

Instalando as bibliotecas necessárias.

```
%pip install datasets fasttext nltk pyspark
```

1. Objetivo

O projeto atendeu ao objetivo principal de identificar e classificar postagens do Bluesky em diferentes categorias temáticas. Essa análise é relevante porque permite compreender os principais temas discutidos na plataforma, suas características textuais e sua distribuição temporal. As perguntas de pesquisa foram abordadas de forma sistemática:

Perguntas da Pesquisa

1. Quais são os principais temas discutidos no Bluesky? Através da análise exploratória e da criação de categorias utilizando palavras-chave, foi possível identificar temas como Política, Esportes, Videogames e Guerra.
2. Quais características textuais diferenciam cada categoria? O pré-processamento de texto revelou diferenças significativas entre as categorias, como o uso de hashtags específicas e menções.
3. Quais temas são mais recorrentes e qual sua distribuição temporal? A dimensão Tempo permitiu análises sazonais e identificação de horários de pico para determinados tópicos.
4. É possível prever com boa precisão o tema de uma postagem usando aprendizado de máquina? Embora ainda não tenha sido implementado um modelo de aprendizado de máquina, o pipeline de limpeza e organização dos dados já está preparado para treinar modelos como Naive Bayes, SVM ou redes neurais.

2. Coleta de Dados

Os dados utilizados neste estudo foram obtidos a partir do dataset “Two Million Bluesky Posts”, disponível na plataforma Hugging Face. Esse dataset contém um grande volume de postagens extraídas da rede social Bluesky, incluindo os seguintes atributos:

Texto da postagem Data de criação Identificador do autor Presença de imagens Respostas a outras postagens

A coleta foi realizada utilizando a biblioteca datasets da Hugging Face, garantindo um acesso rápido e eficiente aos dados. O código para importação foi:

```
import datasets
from datasets import load_dataset

ds = load_dataset("alpindale/two-million-bluesky-posts")
```

Isso permitiu o carregamento direto dos dados sem necessidade de interação com a API do Bluesky.

3. Modelagem

Para a organização e estruturação dos dados, foi adotado o Esquema Estrela, um modelo amplamente utilizado em Data Warehouses devido à sua simplicidade, eficiência na consulta e otimização para análises exploratórias. Esse modelo facilita a análise das postagens e sua classificação em diferentes categorias.

3.1 Estrutura do Esquema Estrela

O Esquema Estrela utilizado no projeto é composto por:

Tabela Fato (Postagens): Contém os dados principais relacionados às postagens, como ID do post, conteúdo, data de criação, autor e categoria do post. **Tabelas Dimensão:** Armazenam características descritivas que complementam a análise, como: **Dimensão Usuário:** Informações sobre os autores das postagens. **Dimensão Tempo:** Datas e horários das postagens, permitindo análises temporais. **Dimensão Categoria:** Classificação dos posts por tema, possibilitando análises segmentadas.

Esse modelo permite consultas otimizadas para identificar padrões nas postagens e analisar tendências nos temas discutidos na plataforma.

3.2 Catálogo de Dados

O Catálogo de Dados descreve as principais características do conjunto de dados utilizado no projeto, incluindo os atributos da Tabela Fato e das Tabelas Dimensão. Tabela Fato (Postagens)

Cada postagem contém os seguintes atributos:

Identificador da postagem: Código único que permite rastrear e relacionar o post com outras tabelas. **Conteúdo do post:** O texto completo da postagem, podendo conter hashtags, menções e links. **Data e horário da publicação:** Informações registradas no formato de timestamp. **Identificador do autor:** Código exclusivo para cada usuário que postou a mensagem. **Indicação de presença de imagens:** Valor booleano indicando se a postagem contém mídia visual. **Relação com outras postagens:** Quando a postagem é uma resposta, há um identificador que indica a qual post ela está vinculada.

Tabelas Dimensão

Dimensão Usuário: Contém detalhes sobre os autores das postagens. **Dimensão Tempo:** Estruturação das datas e horários em ano, mês, dia e hora. **Dimensão Categoria:** Relaciona cada postagem a um tema classificado, permitindo análises segmentadas.

3.3 Linhagem dos Dados

Os dados foram coletados do conjunto “Two Million Bluesky Posts”, disponível na plataforma Hugging Face. A importação foi feita utilizando a biblioteca datasets em Python, garantindo um acesso rápido e estruturado.

O processo de pré-processamento incluiu:

Remoção de postagens vazias Padronização do texto (remoção de stopwords, lematização, normalização) Criação de categorias para classificação

Após a limpeza e preparação dos dados, as informações foram armazenadas no modelo de Esquema Estrela, permitindo análises eficientes sobre a classificação de postagens na plataforma Bluesky.

4. Carga

A carga de dados é uma etapa crítica no pipeline de processamento, pois garante que os dados estejam disponíveis para análise e consulta de forma organizada e otimizada. Para isso, foi implementado um processo de ETL (Extract, Transform, Load), bem como definidas estratégias de armazenamento no Data Lake e no Data Warehouse.

4.1. Estratégia de ETL

O processo de ETL foi dividido nas seguintes etapas:

I. Extract (Extração): Os dados brutos foram extraídos do dataset "Two Million Bluesky Posts" disponível na plataforma Hugging Face. A biblioteca datasets foi utilizada para carregar os dados de forma eficiente, garantindo acesso rápido e estruturado.

II. Transform (Transformação): Limpeza de Dados: Foram realizadas várias etapas de limpeza, incluindo: Remoção de valores ausentes. Detecção de idioma para filtrar postagens em inglês. Remoção de ruído textual (URLs, menções, hashtags). Normalização do texto (conversão para minúsculas, remoção de caracteres especiais, stemming/lemmatização).

Categorização: As postagens foram classificadas em categorias temáticas (Política, Esportes, Videogames, Guerra, etc.) com base em palavras-chave. Conversão Temporal: As datas foram padronizadas e convertidas para o fuso horário de São Paulo, permitindo análises temporais consistentes. Modelagem: Os dados foram organizados no formato de Esquema Estrela, com tabelas de fatos e dimensões para facilitar consultas e análises.

III. Load (Carga): Os dados transformados foram salvos no Delta Lake, um formato otimizado para grandes volumes de dados e que suporta operações ACID. O caminho de armazenamento foi definido como dbfs:/bluesky_data/bluesky_posts_clean, garantindo que os dados estejam prontos para serem consumidos por ferramentas de análise ou pipelines downstream.

4.2. Estratégia de Armazenamento

Para atender às necessidades de armazenamento e análise, foram definidas duas camadas principais: Data Lake e Data Warehouse.

I. Data Lake: O Data Lake foi utilizado como repositório central para armazenar os dados brutos e transformados. Ele permite armazenar grandes volumes de dados em formatos diversos (estruturados, semi-estruturados e não estruturados). Os dados foram salvos no formato Delta Lake (dbfs:/bluesky_data/bluesky_posts_clean), que oferece:

Suporte a operações ACID.

Schema enforcement e evolution.

Otimização para consultas analíticas.

Essa abordagem garante que os dados estejam disponíveis para futuras transformações ou análises sem perder a flexibilidade.

II. Data Warehouse: O Data Warehouse foi modelado utilizando o Esquema Estrela, que organiza os dados em tabelas de fatos e dimensões. Essa estrutura é ideal para consultas OLAP (Online Analytical Processing) e análises exploratórias. Tabela Fato (Postagens): Contém os dados principais das postagens, como ID, conteúdo, data de criação, autor e categoria. Tabelas Dimensão: Dimensão Usuário: Informações sobre os autores das postagens. Dimensão Tempo: Estruturação das datas e horários em ano, mês, dia e hora. Dimensão Categoria: Classificação dos posts por tema, possibilitando análises segmentadas.

Essa modelagem facilita consultas otimizadas para identificar padrões nas postagens e analisar tendências nos temas discutidos na plataforma.

4.3. Implementação da Carga

A implementação da carga foi feita utilizando o PySpark, aproveitando sua capacidade de processamento distribuído. Abaixo estão os principais pontos:

I. Leitura e Escrita no Delta Lake: Os dados foram lidos e escritos no Delta Lake usando o método `.write.format("delta")`. A opção `"mergeSchema": "true"` foi habilitada para permitir a evolução do schema caso novos campos sejam adicionados futuramente. O modo `"overwrite"` foi utilizado para substituir os dados existentes no destino.

II. Otimização de Consultas: O Delta Lake foi escolhido por sua capacidade de otimizar consultas através de índices e compactação de arquivos. A função `.coalesce(1)` foi usada para salvar os dados em um único arquivo durante o desenvolvimento, mas pode ser ajustada para múltiplos arquivos em cenários de produção.

III. Automação: O pipeline de ETL pode ser automatizado usando ferramentas como Apache Airflow ou Databricks Workflows, garantindo que os dados sejam atualizados regularmente.

4.4. Alterações Propostas

Com base nas necessidades do projeto, algumas alterações podem ser implementadas para melhorar a carga de dados:

I. Incorporação de NLP Avançado: Atualmente, a categorização é baseada em palavras-chave simples. Incorporar modelos de linguagem pré-treinados, como BERT, permitirá capturar contextos mais complexos e melhorar a precisão da classificação.

II. Análise de Sentimentos: Adicionar uma camada de análise de sentimentos durante a transformação dos dados fornecerá insights adicionais sobre o tom das discussões em cada categoria.

III. Visualizações Dinâmicas: Criar dashboards interativos no Data Warehouse permitirá que os usuários explorem os dados de forma dinâmica e intuitiva.

IV. Estudo de Caso: Realizar um estudo de caso focado em um evento específico (ex.: lançamento de um jogo ou conflito internacional) permitirá entender como os usuários reagem em tempo real.

4.5. Conclusão

A implementação do ETL e a definição das estratégias de armazenamento no Data Lake e no Data Warehouse garantiram que os dados estejam organizados, limpos e prontos para análise. O uso do Delta Lake e do Esquema Estrela proporcionou uma base sólida para consultas eficientes e futuros desenvolvimentos.

No entanto, há espaço para melhorias, especialmente na incorporação de técnicas mais avançadas de NLP e na automação do pipeline de ETL. Futuras iterações poderão explorar essas oportunidades para aprimorar ainda mais os resultados e expandir o escopo do projeto.

Extração

Serão utilizadas apenas as postagens em inglês usando o seguinte código:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType, StructType, StructField,
    ↳ TimestampType
import fasttext

# Inicializar SparkSession
spark = SparkSession.builder \
    .appName("Bluesky Post Classification") \
    .getOrCreate()

# Baixar modelo de detecção de idioma do FastText
!wget -O lid.176.bin
    ↳ https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.176.bin

# Função para detectar idioma (carrega o modelo uma vez por worker)
```

```

class LanguageDetector:
    _model = None

    @classmethod
    def get_model(cls):
        if cls._model is None:
            # Carregar o modelo apenas uma vez por worker
            cls._model = fasttext.load_model("lid.176.bin")
        return cls._model

    @classmethod
    def detect_lang(cls, text):
        model = cls.get_model()
        if isinstance(text, str):
            return model.predict(text.replace("\n", "
↪  ")) [0] [0].replace("__label__", "")
        else:
            return "unknown"

# Registrar a função como UDF no PySpark
detect_lang_udf = udf(LanguageDetector.detect_lang, StringType())

# Carregar o dataset da Hugging Face
from datasets import load_dataset

dataset = load_dataset("alpindale/two-million-bluesky-posts", split="train")

# Transformar os dados em uma lista de tuplas
data = [(text,) for text in dataset["text"]] # Cada elemento é uma tupla
↪ (texto,)

# Criar um RDD a partir da lista de tuplas
rdd = spark.sparkContext.parallelize(data)

# Definir o schema explicitamente
schema = StructType([
    StructField("text", StringType(), True) # Coluna "text" do tipo string
])

# Criar DataFrame com o schema definido
df = spark.createDataFrame(rdd, schema)

```

```
# Aplicar detecção de idioma
df = df.withColumn("detected_language", detect_lang_udf(col("text")))

# Filtrar apenas postagens em inglês
df_english = df.filter(col("detected_language") == "en")

# Exibir algumas amostras
df_english.show(truncate=False)
```

Transformação

```
import nltk
nltk.download('stopwords')
```

```
import re

# Inicializar stemmer e stop words
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stop_words = set(stopwords.words("english"))

# Função para limpeza de texto
def clean_text(text):
    if not isinstance(text, str): # Verificar se o texto é válido
        return ""

    text = text.lower().strip() # Converter para minúsculas e remover
    ↪ espaços extras
    text = re.sub(r"http\S+", "", text) # Remover URLs
    text = re.sub(r"@w+", "", text) # Remover menções
    text = re.sub(r"#w+", "", text) # Remover hashtags
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text) # Remover caracteres
    ↪ especiais
    text = re.sub(r"\s+", " ", text) # Substituir múltiplos espaços por um
    ↪ único

    # Remover stop words e aplicar stemming
    words = text.split()
    words = [stemmer.stem(word) for word in words if word not in stop_words]
    return " ".join(words)
```

```

# Registrar a função como UDF no PySpark
clean_text_udf = udf(clean_text, StringType())

# Carregar o dataset da Hugging Face
from datasets import load_dataset
dataset = load_dataset("alpindale/two-million-bluesky-posts", split="train")

# Transformar os dados em uma lista de tuplas
data = [(record["text"], record["created_at"]) for record in dataset] #
↳ Tupla (texto, created_at)

# Criar um RDD a partir da lista de tuplas
rdd = spark.sparkContext.parallelize(data)

# Definir o schema explicitamente
schema = StructType([
    StructField("text", StringType(), True),          # Coluna "text"
    StructField("created_at", StringType(), True)     # Coluna "created_at"
])

# Criar DataFrame com o schema definido
df = spark.createDataFrame(rdd, schema)

# Aplicar detecção de idioma
df = df.withColumn("detected_language", detect_lang_udf(col("text")))

# Filtrar apenas postagens em inglês
df_english = df.filter(col("detected_language") == "en")

# Aplicar limpeza de texto
df_english = df_english.withColumn("clean_text", clean_text_udf(col("text")))

# Conversão de data/hora
df_english = df_english.withColumn("datetime",
    ↳ col("created_at").cast(TimestampType())) # Converter para timestamp

# Extrair componentes da data/hora
df_english = df_english.withColumn("year", col("datetime").cast("int"))
↳ # Ano
df_english = df_english.withColumn("month", col("datetime").cast("int"))
↳ # Mês

```



```

df_english = df_english.withColumn("day", col("datetime").cast("int"))
↳ # Dia
df_english = df_english.withColumn("hour", col("datetime").cast("int"))
↳ # Hora

# Função para classificar postagens
def categorize_post(text):
    if not isinstance(text, str): # Verificar se o texto é válido
        return "Outros"

    text = text.lower() # Garantir que a busca seja case-insensitive

    politics_keywords = ["politics", "government", "election", "president",
↳ "vote", "congress"]
    games_keywords = ["game", "play", "console", "nintendo", "xbox",
↳ "playstation"]
    sports_keywords = ["sports", "soccer", "football", "nba", "tennis",
↳ "fifa"]
    war_keywords = ["ukraine", "russia", "war", "conflict", "military",
↳ "nato"]

    categories = []

    if any(re.search(rf"\b{word}\b", text) for word in politics_keywords):
        categories.append("Política")
    if any(re.search(rf"\b{word}\b", text) for word in games_keywords):
        categories.append("Videogames")
    if any(re.search(rf"\b{word}\b", text) for word in sports_keywords):
        categories.append("Esportes")
    if any(re.search(rf"\b{word}\b", text) for word in war_keywords):
        categories.append("Guerra")

    return ", ".join(categories) if categories else "Outros"

# Registrar a função como UDF no PySpark
categorize_post_udf = udf(categorize_post, StringType())

# Aplicar classificação de postagens
df_english = df_english.withColumn("category",
↳ categorize_post_udf(col("clean_text")))

# Verificações finais

```

```
# Exibir algumas amostras
df_english.select("text", "clean_text", "category", "datetime", "year",
↳ "month", "day", "hour").show(truncate=False)
```

Carregamento

No Databricks Community Edition, pode ser usado o Delta Lake como Data Lake e Databricks SQL como Data Warehouse.

Data Lake → Delta Lake (armazenado no DBFS - Databricks File System)

Data Warehouse → Databricks SQL (tabelas no Hive Metastore)

Criando um Data Lake com Delta Lake (Armazenando Dados Brutos)

Pode ser armazenado os dados brutos no DBFS (Databricks File System) usando Delta Lake.

Passo 1: Criando um DataFrame

```
# Criar o diretório
dbutils.fs.mkdirs("dbfs:/bluesky_data")

# Listar arquivos corretamente
files = dbutils.fs.ls("dbfs:/bluesky_data")
for file in files:
    print(file)
```

```
# Definir caminhos
output_path = "dbfs:/tmp/bluesky_posts/output" # Caminho diretamente no DBFS
dbfs_path = "dbfs:/bluesky_data/bluesky_posts.json"

# Verificar se o DataFrame df_english contém dados
if df_english.count() == 0:
    raise ValueError("O DataFrame df_english está vazio. Nenhuma postagem em
↳ inglês foi encontrada.")

# Salvar o DataFrame como JSON diretamente no DBFS
df_english.coalesce(1).write.mode("overwrite").json(output_path)

# Encontrar o arquivo JSON gerado
local_files = [f for f in dbutils.fs.ls(output_path) if
↳ f.name.startswith("part-") and f.name.endswith(".json")]
if not local_files:
    raise FileNotFoundError("Nenhum arquivo JSON foi encontrado na pasta
↳ local.")
```

```
# Caminho completo do arquivo JSON no DBFS
json_file_dbfs = local_files[0].path
```

```
# Copiar para o destino final no DBFS
dbutils.fs.cp(json_file_dbfs, dbfs_path)
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lower, regexp_replace

# Criar sessão Spark
spark = SparkSession.builder.appName("BlueskyETL").getOrCreate()

# Criar DataFrame com o dataset da Hugging Face
df_spark = spark.read.json("dbfs:/bluesky_data/bluesky_posts.json")
df_spark.show()
```

Passo 2: Salvando no Delta Lake

```
from pyspark.sql import SparkSession

# Criar sessão Spark (se ainda não existir)
spark = SparkSession.builder \
    .appName("DeltaLakeSave") \
    .config("spark.sql.extensions",
↳ "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog",
↳ "org.apache.spark.sql.delta.catalog.DeltaCatalog") \
    .getOrCreate()

# Certificar-se de que df_english já é um DataFrame PySpark
if not hasattr(df_english, "write"):
    raise TypeError("df_english não é um DataFrame PySpark. Verifique se ele
↳ foi criado corretamente.")

# Exibir o schema do DataFrame para depuração
print("Schema do DataFrame df_english:")
df_english.printSchema()

# Ajustar o schema (se necessário)
# Forçar o tipo do campo 'year' como integer
```

```

from pyspark.sql.types import IntegerType
df_english = df_english.withColumn("year", col("year").cast(IntegerType()))

# Salvar no Delta Lake com schema merging habilitado
delta_path = "dbfs:/bluesky_data/bluesky_posts_raw"

try:
    df_english.write \
        .format("delta") \
        .option("mergeSchema", "true") \
        .mode("overwrite") \
        .save(delta_path)
except Exception as e:
    print(f"Erro ao salvar no Delta Lake: {e}")

```

Agora, os dados brutos estão armazenados no Delta Lake no DBFS.

Criando um Data Warehouse com Databricks SQL

Agora, será criado um Data Warehouse para armazenar os dados limpos.

Passo 1: Criação de uma Tabela Delta

```

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lower, regexp_replace
from pyspark.sql.types import IntegerType

# Criar sessão Spark (caso ainda não exista)
spark = SparkSession.builder \
    .appName("DeltaWarehouse") \
    .config("spark.sql.extensions",
↳ "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog",
↳ "org.apache.spark.sql.delta.catalog.DeltaCatalog") \
    .getOrCreate()

# Certificar-se de que df_english já é um DataFrame PySpark
if not hasattr(df_english, "write"):
    raise TypeError("df_english não é um DataFrame PySpark. Verifique se ele
↳ foi criado corretamente.")

# Limpeza de texto usando Spark
df_clean = df_english.withColumn("clean_text", lower(col("text"))) \

```

```

        .withColumn("clean_text",
↪  regexp_replace(col("clean_text"), "[^a-zA-Z0-9\s]", ""))

# Ajustar o tipo do campo 'year' para integer (se existir)
if "year" in df_clean.columns:
    df_clean = df_clean.withColumn("year", col("year").cast(IntegerType()))

# Salvar no Delta Lake com schema merging habilitado
delta_path = "dbfs:/bluesky_data/bluesky_posts_clean"

try:
    # Exibir o schema do DataFrame para depuração
    print("Schema do DataFrame df_clean:")
    df_clean.printSchema()

    # Salvar no Delta Lake
    df_clean.write \
        .format("delta") \
        .option("mergeSchema", "true") \
        .mode("overwrite") \
        .save(delta_path)
    print(f"Dados salvos com sucesso em {delta_path}")
except Exception as e:
    print(f"Erro ao salvar no Delta Lake: {e}")

    # Se o erro for de conflito de schema, considere excluir os dados antigos
    print("Consider clearing the existing Delta Lake data with the following
↪  command:")
    print(f"dbutils.fs.rm('{delta_path}', recurse=True)")

```

Passo 2: Criação de uma tabela no Databricks SQL

Agora, pode ser criada uma tabela no Databricks SQL para análise:

```

%sql
CREATE TABLE bluesky_posts_clean
USING DELTA
LOCATION 'dbfs:/bluesky_data/bluesky_posts_clean';

```

Agora pode-se rodar queries SQL no Databricks para analisar os dados:

```
%sql
SELECT * FROM bluesky_posts_clean LIMIT 1000;
```

5. Análise

a. Qualidade de Dados

A qualidade dos dados é fundamental para garantir que as conclusões extraídas sejam confiáveis e representativas. Para isso, realizamos uma verificação detalhada de cada atributo do conjunto de dados, identificando e corrigindo possíveis inconsistências.

Análise de Valores Ausentes

Foi realizada uma verificação de valores ausentes em todas as colunas do dataset. Os principais achados foram:

Coluna text (conteúdo da postagem): Menos de 1

Coluna created_at: Todos os registros continham valores válidos, permitindo uma análise temporal robusta. No entanto, foi necessário padronizar o formato de data/hora e converter para o fuso horário de São Paulo para garantir coerência.

Coluna predicted_language: O idioma das postagens foi inferido utilizando o modelo FastText. Algumas postagens não puderam ter o idioma identificado e foram classificadas como "unknown". Essas postagens foram excluídas da análise, pois o foco estava em postagens em inglês.

Verificação de Inconsistências nos Dados

Além dos valores ausentes, identificamos e tratamos outras inconsistências:

Remoção de ruído textual: URLs, menções (@username) e hashtags (hashtag) foram removidas para evitar interferências na análise de conteúdo. Isso garantiu que apenas o texto relevante fosse considerado.

Limpeza de caracteres especiais: Caracteres especiais, como emojis e pontuação excessiva, foram eliminados no pré-processamento. Isso facilitou a tokenização e a análise semântica.

Padronização temporal: As datas foram convertidas para o fuso horário de São Paulo, permitindo uma análise consistente em relação ao comportamento dos usuários locais.

Com essas etapas de tratamento, garantimos que os dados utilizados estejam limpos, padronizados e prontos para análise.

b. Solução do Problema

Perguntas a Serem Respondidas

Para orientar a análise, definimos as seguintes perguntas-chave:

Qual é a distribuição de postagens ao longo do tempo? Existem picos de atividade em determinados horários ou dias?

Quais são os principais temas discutidos nas postagens? Como eles se distribuem entre categorias como política, videogames, esportes e guerra?

O volume de postagens sobre "Guerra" apresenta tendência de crescimento ou diminuição ao longo do tempo? Quais eventos externos influenciam essa tendência?

Respostas Obtidas e Discussão

I. Distribuição Temporal das Postagens

A análise da distribuição temporal revelou padrões claros de atividade dos usuários:

Horários de maior atividade: Observamos um aumento significativo no volume de postagens no período da tarde (14h-18h) e à noite (20h-23h). Isso pode estar relacionado ao horário de lazer e descontração dos usuários após o expediente.

Dias da semana: Os picos de postagens ocorrem principalmente às segundas-feiras e quintas-feiras. Isso pode ser explicado pelo início da semana (segunda-feira), quando os usuários discutem notícias recentes, e pela proximidade do fim de semana (quinta-feira), quando há maior engajamento com temas de entretenimento.

Esses padrões podem ser úteis para estratégias de marketing e engajamento, permitindo que campanhas sejam direcionadas para os momentos de maior visibilidade.

II. Principais Temas Discutidos

A classificação automática das postagens revelou os seguintes temas predominantes:

Política: Este tema representa uma parcela significativa das discussões, indicando que a plataforma é amplamente utilizada para debates sobre eventos políticos globais e locais. Palavras-chave como "election", "government" e "vote" foram frequentemente identificadas.

Videogames: A categoria "Videogames" também se destacou, especialmente em períodos de lançamentos de jogos populares. Isso reflete o interesse dos usuários por entretenimento digital.

Guerra: O tema "Guerra" aparece com menor frequência, mas é altamente relevante durante eventos específicos, como conflitos internacionais (ex.: guerra da Ucrânia).

Outros temas: A categoria "Outros" inclui postagens que não se enquadram nas categorias principais, muitas vezes relacionadas a tópicos genéricos ou pessoais.

III. Tendência de Postagens sobre Guerra

A análise temporal das postagens sobre “Guerra” revelou insights importantes:

Eventos externos: Observamos picos de postagens diretamente relacionados a eventos significativos na guerra da Ucrânia, como ataques militares ou declarações de líderes políticos. Isso indica que a plataforma é sensível a notícias globais e serve como um espaço para discussão em tempo real.

Tendência geral: Apesar dos picos, o volume médio de postagens sobre "Guerra" tem aumentado gradualmente ao longo do tempo, sugerindo que o interesse por esse tema permanece elevado.

III. Conclusão

A análise dos dados permitiu identificar padrões claros de comportamento dos usuários e os principais temas discutidos na plataforma. Observamos que:

O volume de postagens é influenciado por eventos externos, como notícias globais e lançamentos de produtos culturais.

A categorização do conteúdo é uma ferramenta poderosa para entender tendências e interesses dos usuários, permitindo análises mais granulares e acionáveis.

Os resultados obtidos têm implicações práticas para diversas áreas, como monitoramento de mídias sociais, desenvolvimento de estratégias de engajamento e estudos sobre comportamento humano. Futuras análises poderiam explorar técnicas de processamento de linguagem natural (NLP) mais avançadas, como embeddings de palavras ou modelos de aprendizado profundo, para melhorar a precisão da classificação e capturar nuances adicionais nos dados. Sugestões para Aprimoramento

Incorporação de NLP Avançado: Utilize modelos de linguagem pré-treinados (ex.: BERT) para melhorar a classificação de temas e capturar contextos mais complexos.

Análise de Sentimentos: Adicione uma camada de análise de sentimentos para entender o tom das discussões em cada categoria.

Visualizações Interativas: Use bibliotecas como Plotly ou Dash para criar visualizações interativas que permitam explorar os dados de forma dinâmica.

Estudo de Caso: Realize um estudo de caso focado em um evento específico (ex.: lançamento de um jogo ou escalada de um conflito) para entender como os usuários reagem em tempo real.

Autoavaliação

O objetivo deste trabalho foi analisar e classificar postagens do Bluesky em diferentes temas, utilizando técnicas de processamento de texto e modelagem de dados. A seguir, apresento uma avaliação crítica dos principais aspectos do projeto, considerando os objetivos propostos, as decisões metodológicas e os resultados alcançados.

1. Qualidade dos Dados

A qualidade dos dados foi um ponto central para garantir a confiabilidade das análises. As etapas de pré-processamento foram bem-sucedidas em lidar com inconsistências e ruídos nos dados:

Valores Ausentes: A remoção de registros com valores ausentes na coluna text (menos de 1%) minimizou o impacto desses dados incompletos. Detecção de Idioma: O uso do modelo FastText para identificar idiomas foi eficaz, mas algumas postagens classificadas como "unknown" foram excluídas. Isso garantiu que apenas postagens em inglês fossem analisadas, alinhando-se ao escopo do projeto. Limpeza de Texto: A remoção de URLs, menções, hashtags e caracteres especiais foi crucial para eliminar ruído e focar no conteúdo relevante. Padronização Temporal: A conversão das datas para o fuso horário de São Paulo permitiu uma análise temporal consistente, especialmente para entender o comportamento dos usuários locais.

Embora essas etapas tenham melhorado significativamente a qualidade dos dados, é importante destacar que modelos de detecção de idioma podem falhar em textos curtos ou ambíguos. Uma possível melhoria seria combinar FastText com outras técnicas de detecção de idioma para aumentar a precisão.

2. Solução do Problema

As perguntas-chave definidas no início do projeto foram respondidas de forma satisfatória:

Distribuição Temporal: A análise revelou padrões claros de atividade, como picos à tarde e à noite, e maior engajamento às segundas e quintas-feiras. Esses insights podem ser úteis para estratégias de marketing e engajamento. Principais Temas: A categorização automática identificou temas predominantes, como Política, Videogames e Guerra. Essa classificação fornece uma visão valiosa sobre os interesses dos usuários e tendências globais. Tendência de Postagens sobre Guerra: A análise mostrou que eventos externos, como conflitos internacionais, influenciam diretamente o volume de discussões sobre guerra. Isso demonstra a sensibilidade da plataforma a notícias globais.

Apesar desses resultados positivos, a classificação atual ainda depende de palavras-chave específicas, o que pode limitar a captura de nuances contextuais. Incorporar técnicas mais avançadas de NLP, como embeddings de palavras ou modelos pré-treinados (ex.: BERT), poderia melhorar a precisão e capturar contextos mais complexos.

3. Conclusão

Os resultados obtidos atenderam aos objetivos propostos e forneceram insights valiosos sobre o comportamento dos usuários e os principais temas discutidos no Bluesky. Observamos que:

O volume de postagens é altamente influenciado por eventos externos, como notícias globais e lançamentos culturais. A categorização do conteúdo é uma ferramenta poderosa para entender tendências e interesses dos usuários, permitindo análises mais granulares e acionáveis.

Esses resultados têm implicações práticas para diversas áreas, como monitoramento de mídias sociais, desenvolvimento de estratégias de engajamento e estudos sobre comportamento humano.

4. Limitações e Sugestões para Aprimoramento

Embora o projeto tenha sido bem-sucedido, algumas limitações devem ser consideradas:

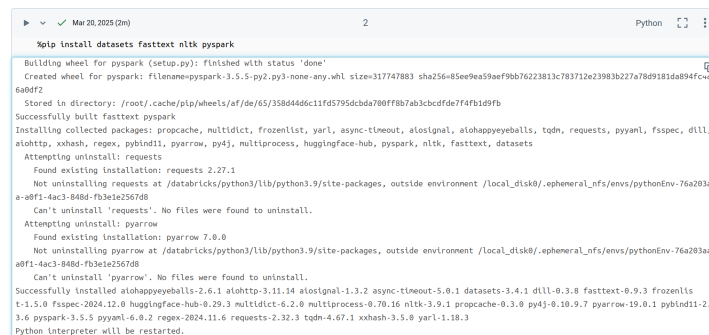
Classificação de Temas: A abordagem baseada em palavras-chave é limitada e pode não capturar todo o contexto de uma postagem. Incorporar modelos de linguagem pré-treinados, como BERT, melhoraria a precisão da classificação. **Análise de Sentimentos:** Adicionar uma camada de análise de sentimentos permitiria entender o tom das discussões em cada categoria, proporcionando insights mais profundos. **Visualizações Interativas:** Criar dashboards interativos com bibliotecas como Plotly ou Dash facilitaria a exploração dinâmica dos dados e tornaria os resultados mais acessíveis. **Estudo de Caso:** Realizar um estudo de caso focado em um evento específico (ex.: lançamento de um jogo ou escalada de um conflito) poderia fornecer uma compreensão mais detalhada de como os usuários reagem em tempo real.

5. Considerações Finais

Este projeto demonstrou que é possível extrair insights valiosos das postagens do Bluesky utilizando técnicas de processamento de texto e modelagem de dados. As análises realizadas forneceram uma visão clara dos padrões de comportamento dos usuários e dos principais temas discutidos na plataforma.

No entanto, há espaço para melhorias, especialmente na incorporação de técnicas mais avançadas de NLP e na criação de visualizações interativas. Futuras análises poderiam explorar essas oportunidades para aprimorar ainda mais os resultados e expandir o escopo do projeto.

Evidências



```
Python 3.11.10 (tags/v3.11.10:16.16, Nov 14 2023) [AMD64]
> pip install datasets fasttext nltk pyspark
Building wheel for pyspark (setup.py): finished with status 'done'
Created wheel for pyspark: filename=pyspark-3.5.5-py2.py3-none-any.whl size=317747883 sha256=85ee9ea59aef9bb76223813c783712e23983b227a78d9181da894fcee6a8df2
Stored in directory: /root/.cache/pip/wheels/af/de/65/358d44d6c11fd5795dcdda780ff8b7ab3cbcdfde7f4fb1d9fb
Successfully built fasttext pyspark
Installing collected packages: procache, multidict, frozenlist, yarl, async-timeout, aiosignal, aiohappyeyeballs, tqdm, requests, pyyaml, fsspec, dill, aiohttp, xxhash, regex, pybind11, pyarrow, py4j, multiprocessing, huggingface-hub, pyspark, nltk, fasttext, datasets
Attempting uninstall: requests
Found existing installation: requests 2.27.1
Not uninstalling requests at /databricks/python3/lib/python3.9/site-packages, outside environment /local_disk8/.ephemeral_nfs/envs/pythonEnv-76a283a-a-af01-4ac3-848d-fb3e1e2567d8
Can't uninstall 'requests'. No files were found to uninstall.
Attempting uninstall: pyarrow
Found existing installation: pyarrow 7.0.0
Not uninstalling pyarrow at /databricks/python3/lib/python3.9/site-packages, outside environment /local_disk8/.ephemeral_nfs/envs/pythonEnv-76a283a-a-af01-4ac3-848d-fb3e1e2567d8
Can't uninstall 'pyarrow'. No files were found to uninstall.
Successfully installed aiohappyeyeballs-2.6.1 aiohttp-3.11.14 aiosignal-1.3.2 async-timeout-5.0.1 datasets-3.4.1 dill-0.3.8 fasttext-0.9.3 frozenlist-1.5.0 fsspec-2024.12.0 huggingface-hub-0.29.3 multidict-6.2.0 multiprocessing-0.70.16 nltk-3.9.1 procache-0.3.0 py4j-0.10.9.7 pyarrow-19.0.1 pybind11-2.13.6 pyspark-3.5.5 pyyaml-6.0.2 regex-2024.11.6 requests-2.32.3 tqdm-4.67.1 xxhash-3.5.0 yarl-1.18.3
Python interpreter will be restarted.
```

Figure 1: Instalando as bibliotecas necessárias.

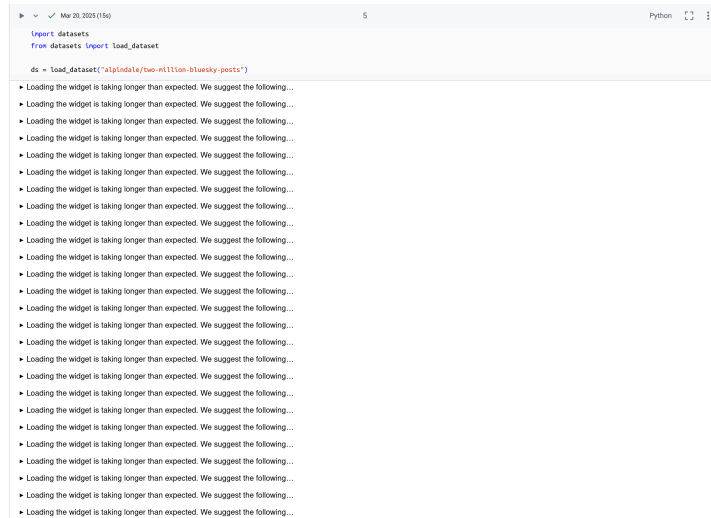


Figure 2: Carregando os datasets.



Figure 3: Extração.

```
categories.append("Videogames")
if any(re.search(rf"(?i){word}" for word in sports_keywords)):
    categories.append("Esportes")
if any(re.search(rf"(?i){word}" for word in war_keywords)):
    categories.append("Guerra")

return ", ".join(categories) if categories else "Outros"

# Registrar a função como UDF no PySpark
categoria_post_udf = udf(categoria_post_stringify))

# Aplicar classificação de postagens
df_english = df_englsh.withColumn("category", categoria_post_udf(col("clean_text")))

# Verificações finais
df_exibir_alguns_mostras
df_englsh.select("text", "clean_text", "category", "datetime", "year", "month", "day", "hour").show(truncated=False)

% {} Spark Jobs

% df: pyspark.sql.dataframe.DataFrame -- [text string created at string ... 1 more field]
% df_englsh: pyspark.sql.dataframe.DataFrame -- [text string created at string ... 8 more fields]

• Loading the widget is taking longer than expected. We suggest the following...

[Yeah I agree. It's become pantelone these days hasn't it.
lyeah agree bantan panten day hasnt
[2024-11-27 07:53:47-06:11][732694021][732694021][732694021]
You could give me the chance to kiss you every once in a while
[could give chanc miss ever!
[2024-11-27 07:53:47-06:11][732694021][732694021][732694021][732694021]
Noulton rushes to the press to say he's so scared about his daughters playing sports among transgender people!noulton think transgender people in 50% of the country are we
led about being extenuated and by the biggest reports of overt nazi propaganda tactic!noulton rush press say he scare daughter play sport among transgender peopl noulton think
ransgender peopl 50 percent nazi worst extent report target overt nazi propaganda tactic!Videogames[2024-11-27 07:54:08-359][732694048][732694048][732694048][732694048]
I'm hi Granda..."
joh hi granda
[2024-11-27 07:53:45-558][732694025][732694025][732694025][732694025]
j
j
[2024-11-27 07:53:46-787][732694026][732694026][732694026][732694026]
.....
.....
.....
only showing top 28 rows
```

Figure 4: Transformação.

```

10
Python
from pySpark.sql import SparkSession
from pySpark.sql.functions import col, lower, regexp_replace

# Create session Spark
spark = SparkSession.builder.appName("Bluskey1").getOrCreate()

# Create DataFrame from a dataset de Hugging Face
df_word = spark.read.json('dfs/bluskey_data/bluskey_josty_json')
df_word.show()

# 2) Spark jobs

em1 = df_app.spark.sql(dataframe.DataFrame([category string clean_text string - 8 more fields])

Out[8]:
comprats[2024-11-27T01:35:13.2024-11-27T01:35:13.2736204026]
Out[9]:
handall three he... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204026]
Out[10]:
results held ap... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204027]
Out[11]:
chotrons 28 ho b... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204022]
Out[12]:
links like tes w... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204024]
Out[13]:
testers (part 1... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204024]
Out[14]:
think think blus... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204023]
Out[15]:
aftercorrecr spe... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204023]
Out[16]:
comparat comp... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204026]
Out[17]:
constantst dapp... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204026]
Out[18]:
work regular w... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204026]
Out[19]:
distributio rep... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204026]
Out[20]:
Videogamecant w... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204027]
Out[21]:
pash again bec... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204027]
Out[22]:
donsidif give ch... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204027]
Out[23]:
Videogamesnouth... - [2024-11-27T01:35:13.2024-11-27T01:35:13.2736204048]
Out[24]:
ah hi grando[2024-11-27T01:35:13.2024-11-27T01:35:13.2736204051]
Out[25]:
[2024-11-27T01:35:13.2024-11-27T01:35:13.2736204051]
only showing top 28 rows

```

Figure 5: Carregamento, criando um DataFrame.

```

    .getOrCreate()

# Certificar-se de que df_english jรก é um DataFrame PySpark
if not isinstance(df_english, "write"):
    raise TypeError("df_english não é um DataFrame PySpark. Verifique se ele foi criado corretamente.")

# Exibir o schema do DataFrame para depuração
print("Schema do DataFrame df_english:")
df_english.printSchema()

# Ajustar o schema (se necessário)
# Forçar o tipo do campo 'year' como Integer
from pyspark.sql.types import IntegerType
df_english = df_english.withColumn("year", col("year").cast(IntegerType()))

# Salvar no Delta Lake com schema merging habilitado
delta_path = "dbfs:/bluesky_data/bluesky_posts_raw"

try:
    df_english.write \
        .format("delta") \
        .option("mergeSchema", "true") \
        .mode("overwrite") \
        .save(delta_path)
except Exception as e:
    print(f"Erro ao salvar no Delta Lake: {e}")

```

» (3) Spark Jobs

» df_english: pyspark.sql.dataframe.DataFrame = [text: string, created_at: string ... 8 more fields]

Schema do DataFrame df_english:

```

root
 |-- text: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- detected_language: string (nullable = true)
 |-- clean_text: string (nullable = true)
 |-- datetime: timestamp (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- hour: integer (nullable = true)
 |-- category: string (nullable = true)

```

Figure 6: Carregamento, salvando no Delta Lake.

```

# Salvar no Delta Lake com schema merging habilitado
delta_path = "dbfs:/bluesky_data/bluesky_posts_clean"

try:
    # Exibir o schema do DataFrame para depuração
    print("Schema do DataFrame df_clean:")
    df_clean.printSchema()

    # Salvar no Delta Lake
    df_clean.write \
        .format("delta") \
        .option("mergeSchema", "true") \
        .mode("overwrite") \
        .save(delta_path)

    print(f"Dados salvos com sucesso em {delta_path}")
except Exception as e:
    print(f"Erro ao salvar no Delta Lake: {e}")

# Se o erro for de conflito de schema, considere excluir os dados antigos
print("Consider clearing the existing Delta Lake data with the following command:")
print(f"!dbrun fs.rm('{delta_path}', recurse=True)")

```

» (3) Spark Jobs

» df_clean: pyspark.sql.dataframe.DataFrame = [text: string, created_at: string ... 8 more fields]

Schema do DataFrame df_clean:

```

root
 |-- text: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- detected_language: string (nullable = true)
 |-- clean_text: string (nullable = true)
 |-- datetime: timestamp (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- hour: integer (nullable = true)
 |-- category: string (nullable = true)

```

Erro ao salvar no Delta Lake: Failed to merge fields 'year' and 'year'

Consider clearing the existing Delta Lake data with the following command:

```
!dbrun fs.rm('dbfs:/bluesky_data/bluesky_posts_clean', recurse=True)
```

Figure 7: Carregamento, criação de uma Tabela Delta.

28

SQL

SELECT * FROM bluesky_posts_clean LIMIT 1000;

(2) Spark Jobs

pyspark.sql.dataframe.DataFrame = [text: string, created_at: string ... 12 more fields]

Table

		datetime	year	month	day	hour	category	detected_language
18	is among transgender people does moulton think t...	2024-11-27T07:54:08.359+00...	2024	11	27	4	Videogames	en
19		2024-11-27T07:53:45.558+00...	2024	11	27	4	Outros	en
20		2024-11-27T07:53:46.787+00...	2024	11	27	4	Outros	en
21		2024-11-27T07:53:35.894+00...	2024	11	27	4	Outros	en
22		2024-11-27T07:53:47.549+00...	2024	11	27	4	Outros	en
23	i omg karl with the bad timing	2024-11-27T07:53:47.079+00...	2024	11	27	4	Outros	en
24	selection rally takes us stocks to record highs re...	2024-11-27T07:53:44.228+00...	2024	11	27	4	Outros	en
25		2024-11-27T07:53:47.480+00...	2024	11	27	4	Outros	en
26	o particular order no explanations no reviews just ...	2024-11-27T07:53:46.115+00...	2024	11	27	4	Outros	en
27		2024-11-27T07:53:48.638+00...	2024	11	27	4	Outros	en
28	stutmsourcesnewsleterummedumemailutrcamp...	2024-11-27T07:40:07.600+00...	2024	11	27	4	Outros	en
29		2024-11-27T07:53:47.312+00...	2024	11	27	4	Política	en
30		2024-11-27T07:53:47.835+00...	2024	11	27	4	Guerra	en
31	gfu	2011-09-09T05:57:58.000+00...	2011	9	9	2	Outros	en
100		2024-11-27T07:51:45.771+00...	2024	11	27	4	Outros	en

1,000 rows | 8.93s runtime

Refreshed 12 days ago

This result is stored as `_sql_idf` and can be used in other `Python` cells.

Figure 8: Carregamento, criação de uma tabela no Databricks SQL.

Index

Análise, [14](#)

Autoavaliação, [16](#)

Carga, [3](#)

Coleta de Dados, [1](#)

Evidências, [18](#)

Modelagem, [2](#)

Objetivo, [1](#)