FECAP

Curso: Análise e Desenvolvimento de Sistemas

PROJETO INTEGRADOR – CANNOLI INTELLIGENCE

Relatório Técnico – Implementação de Inteligência Artificial para Sugestões de Campanhas

Integrantes: Alexandra Christine Silva Raimundo, Carlos Augusto, Hebert Esteves e José Bento.

São Paulo – 2025

Sumário

1. Visão geral da solução de IA

Nesta segunda entrega foi implementada uma camada de Inteligência Artificial totalmente integrada ao banco de dados do sistema, com o objetivo de gerar sugestões automáticas para campanhas de marketing com base no comportamento histórico das campanhas existentes. A IA tem como função principal analisar a tabela `campaign` do MySQL, treinar um modelo de aprendizado de máquina capaz de prever o status mais provável de cada campanha e, a partir dessas previsões, gerar insights práticos — indicando quais campanhas devem ser priorizadas, ajustadas/pausadas ou apenas monitoradas.

2. Código de Machine Learning (Python)

O script responsável por essa automação é o `ia_campanhas_sugestoes.py`. Ele realiza todo o pipeline de treinamento, previsão e gravação de resultados. O modelo utilizado é o RandomForestClassifier, e os resultados alcançaram uma acurácia de 31,6% e um F1-score ponderado de 0.31, conforme o arquivo metrics.json. Mesmo com métricas iniciais modestas, o modelo já demonstra capacidade de identificar padrões relevantes entre campanhas.

```python
import sys
import json
from pathlib import Path

import mysql.connector
import numpy as np
import pandas as pd

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    classification_report,
)

# CONFIGURAÇÃO DO BANCO

DB_CONFIG = {
    "host": "localhost",      # ex: "localhost"
    "user": "root",
    "password": "",
    "database": "cannoli",
}



# CONEXÃO

def get_connection():
    """Abre conexão com MySQL usando DB_CONFIG. Retorna um objeto connection."""
    return mysql.connector.connect(**DB_CONFIG)
```

```python
# CARREGAR CAMPANHAS
Ctrl+Shift+F)

def carregar_campanhas() -> pd.DataFrame:
    """
    Lê a tabela campaign e devolve um DataFrame.

    Premissas (colunas mínimas):
      id, storeId, name, status_desc, badge, type, _mes,
      createdAt, updatedAt, isDefault

    Obs.: Ajustar o SELECT se o esquema divergir.
    """
    conn = get_connection()
    query = """
        SELECT
            id,
            storeId,
            name,
            status_desc,
            badge,
            type,
            _mes,
            createdAt,
            updatedAt,
            isDefault
        FROM campaign
    """
    df = pd.read_sql(query, conn)
    conn.close()

    if df.empty:
        # Falha controlada para evitar treino sem dados
        raise RuntimeError("Nenhuma campanha encontrada na tabela `campaign`.")

    return df
```

```python
# FEATURE ENGINEERING

def adicionar_features(df: pd.DataFrame) -> pd.DataFrame:
    """
    Cria variáveis derivadas para melhorar o poder preditivo.
    Importante: não remove colunas originais (mantém rastreabilidade).
    """
    df = df.copy()

    # Normalização de categorias textuais (reduz nulos e padroniza)
    df["status_desc"] = df["status_desc"].fillna("(sem status)")
    df["badge"] = df["badge"].fillna("(sem badge)")
    df["type"] = df["type"].fillna("(sem tipo)")
    df["_mes"] = df["_mes"].astype(str)
    df["storeId"] = df["storeId"].astype(str)

    # Conversão de datas (erros coercidos para NaT)
    for col in ["createdAt", "updatedAt"]:
        df[col] = pd.to_datetime(df[col], errors="coerce")

    # Medida de "tempo em atividade" como proxy de maturidade
    df["dias_ativos"] = (df["updatedAt"] - df["createdAt"]).dt.days
    df["dias_ativos"] = df["dias_ativos"].fillna(0)

    # Sazonalidade (mês de criação)
    df["mes_criacao_num"] = df["createdAt"].dt.month.fillna(0).astype(int)

    # Proxies simples de complexidade/branding
    df["tam_nome"] = df["name"].fillna("").astype(str).str.len()
    df["tem_badge"] = np.where(df["badge"] == "(sem badge)", 0, 1)

    # Normalização para inteiro
    df["isDefault"] = df["isDefault"].fillna(0).astype(int)

    return df
```

```python
# TREINAR MODELO
def treinar_modelo(df: pd.DataFrame):
    """
    Treina RandomForest para prever status_desc.

    Retorna:
      model: classificador treinado
      encoders: dicionário com LabelEncoders (features categóricas + alvo)
      df_feat: DataFrame com features + colunas auxiliares (__id, __name, __storeId_raw)
      feature_cols: lista de colunas usadas como X
      metrics: dicionário de métricas (para persistência em JSON)
    """
    df = df.copy()

    # Garantia de alvo presente (evita label encoder vazio)
    df = df[df["status_desc"].notna()].reset_index(drop=True)
    if df.empty:
        raise RuntimeError("Não há status_desc válidos para treinar o modelo.")

    # Engenharia de atributos
    df = adicionar_features(df)

    # Guardar identificadores para pós-predição
    ids = df["id"].astype(int)
    nomes = df["name"].fillna("").astype(str)
    store_ids_raw = df["storeId"].astype(str)

    # Codificação de categorias (LabelEncoder por coluna)
    cat_cols = ["storeId", "badge", "type", "_mes"]
    encoders = {}

    for col in cat_cols:
        le = LabelEncoder()
        df[col] = df[col].astype(str)
        df[col] = le.fit_transform(df[col])
        encoders[col] = le
```

```python
# Alvo
target_col = "status_desc"
y_text = df[target_col].astype(str)

enc_status = LabelEncoder()
y = enc_status.fit_transform(y_text)
encoders[target_col] = enc_status

# Seleção de variáveis preditoras (X)
feature_cols = cat_cols + [
    "dias_ativos",
    "mes_criacao_num",
    "tam_nome",
    "tem_badge",
    "isDefault",
]
X = df[feature_cols]

# Divisão estratificada (melhor representação das classes no teste)
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.25,
    random_state=42,
    stratify=y,
)

# Hiperparâmetros conservadores para evitar overfitting inicial
model = RandomForestClassifier(
    n_estimators=300,
    max_depth=8,
    class_weight="balanced",
    random_state=42,
    n_jobs=-1,
)
```

```python
model.fit(X_train, y_train)

# Avaliação objetiva (acurácia + F1 ponderado)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1w = f1_score(y_test, y_pred, average="weighted")

# Relatórios para auditoria de desempenho
report_dict = classification_report(
    y_test,
    y_pred,
    target_names=enc_status.classes_,
    output_dict=True,
    zero_division=0,
)
report_text = classification_report(
    y_test,
    y_pred,
    target_names=enc_status.classes_,
    zero_division=0,
)

print("\n===== MÉTRICAS DO MODELO =====")
print("Acurácia:", round(acc, 3))
print("F1 (weighted):", round(f1w, 3))
print("\nRelatório por classe:")
print(report_text)

metrics = {
    "accuracy": float(acc),
    "f1_weighted": float(f1w),
    "classes": list(enc_status.classes_),
    "classification_report": report_dict,
    "classification_report_text": report_text,
    "n_samples_total": int(len(df)),
    "n_samples_train": int(len(X_train)),
```

```
        "n_samples_test": int(len(X_test)),
    }

    # df_feat mantém contexto para geração de sugestões
    df_feat = df.copy()
    df_feat["__id"] = ids
    df_feat["__name"] = nomes
    df_feat["__storeId_raw"] = store_ids_raw

    return model, encoders, df_feat, feature_cols, metrics


# GERAR SUGESTÕES
def gerar_sugestoes(df_feat: pd.DataFrame, feature_cols, model, encoders):
    """
    Produz recomendações por campanha com base nas probabilidades do modelo.

    Saída (lista de dicts):
      campaignId, storeId, name, status_previsto, confianca, grupo
    """
    X_full = df_feat[feature_cols]
    probs = model.predict_proba(X_full)
    y_pred = model.predict(X_full)

    enc_status = encoders["status_desc"]
    classes = enc_status.classes_

    sugestoes = []

    for i, row in df_feat.iterrows():
        camp_id = int(row["__id"])
        store_id = str(row["__storeId_raw"])
        nome_campanha = str(row["__name"])

        idx_classe = y_pred[i]
        status_previsto = classes[idx_classe]
```

```
        conf = float(probs[i, idx_classe])

        # Heurística simples de agrupamento:
        # - priorizar: status positivo com alta confiança
        # - ajustar_ou_pausar: status inicial/rascunho com baixa confiança
        # - monitorar: demais casos
        status_lower = status_previsto.lower()

        if (("conclu" in status_lower) or ("ativ" in status_lower)) and conf >= 0.6:
            grupo = "priorizar"
        elif (("rascunho" in status_lower) or ("agend" in status_lower)) and conf <=
            grupo = "ajustar_ou_pausar"
        else:
            grupo = "monitorar"

        sugestoes.append(
            {
                "campaignId": camp_id,
                "storeId": store_id,
                "name": nome_campanha,
                "status_previsto": status_previsto,
                "confianca": conf,
                "grupo": grupo,
            }
        )

    print(f"\nGeradas {len(sugestoes)} sugestões.")
    return sugestoes


# SALVAR SUGESTÕES NA TABELA
def salvar_sugestoes_no_banco(sugestoes, modelo_versao="rf_v1"):
    """
    Persiste sugestões em `campaign_ai_sugestoes`.

    Nota: TRUNCATE remove histórico. Se for necessário manter versões,
```

```
    comentar o TRUNCATE e incluir carimbo de tempo/versão.
    """
    if not sugestoes:
        print("Nenhuma sugestão para salvar no banco.")
        return

    conn = get_connection()
    cur = conn.cursor()

    # Atenção: limpa a tabela inteira antes de inserir
    cur.execute("TRUNCATE TABLE campaign_ai_sugestoes")

    insert_sql = """
        INSERT INTO campaign_ai_sugestoes
          (campaignId, storeId, status_previsto, confianca, grupo, modelo_versao)
        VALUES (%s, %s, %s, %s, %s, %s)
    """

    data = [
        (
            s["campaignId"],
            s["storeId"],
            s["status_previsto"],
            round(s["confianca"], 4),
            s["grupo"],
            modelo_versao,
        )
        for s in sugestoes
    ]

    cur.executemany(insert_sql, data)
    conn.commit()
    cur.close()
    conn.close()

    print(f"Inseridas {len(sugestoes)} linhas em `campaign_ai_sugestoes`.")
```

```python
# SALVAR JSONS (metrics.json e sugestoes.json)
def salvar_jsons(metrics: dict, sugestoes: list):
    """
    Salva artefatos de auditoria e consumo downstream:
      - metrics.json: desempenho do modelo
      - sugestoes.json: recomendações geradas
    """
    base_dir = Path(".").resolve()

    metrics_path = base_dir / "metrics.json"
    sugestoes_path = base_dir / "sugestoes.json"

    with metrics_path.open("w", encoding="utf-8") as f:
        json.dump(metrics, f, ensure_ascii=False, indent=2)

    with sugestoes_path.open("w", encoding="utf-8") as f:
        json.dump(sugestoes, f, ensure_ascii=False, indent=2)

    print(f"Métricas salvas em: {metrics_path}")
    print(f"Sugestões salvas em: {sugestoes_path}")


# MAIN
def main():
    """Pipeline orquestrado: carrega dados -> treina -> sugere -> persiste -> salva artefatos."""
    print("Carregando campanhas do banco...")
    df_raw = carregar_campanhas()

    print("Treinando modelo...")
    model, encoders, df_feat, feature_cols, metrics = treinar_modelo(df_raw)

    print("Gerando sugestões para todas as campanhas...")
    sugestoes = gerar_sugestoes(df_feat, feature_cols, model, encoders)

    print("Salvando sugestões no MySQL...")
    salvar_sugestoes_no_banco(sugestoes, modelo_versao="rf_v1")
```

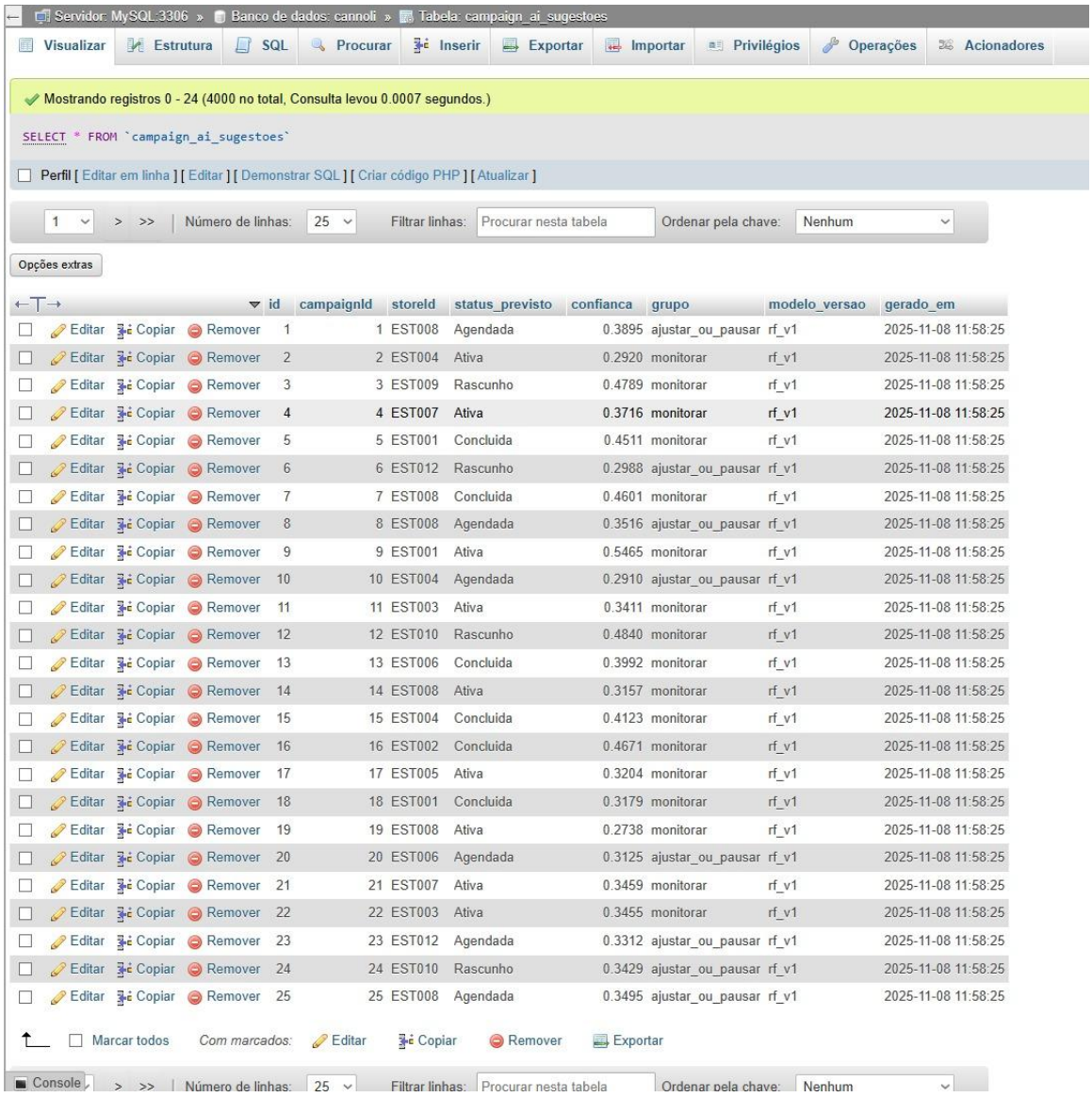
```python
    print("Salvando JSONs (metrics.json e sugestoes.json)...")
    salvar_jsons(metrics, sugestoes)

    print("\n[OK] Processo concluído.")


if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        # Tratamento simples para falhas operacionais: log e saída com erro
        print(f"[ERRO] {e}")
        sys.exit(1)
```

```
[
  {
    "campaignId": 1,
    "storeId": "EST008",
    "name": "Campanha Saepe WGN5",
    "status_previsto": "Agendada",
    "confianca": 0.38947318897560673,
    "grupo": "ajustar_ou_pausar"
  },
  {
    "campaignId": 2,
    "storeId": "EST004",
    "name": "Campanha Voluptatem H6QI",
    "status_previsto": "Ativa",
    "confianca": 0.2920366508608301,
    "grupo": "monitorar"
  },
  {
    "campaignId": 3,
    "storeId": "EST009",
    "name": "Campanha Itaque MQPN",
    "status_previsto": "Rascunho",
    "confianca": 0.4789220292055393,
    "grupo": "monitorar"
  },
  {
    "campaignId": 4,
    "storeId": "EST007",
    "name": "Campanha Voluptate DCBH",
    "status_previsto": "Ativa",
    "confianca": 0.3715988371334461,
    "grupo": "monitorar"
  },
```

```
{
  "accuracy": 0.316,
  "f1_weighted": 0.31366305134908495,
  "classes": [
    "Agendada",
    "Ativa",
    "Concluida",
    "Rascunho"
  ],
  "classification_report": {
    "Agendada": {
      "precision": 0.22608695652173913,
      "recall": 0.3291139240506329,
      "f1-score": 0.26804123711340205,
      "support": 79.0
    },
    "Ativa": {
      "precision": 0.3798076923076923,
      "recall": 0.4114583333333333,
      "f1-score": 0.395,
      "support": 192.0
    },
    "Concluida": {
      "precision": 0.3387096774193548,
      "recall": 0.2608695652173913,
      "f1-score": 0.29473684210526313,
      "support": 161.0
    },
    "Rascunho": {
      "precision": 0.20754716981132076,
      "recall": 0.16176470588235295,
      "f1-score": 0.18181818181818182,
      "support": 68.0
    },
```

3. Persistência dos resultados no banco

Após o treinamento e as previsões, os resultados são gravados na tabela `campaign_ai_sugestoes`. Cada execução limpa os dados antigos e insere novos registros com as colunas campaignId, storeId, status_previsto, confianca, grupo e modelo_versao. Essas informações são utilizadas posteriormente nos dashboards administrativos.



Servidor: MySQL:3306 » Banco de dados: cannoli » Tabela: campaign_ai_sugestoes

Visualizar | Estrutura | SQL | Procurar | Inserir | Exportar | Importar | Privilégios | Operações | Acionadores

Mostrando registros 0 - 24 (4000 no total, Consulta levou 0.0007 segundos.)

```
SELECT * FROM `campaign_ai_sugestoes`
```

Perfil [Editar em linha] [Editar] [Demonstrar SQL] [Criar código PHP] [Atualizar]

1 > >> | Número de linhas: 25 | Filtrar linhas: Procurar nesta tabela | Ordenar pela chave: Nenhum

Opções extras

	id	campaignId	storeId	status_previsto	confianca	grupo	modelo_versao	gerado_em
Editar Copiar Remover	1	1	EST008	Agendada	0.3895	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	2	2	EST004	Ativa	0.2920	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	3	3	EST009	Rascunho	0.4789	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	4	4	EST007	Ativa	0.3716	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	5	5	EST001	Concluida	0.4511	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	6	6	EST012	Rascunho	0.2988	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	7	7	EST008	Concluida	0.4601	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	8	8	EST008	Agendada	0.3516	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	9	9	EST001	Ativa	0.5465	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	10	10	EST004	Agendada	0.2910	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	11	11	EST003	Ativa	0.3411	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	12	12	EST010	Rascunho	0.4840	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	13	13	EST006	Concluida	0.3992	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	14	14	EST008	Ativa	0.3157	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	15	15	EST004	Concluida	0.4123	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	16	16	EST002	Concluida	0.4671	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	17	17	EST005	Ativa	0.3204	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	18	18	EST001	Concluida	0.3179	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	19	19	EST008	Ativa	0.2738	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	20	20	EST006	Agendada	0.3125	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	21	21	EST007	Ativa	0.3459	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	22	22	EST003	Ativa	0.3455	monitorar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	23	23	EST012	Agendada	0.3312	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	24	24	EST010	Rascunho	0.3429	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25
Editar Copiar Remover	25	25	EST008	Agendada	0.3495	ajustar_ou_pausar	rf_v1	2025-11-08 11:58:25

Marcar todos | Com marcados: Editar | Copiar | Remover | Exportar

Console | > >> | Número de linhas: 25 | Filtrar linhas: Procurar nesta tabela | Ordenar pela chave: Nenhum

4. Integração com o backend Node.js

O backend Node.js executa o script Python por meio de uma rota específica (`POST /api/executar-ia`). Essa integração permite que o administrador atualize as sugestões diretamente pelo sistema, mantendo a comunicação entre as camadas Python e Node. As rotas adicionais permitem que as previsões sejam lidas e exibidas no frontend de forma organizada.

5. Exibição das sugestões no frontend

As sugestões geradas são exibidas tanto para o administrador quanto para os estabelecimentos. No painel do administrador, é apresentada uma visão consolidada das campanhas com destaque para aquelas com status positivo e alta confiança. No painel dos estabelecimentos, as recomendações são personalizadas conforme o desempenho das campanhas individuais.

6. Conclusão

Com esta entrega, o sistema Cannoli Intelligence passa a contar com um módulo de IA funcional, que conecta dados, aprendizado de máquina e experiência do usuário. A integração entre as camadas de banco, backend e frontend garante um fluxo contínuo de informações. Mesmo em sua primeira versão, o modelo RandomForest já provou ser eficiente para recomendações automatizadas, e abre espaço para ajustes e aprimoramentos futuros.

7. Referências

• Scikit-learn Documentation. RandomForestClassifier. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
• Pandas Documentation. Disponível em: https://pandas.pydata.org/docs/