transações.
df_4h_std = pd.DataFrame(cc_group.std())
df_4h_std.columns = ["trans_volume_mstd", "datetime"]
df_4h_std = df_4h_std.reset_index(level=["cc_num"])
df_4h_std = df_4h_std.drop(columns=["cc_num", "datetime"])
df_4h_std = df_4h_std.fillna(0)
df_4h_std = df_4h_std.sort_index()
window_aggs_df = df_4h_std.merge(df_4h_mavg,left_index=True, right_index=True)
```

Figura 28 – Trecho do script de processamento das estatísticas agregadas

```
# Média móvel da frequencia de transações.
df_4h_count = pd.DataFrame(cc_group.count())
df_4h_count.columns = ["trans_freq", "datetime"]
df_4h_count = df_4h_count.reset_index(level=["cc_num"])
df_4h_count = df_4h_count.drop(columns=["cc_num", "datetime"])
df_4h_count = df_4h_count.sort_index()
window_aggs_df = window_aggs_df.merge(df_4h_count,left_index=True, right_index=True)

# Média móvel da distância entre transações consecutivas.
cc_group = transactions[["cc_num", "loc_delta", "datetime"]].groupby("cc_num").rolling(window_len, on="datetime").mean()
df_4h_loc_delta_mavg = pd.DataFrame(cc_group)
df_4h_loc_delta_mavg.columns = ["loc_delta_mavg", "datetime"]
df_4h_loc_delta_mavg = df_4h_loc_delta_mavg.reset_index(level=["cc_num"])
df_4h_loc_delta_mavg = df_4h_loc_delta_mavg.drop(columns=["cc_num", "datetime"])
df_4h_loc_delta_mavg = df_4h_loc_delta_mavg.sort_index()
window_aggs_df = window_aggs_df.merge(df_4h_loc_delta_mavg,left_index=True, right_index=True)

window_aggs_df = window_aggs_df.merge(transactions[["cc_num", "datetime","tid"]].sort_index(),left_index=True, right_index=True)
```

Figura 29 – Trecho do script de processamento das estatísticas agregadas

```
trans_fg = api.feature_store.get_feature_group('transactions', version=2)
window_aggs_fg = api.feature_store.get_feature_group('transactions_4h_aggregate', version=1)

query = trans_fg.select(["fraud_label", "category", "amount", "age_at_transaction", "days_until_card_expires", "loc_delta"])\
    .join(window_aggs_fg.select_except(["cc_num", "datetime"]))

fraud_detection = query.read()

parents = [
    {"name":"transactions","version":2},
    {"name":"transactions_4h_aggregate","version":1}
]
api.write_feature_group("fraud_detection",["tid"],fraud_detection,"exemplo5.py",parents)
```

Figura 30 – Trecho do script de processamento do conjunto de características 'fraud_detection'.

5.4 Quarta etapa: Transformações de pré-processamento e divisão de treinamento e teste

Tendo todas as características desejadas para a detecção de fraudes, foram aplicadas transformações de pré-processamento no dados. Em seguida, o conjunto de dados foi dividido em conjuntos de treinamento e teste. O conjunto resultante das transformações foi armazenada como uma ramificação do conjunto de detecção de fraudes, enquanto que os conjunto de treinamento e teste foram armazenados como amostras desta ramificação.

Analisando o script na figura 32, é possível identificar quais características receberam quais transformações. O script na figura 33 demonstra a proporção de divisão entre treinamento e teste, e garante que ambos os conjunto de treinamento e teste foram gerados pela mesma função de divisão.

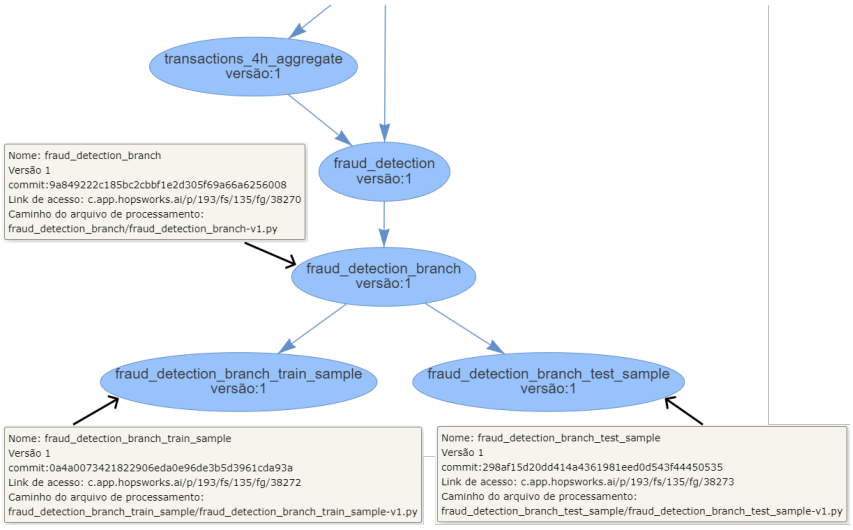

Figura 31 – Visualização do grafo de proveniência após a quarta etapa

```
fg = api.feature_store.get_feature_group('fraud_detection', version=1)
data = fg.read()

encoder = LabelEncoder()
scaler = StandardScaler()
data["category"] = encoder.fit_transform(data["category"])
columns = [
    "amount",
    "age_at_transaction",
    "days_until_card_expires",
    "loc_delta",
    "trans_volume_mstd",
    "trans_volume_mavg",
    "trans_freq",
    "loc_delta_mavg"
]
data[columns] = scaler.fit_transform(data[columns])

parents = [{"name":"fraud_detection","version":1}]
api.write_feature_group("fraud_detection_branch",["tid"],data,"exemplo6.py",parents)
```

Figura 32 – Trecho do script de processamento das transformações de pré-processamento

```
fg = api.feature_store.get_feature_group('fraud_detection_branch', version=1)
data = fg.read()

train, test = train_test_split(data, test_size=0.25)

parents = [{"name":"fraud_detection_branch","version":1}]
api.write_feature_group("fraud_detection_branch_train_sample",["tid"],train,"exemplo7.py",parents)
api.write_feature_group("fraud_detection_branch_test_sample",["tid"],test,"exemplo7.py",parents)
```

Figura 33 – Trecho do script de processamento da divisão entre treinamento e teste

5.5 Estado final

As figura 34 e 35 indicam o grafo de proveniência e o repositório GitHub após a execução de todas as etapas.

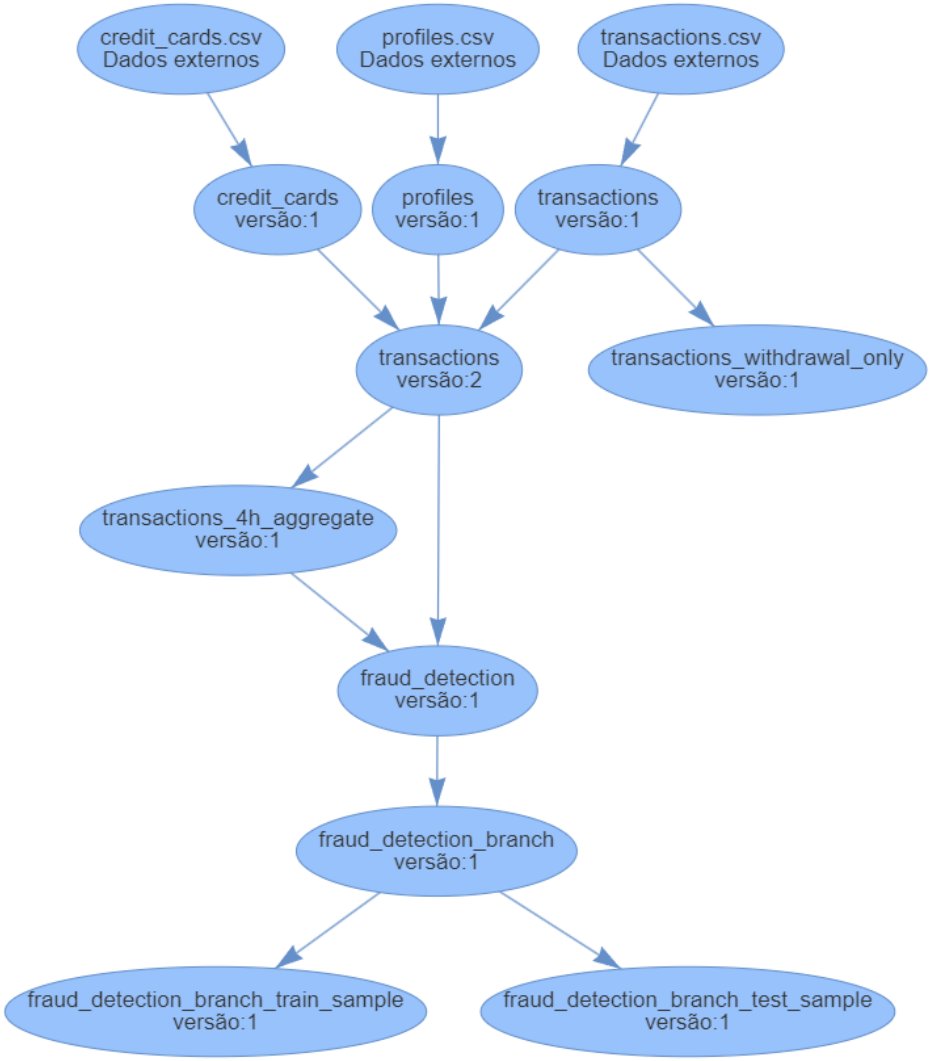

Figura 34 – Visualização do grafo de proveniência final

credit_cards	script de processamento credit_cards v1	42 minutes ago
fraud_detection	script de processamento fraud_detection v1	27 minutes ago
fraud_detection_branch	script de processamento fraud_detection_branch v1	24 minutes ago
fraud_detection_branch_test_sample	script de processamento fraud_detection_branch_test_sample v1	21 minutes ago
fraud_detection_branch_train_sample	script de processamento fraud_detection_branch_train_sample v1	21 minutes ago
profiles	script de processamento profiles v1	42 minutes ago
transactions	script de processamento transactions v2	33 minutes ago
transactions_4h_aggregate	script de processamento transactions_4h_aggregate v1	31 minutes ago
transactions_withdrawal_only	script de processamento transactions_withdrawal_only v1	37 minutes ago
featureGroups.txt	adicionando feature group fraud_detection_branch_test_sample	21 minutes ago
lineageGraph.json	inserindo feature group fraud_detection_branch_test_sample no grafo	21 minutes ago

Figura 35 – Repositório GitHub

Utilizando o grafo de proveniência em conjunto com os scripts do repositório, é

possível identificar a origem e o processamento de qualquer característica dos conjuntos de características armazenadas na *Feature Store*. Por exemplo, considere a característica 'loc_delta' do conjunto de características 'fraud_detection' destacada na figura 36.

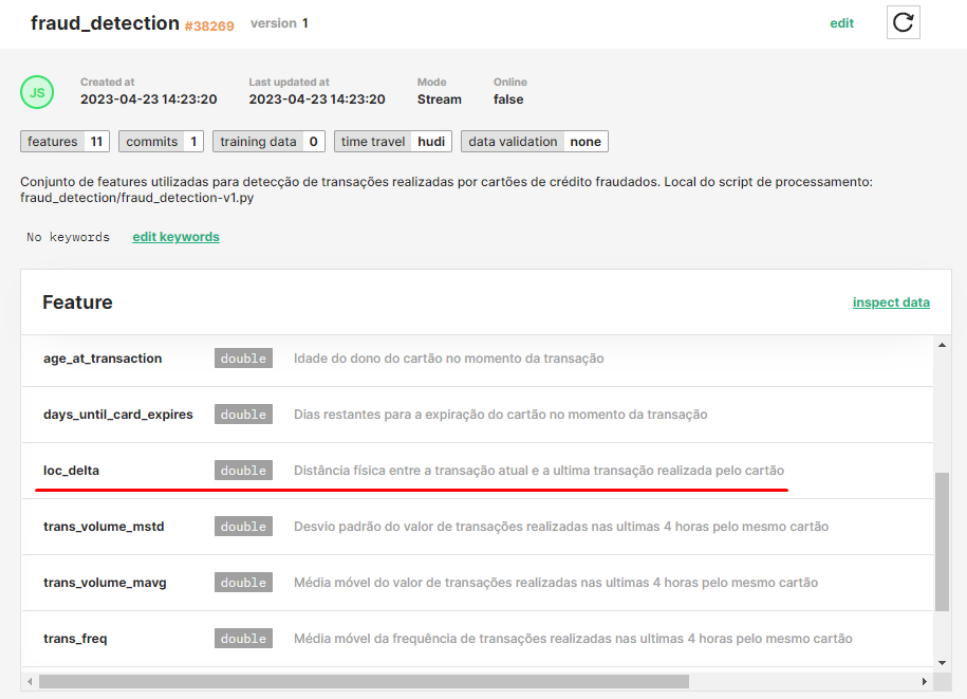

Figura 36 – Detalhes sobre o Feature Group 'fraud_detection'

Analisando o script da figura 30, é evidenciado, através da linha 12, que a característica 'loc_delta' foi selecionada a partir da versão 2 do conjunto de características 'transactions'. Visualizando o script de processamento de tal conjunto na figura 25, é possível identificar o trecho de código das linhas 28 a 44, responsável por computar a característica.

A linha 28 ordena o conjunto de acordo com a data das transações, enquanto que a linha 29 transforma as características 'latitude' e 'longitude' de graus para radianos. A linha 32 define a função que calcula a distância entre duas coordenadas utilizando a fórmula de haversine[1]. Por fim, a linha 44 computa a característica 'loc_delta' utilizando a função auxiliar definida anteriormente.

6 CONCLUSÃO

Visando potencializar a proveniência das características armazenadas em uma *Feature Store*, este trabalho desenvolveu um sistema de obtenção de proveniência, realizando uma integração entre a Hopsworks e o GitHub, tornando possível o armazenamento de técnicas de processamento utilizadas e grafos de linhagem das características. O sistema desenvolvido difere-se de outras propostas de obtenção de proveniência por apresentar uma abordagem simples e funcional, que possibilita a obtenção de informações de processamento independente de sua complexidade, com um impacto mínimo no desempenho do processamento e com uso específico para *Feature Stores*.

Além disso, também foi proposto um diagrama para o gerenciamento de dados, buscando padronizar o armazenamento de conjuntos de dados baseados em diferentes operações frequentemente utilizadas durante o preparo de dados para aprendizado de máquinas. O diagrama define diferentes entidades e relacionamentos capazes de descrever situações comuns no preparo de dados de aprendizado de máquina, fornecendo uma maneira genérica de ilustrar a linhagem de um conjunto de dados e as operações realizadas em tal conjunto no contexto específico de aprendizado de máquina.

Por fim, foi realizado um estudo de caso utilizando a *Feature Store* Hopsworks e o sistema de integração desenvolvido. Diversas operações foram realizadas sobre três base de dados diferentes, utilizando as premissas do diagrama como guia de armazenamento das características resultantes. Com o estudo de caso, foi evidenciado que o sistema de integração potencializa a capacidade de obtenção de proveniência para as características armazenadas na *Feature Store*, tornando possível saber, em detalhes, o processo de formação de cada característica.

Trabalhos futuros incluem o desenvolvimento de métodos para a captura automática de informações de proveniência mantidas pelo sistema, como conjuntos utilizados para a criação do conjunto a ser inserido. Outra vertente de trabalhos futuros incluem a modificação do código fonte da Hopsworks, de forma a incluir nativamente o sistema desenvolvido, possibilitando uma melhor integração das informações de proveniência, como grafo e scripts de processamento, com a interface disponibilizada pela Hopsworks.

REFERÊNCIAS

[1] ZEBARI, R. et al. A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. *Journal of Applied Science and Technology Trends*, v. 1, n. 2, p. 56–70, 2020.

[2] ORR, L. et al. Managing ml pipelines: feature stores and the coming wave of embedding ecosystems. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 14, n. 12, p. 3178–3181, 2021.

[3] IKEDA, R.; WIDOM, J. *Data lineage: A survey*. [S.l.], 2009.

[4] CHAPMAN, A. et al. Capturing and querying fine-grained provenance of preprocessing pipelines in data science. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 14, n. 4, p. 507–520, 2020.

[5] ZHOU, Z.-H. *Machine learning*. [S.l.]: Springer Nature, 2021.

[6] ZHOU, Y.; YU, Y.; DING, B. Towards mlops: A case study of ml pipeline platform. In: IEEE. *2020 International conference on artificial intelligence and computer engineering (ICAICE)*. [S.l.], 2020. p. 494–500.

[7] SCULLEY, D. et al. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, v. 28, 2015.

[8] LIU, H.; MOTODA, H. Feature transformation and subset selection. *IEEE Intell Syst Their Appl*, v. 13, n. 2, p. 26–28, 1998.

[9] SUGIMURA, P.; HARTL, F. Building a reproducible machine learning pipeline. *arXiv preprint arXiv:1810.04570*, 2018.

[10] PATEL, J. Unification of machine learning features. In: IEEE. *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. [S.l.], 2020. p. 1201–1205.

[11] ENGELEN, J. E. V.; HOOS, H. H. A survey on semi-supervised learning. *Machine Learning*, Springer, v. 109, n. 2, p. 373–440, 2020.

[12] NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, USA, 2015. v. 25.

[13] RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[14] HE, K. et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778.

[15] GOODFELLOW, I. et al. Generative adversarial networks. *Communications of the ACM*, ACM New York, NY, USA, v. 63, n. 11, p. 139–144, 2020.

[16] MOTODA, H.; LIU, H. Feature selection, extraction and construction. *Communication of IICM (Institute of Information and Computing Machinery, Taiwan)*, Citeseer, v. 5, n. 67-72, p. 2, 2002.

[17] KHALID, S.; KHALIL, T.; NASREEN, S. A survey of feature selection and feature extraction techniques in machine learning. In: IEEE. *2014 science and information conference*. [S.l.], 2014. p. 372–378.

[18] CAI, J. et al. Feature selection in machine learning: A new perspective. *Neurocomputing*, Elsevier, v. 300, p. 70–79, 2018.

[19] ZHENG, A.; CASARI, A. *Feature engineering for machine learning: principles and techniques for data scientists*. [S.l.]: "O'Reilly Media, Inc.", 2018.

[20] KHURANA, U. et al. Cognito: Automated feature engineering for supervised learning. In: IEEE. *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. [S.l.], 2016. p. 1304–1307.

[21] BABATUNDE, O. H. et al. A genetic algorithm-based feature selection. IJECCE, 2014.

[22] ORESKI, S.; ORESKI, G. Genetic algorithm-based heuristic for feature selection in credit risk assessment. *Expert systems with applications*, Elsevier, v. 41, n. 4, p. 2052–2064, 2014.

[23] FEAST. *Feast Documentation*. <https://docs.hopsworks.ai/3.1/>. Acessado em 24/01/2023.

[24] CLOCKS, L. *Hopsworks Documentation*. <https://docs.hopsworks.ai/3.1/>. Acessado em 24/01/2023.

[25] HERSCHEL, M.; DIESTELKÄMPER, R.; LAHMAR, H. B. A survey on provenance: What for? what form? what from? *The VLDB Journal*, Springer, v. 26, n. 6, p. 881–906, 2017.

[26] CHENEY, J. et al. Provenance in databases: Why, how, and where. *Foundations and Trends® in Databases*, Now Publishers, Inc., v. 1, n. 4, p. 379–474, 2009.

[27] GLAVIC, B. et al. Data provenance. *Foundations and Trends® in Databases*, Now Publishers, Inc., v. 9, n. 3-4, p. 209–441, 2021.

[28] SOUZA, R. et al. Provenance data in the machine learning lifecycle in computational science and engineering. In: IEEE. *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. [S.l.], 2019. p. 1–10.

[29] ORMENISAN, A. A. et al. Implicit provenance for machine learning artifacts. *Proceedings of MLSys*, v. 20, 2020.

[30] KAKANTOUSIS, T. et al. Horizontally scalable ml pipelines with a feature store. In: *Proc. 2nd SysML Conf., Palo Alto, USA*. [S.l.: s.n.], 2019.

[31] CERAR, G. et al. On designing data models for energy feature stores. *arXiv preprint arXiv:2205.04267*, 2022.

[32] PROV-DM: The PROV Data Model. <https://www.w3.org/TR/prov-dm/>. Acessado em 15/01/2023.

[33] PUBLIO, G. C. et al. Ml-schema: exposing the semantics of machine learning with schemas and ontologies. *arXiv preprint arXiv:1807.05351*, 2018.

[34] ORMENISAN, A. A. et al. Time travel and provenance for machine learning pipelines. In: *2020 USENIX Conference on Operational Machine Learning (OpML 20)*. [S.l.: s.n.], 2020.

[35] NAMAKI, M. H. et al. Vamsa: Automated provenance tracking in data science scripts. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. [S.l.: s.n.], 2020. p. 1542–1551.