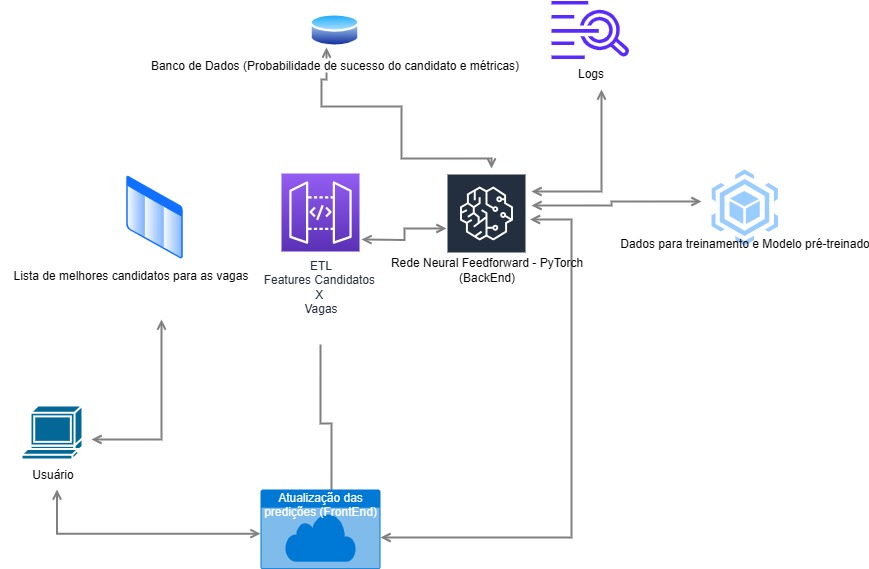
# Projeto: Sistema de Matching com IA

**Projeto:** TC-FIAP - Sistema de Matching em Seleção

**Aluno:** Eric Lopes Mello

**Repositório GIT:** <https://github.com/ericlmello/selecao_rh_com_ia>



## 1. Visão Geral e Objetivo

O **Sistema de Matches Automáticos com IA** é uma aplicação de Machine Learning projetada para revolucionar o processo de recrutamento. A solução utiliza redes neurais para calcular a probabilidade de sucesso entre candidatos e vagas de emprego, fornecendo recomendações automáticas e precisas baseadas em análise semântica de textos.

**Objetivo Principal:** Automatizar e otimizar o processo de matching, fornecendo pontuações de compatibilidade para auxiliar as equipes de recrutamento a tomar decisões mais rápidas e informadas, identificando os melhores talentos de forma eficiente.

## 2. Arquitetura Técnica

A solução foi construída sobre uma base de tecnologias modernas e uma estrutura de arquivos organizada para garantir performance, escalabilidade e manutenibilidade.

### 2.1. Estrutura de Arquivos do Projeto

ml\_ats/

├── app.py # Aplicação Flask principal

├── model.py # (Opcional) Classe do modelo PyTorch

├── data\_processing.py # (Opcional) Scripts de pré-processamento

├── train.py # (Opcional) Script de treinamento

├── config.py # Configurações do projeto

├── requirements.txt # Dependências do Python

├── static/ # Arquivos estáticos (CSS, JS) para a interface

├── templates/ # Templates HTML (index.html)

└── data/ # Dados (jobs.csv, prospects.csv, applicants.csv)

### 2.2. Tecnologias Utilizadas

* **Linguagem:** Python 3
* **Framework Web (API):** Flask
* **Machine Learning:** PyTorch (Rede Neural), Scikit-learn (Pré-processamento), Pandas e NumPy (Manipulação de Dados).
* **Monitoramento de Modelos:** MLflow
* **Banco de Dados:** SQLite

### 2.3. Arquitetura do Modelo de Machine Learning

* **Tipo de Modelo:** Rede Neural Feedforward (implementada com PyTorch).
* **Entrada:** Um vetor de **201 features**, composto por:
  + **100 features** extraídas do currículo do candidato (frequência - via TF-IDF).
  + **100 features** extraídas da descrição da vaga (via TF-IDF).
  + **1 feature extra** (ex: tamanho do currículo).
* **Saída:** Uma única probabilidade entre 0 e 1 (0% a 100%) de sucesso.
* **Ativação Final:** Sigmoid, para garantir que a saída seja uma probabilidade válida.

## 3. Pipeline de Processamento de Dados

O tratamento dos dados é uma etapa crucial e segue um pipeline robusto:

1. **Limpeza de Dados:** Textos de currículos e vagas passam por uma limpeza rigorosa para remover caracteres inválidos ("surrogates"), tratar valores nulos e padronizar o conteúdo.
2. **Extração de Features (TF-IDF):** A técnica TF-IDF (Term Frequency-Inverse Document Frequency) é utilizada para converter os textos em vetores numéricos. Ela captura a importância de cada palavra, permitindo que o modelo entenda o conteúdo semanticamente.
3. **Composição do Vetor Final:** Os vetores do CV, da vaga e as features extras são combinados em um único vetor de 201 dimensões.
4. **Normalização:** Os valores do vetor final são normalizados (usando MinMaxScaler) para ficarem em uma escala consistente (geralmente entre 0 e 1), o que otimiza o treinamento da rede neural.
5. **Conversão para Tensor:** O vetor normalizado é convertido para um Tensor PyTorch, o formato de entrada esperado pelo modelo.

***3.1 O Processo de Tratamento de Dados***

Para que o nosso sistema de IA funcione de forma rápida e eficiente, os dados brutos precisam de ser pré-processados. Este tratamento é feito uma única vez através de um script especializado chamado create\_sample.py.

A fonte dos dados são:

**aplicants:**  
https://drive.google.com/file/d/1Z0dOk8FMjazQo03PuUeNGZOW-rxtpzmO/view?usp=sharing

**prospects:**  
https://drive.google.com/file/d/17RkgTlckZ6ItDqgsDCT8H5HZn\_OwmxQO/view?usp=sharing

**vagas:**  
https://drive.google.com/file/d/1h8Lk5LM8VE5TF80mngCcbsQ14qA2rbw\_/view?usp=sharing

*É importante notar que este método de pré-processamento e amostragem foi adotado como uma solução estratégica para as limitações de ambiente. Plataformas como o GitHub têm restrições quanto ao tamanho dos arquivos, e o ambiente de produção no Render (plano gratuito) tem um limite de memória RAM. Ao processar e reduzir os dados, garantimos que a aplicação permanece leve, rápida e económica, sem comprometer a sua capacidade de retreinar o modelo de Machine Learning com dados relevantes.*

### O Que o Script Faz (Função a Função)

O script automatiza todo o trabalho pesado, seguindo uma lógica clara:

**1.** carregar\_json\_bruto(caminho)

* **O que faz:** Esta função é o ponto de partida. Ela simplesmente lê um dos seus ficheiros JSON originais, que têm uma estrutura "aninhada" (dados dentro de dados), e carrega-o para a memória.

**2.** processar\_jobs()**,** processar\_prospects()**,** processar\_applicants()

* **O que fazem:** Este é o coração do tratamento. Cada uma destas funções recebe os dados brutos e aninhados e os transforma numa tabela simples e "achatada" (um DataFrame do pandas). Elas sabem exatamente onde encontrar cada informação importante e organizam tudo em colunas, como numa folha de cálculo.
  + processar\_jobs(data)**:** Esta função navega pela estrutura complexa do JSON de vagas. Para cada vaga, ela entra nos objetos informacoes\_basicas e perfil\_vaga para extrair campos essenciais como titulo\_vaga, cliente e cidade, criando uma linha limpa e organizada para cada vaga.
  + processar\_prospects(data)**:** Esta função lida com a relação entre vagas e candidatos. Ela percorre cada vaga, encontra a lista aninhada chamada prospects e cria uma linha separada para cada candidato associado a essa vaga, garantindo que a ligação vaga\_id -> codigo do candidato seja mantida.
  + processar\_applicants(data)**:** Esta é a função de tratamento mais detalhada. Para cada candidato, ela navega por múltiplas secções aninhadas (infos\_basicas, informacoes\_pessoais, informacoes\_profissionais, etc.) para recolher todos os detalhes, desde o nome e email até à experiência e habilidades, consolidando tudo numa única linha de tabela.

**3. Bloco Principal de Execução (**if \_\_name\_\_ == "\_\_main\_\_":**)**

* **O que faz:** Orquestra todo o processo e adiciona as verificações finais:
  + ***Processamento:*** *Usa as funções acima para converter os ficheiros JSON brutos em tabelas de dados limpas.*
  + ***Verificação de Nulos:*** *Após o tratamento, é importante verificar cada tabela em busca de dados em falta (nulos) e exibe uma contagem. Isto garante a qualidade e a integridade dos dados antes de prosseguir.*
  + ***Resumo Estatístico:*** *Podemos executar a função describe() offline em cada tabela de dados. Isto gera um resumo estatístico completo (contagem, média, desvio padrão, mínimo, máximo, etc.) para as colunas numéricas, oferecendo uma visão geral e quantitativa dos dados processados.*
  + ***Criação da Amostra:*** *Por fim, ele cria e salva a amostra final otimizada que será usada pela aplicação*

## 4. Funcionalidades e Endpoints da API

O sistema é exposto através de uma API RESTful. Abaixo estão os principais endpoints e exemplos de como usá-los com curl.

|  |  |  |
| --- | --- | --- |
| Rota | Método | Descrição |
| / | GET | Exibe a interface principal da aplicação. |
| /predict | POST | **(Principal)** Recebe um JSON com job\_id e candidate\_id e retorna a predição de compatibilidade. |
| /evaluate\_model | GET | Executa uma avaliação do modelo atual com dados históricos e retorna as métricas de performance. |
| /trigger\_retraining | POST | Inicia o processo de retreino de um novo modelo. Pode receber hiperparâmetros em um corpo JSON. |
| /health | GET | Retorna um status 200 OK e um JSON com a saúde de cada componente da aplicação. |
| /jobs | GET | Retorna uma lista das vagas disponíveis no sistema. |
| /candidates | GET | Retorna uma lista dos candidatos disponíveis. |
| /matches | GET | Retorna até 1000 combinações válidas de vaga-candidato. |

#### Exemplo de Uso da API (curl)

# Realizar uma predição

curl -X POST -H "Content-Type: application/json" -d '{"job\_id": "4534", "candidate\_id": "11132"}' http://127.0.0.1:5000/predict

# Listar vagas disponíveis

curl http://127.0.0.1:5000/jobs

# Acionar o retreino do modelo com hiperparâmetros personalizados

curl -X POST -H "Content-Type: application/json" -d '{"epochs": 10, "learning\_rate": 0.005}' http://127.0.0.1:5000/trigger\_retraining

## 5. Monitoramento, Métricas e Logs

Monitorar a performance do modelo é fundamental. O sistema oferece múltiplas formas de fazer isso.

### 5.1. Logs Detalhados em Tempo Real

Para cada predição, o sistema gera logs detalhados no terminal, permitindo um acompanhamento granular:

NOVA PREDIÇÃO INICIADA - 14:30:25

Recebido: job\_id=5181, candidate\_id=31006

Predição concluída: probabilidade=0.7234, tempo=0.156s.

### 5.2. Armazenamento Histórico (Banco de Dados SQLite)

* **Registro de Predições:** Cada predição é salva no banco de dados, criando um histórico auditável com timestamp, IDs, probabilidade e tempo de resposta.
* **Relatórios de Avaliação:** Os resultados da rota /evaluate\_model também são salvos, permitindo acompanhar a evolução da performance do modelo ao longo do tempo.

### 5.3. Monitoramento Avançado com MLflow

O sistema é integrado com o MLflow para um monitoramento visual e profissional.

**Como usar:**

1. **Inicie a Interface do MLflow:** No terminal, execute mlflow ui.
2. **Acesse no Navegador:** Abra http://127.0.0.1:5001 (ou a porta indicada).
3. **Explore:** Na interface, é possível visualizar cada execução (run) do modelo, comparar métricas de performance (acurácia, F1-score), analisar os parâmetros usados e visualizar os modelos salvos como "artefatos".

### 5.4. Métricas de Avaliação do Modelo

O sistema calcula as seguintes métricas de performance através da rota /evaluate\_model:

* **Acurácia:** Percentual de predições corretas.
* **Precisão:** Dos recomendados como "bons", quantos realmente eram.
* **Recall:** Dos que eram realmente "bons", quantos o modelo conseguiu encontrar.
* **F1-Score:** Média harmônica entre precisão e recall, uma métrica de balanço geral.

O limite (MINIMUM\_F1\_SCORE\_THRESHOLD) funciona como um "portão de qualidade" **após** um novo modelo já ter sido treinado. Se o limite não for atingido, um novo modelo é treinado. A performance desse novo modelo é medida. Se a performance for **inferior** ao seu limite, o sistema **rejeita** o novo modelo e **mantém o modelo antigo** em produção.

## 6. Garantia de Qualidade e Testes

Para garantir a estabilidade e a confiabilidade, foi implementada uma suíte de testes automatizados com **Pytest**.

* **Ambiente de Teste Isolado:** Antes da execução, um ambiente limpo é criado com dados fictícios para garantir que os testes sejam consistentes e não afetem os dados reais.
* **Testes de Integração:** Validam se os componentes principais (API, banco de dados, modelo) funcionam corretamente em conjunto. Testam as rotas /health, /predict (sucesso e falha) e /jobs.
* **Testes Unitários:** Focam na lógica de funções críticas, como a calculate\_metrics, para garantir que os cálculos matemáticos estejam corretos em diferentes cenários (pontuação perfeita, mista e casos extremos).

**Como Executar:** A suíte de testes pode ser executada com um único comando:

pytest -v

## 7. Conclusão e Próximos Passos

O Sistema de Recomendação de Candidatos é uma aplicação robusta, inteligente e confiável, pronta para ser integrada ao fluxo de trabalho de recrutamento. Sua arquitetura modular e suíte de testes completa garantem não apenas o funcionamento atual, mas também a facilidade de manutenção e evolução futura.

**Possíveis Evoluções:**

* Criar uma interface gráfica para visualizar as métricas e a distribuição das predições em tempo real.
* **Validação Cruzada:** Implementar um endpoint para executar k-fold cross-validation, fornecendo uma estimativa mais robusta da performance do modelo.
* Integrar o projeto a sistemas de CI/CD (como GitHub Actions) para automatizar a execução de testes a cada nova alteração no código ou entrada de dados de candidatos e vagas.
* Modelagem de dados da entrevista do candidato, quando disponíveis
* Uso de plataforma de hospedagem/deploy mais robusta