Sistema de Recomendação de Comida

Etapa 1: Proposta

A Fase de planejamento constitui um período crucial no ciclo de desenvolvimento, sendo o momento onde são estabelecidas e definidas todas as bases fundamentais do projeto. Durante esta etapa inicial, a equipe identifica os requisitos essenciais, determina o escopo e define os parâmetros que guiarão todo o processo subsequente de implementação.

Problema e Objetivo

Problema: Usuários de aplicativos de entrega de comida acabam sobrecarregados pela grande quantidade de opções disponíveis, o que faz com que percam tempo procurando algo que gostem, resultando em uma experiência ruim de usuário.

Objetivo: Desenvolver um modelo de Machine Learning que forneça recomendações personalizadas de pratos e lanches de restaurantes para os usuários, com o objetivo de aumentar o engajamento e a probabilidade de finalizar um pedido. O modelo irá prever uma nota que o usuário daria para um item que ele ainda não experimentou.

Dataset(Fonte de dados)

É necessário obter um dataset público com dados suficientes para trabalhar adequadamente. Como não é possível conseguir um dataset do iFood, pesquisei e identifiquei alguns conjuntos de dados potencialmente úteis para nosso projeto.

Alternativa: Food.com Recipes and Interactions

- **Link:** `https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions/data`
- **Justificativa:** Este dataset é extenso e completo, contendo dados de restaurantes e informações detalhadas de usuários. Por sua complexidade e riqueza de informações, representa nossa primeira opção.

Alternativa: Yelp Dataset

- **Link:** `https://www.yelp.com/dataset`
- **Justificativa:** Este dataset é extenso e contém dados detalhados de negócios (restaurantes), avaliações e informações dos usuários. Embora seja mais complexo, oferece uma rica fonte de features para extração. É possível filtrar especificamente para a categoria "restaurantes".

Devido à facilidade de acesso e adequação dos tipos de dados, escolhemos o dataset **Food.com Recipes and Interactions**. O fato de estar disponível na plataforma Kaggle também facilitou seu uso.

Métrica Principal de Avaliação

Como o objetivo é prever uma nota numérica de 1 a 5, estamos diante de um problema de REGRESSÃO.

Métrica Principal:

- **RMSE:** É a métrica mais comum para este tipo de problema. Ela penaliza erros grandes com mais intensidade. O objetivo é minimizá-la. Em termos práticos, ela nos mostra, em média, o quão distante (em estrelas) nossa previsão está da realidade.

Riscos Potênciais:

- **Dados Iniciais:** O modelo não saberá recomendar para um novo usuário (sem histórico de avaliações) ou como recomendar um novo item (que nunca foi avaliado).
- **Desbalanceamento de Dados:** Frequentemente, os usuários avaliam itens que amaram ou odiaram. Isso resulta em uma concentração de notas nos extremos (como 1 e 5) e uma escassez de avaliações intermediárias.

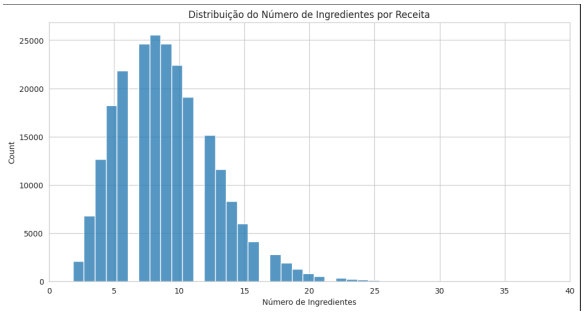
Etapa 2: Dados(Análise e Pré-Processamento)

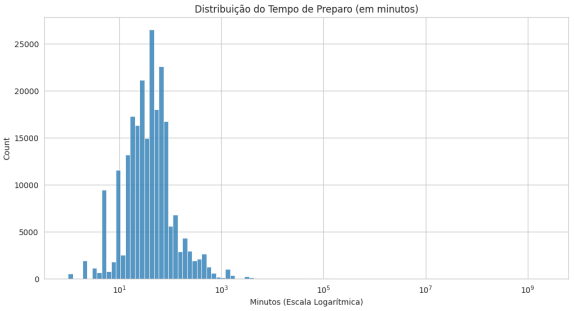
Para começar a parte de pré-processamento dos dados, primeiro precisamos começar com uma analise exploratória dos dados. para ver como os dados estão estruturados e tentar achar as nossas colunas chave.

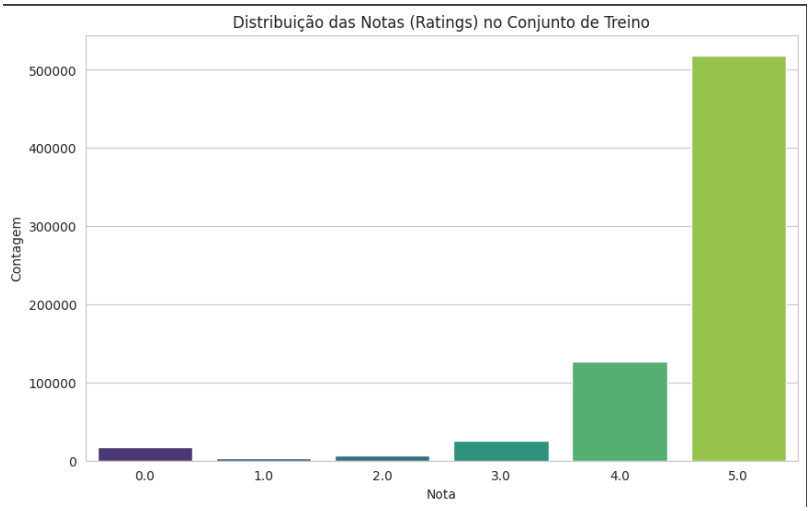
Análise Exploratória

Fazendo uma primeira análise, a gente vê que a maioria das avaliações é nota 5. Isso mostra que os usuários tendem a avaliar as receitas que gostaram muito e quase não há notas de 1 a 3 estrelas, isso pode acabar se tornando um problema.

Rodando mais um pouco a gente acaba achando muitas informações úteis, como o `rating`, tempo de preparo `minutes`, tags `tags`, numero de passos `n_steps` e ingredientes `ingredients`. Também conseguimos observar que o tempo de preparo é enviesada para a direita(muitas receitas rapidas, poucas demoradas) e que a maioria das receitas tem entre 5 e 15 ingredientes.







Pré-Processamento e Criação de Features

Criando média e número de avaliação de usuários e receitas

Para começar vamos calcular o rating médio e o número de avaliações para cada receita e também para cada usuário.

```
print("Calculando features de receitas...")
media_rating_por_item = df_train.groupby('recipe_id')['rating'].mean().rename('media_rating_item')
num_avaliacoes_por_item = df_train.groupby('recipe_id')['rating'].count().rename('num_avaliacoes_item')

df_receitas = df_receitas.merge(media_rating_por_item, left_on='id', right_index=True, how='left')
df_receitas = df_receitas.merge(num_avaliacoes_por_item, left_on='id', right_index=True, how='left')

print("Calculando features de usuários...")
media_rating_por_usuario = df_train.groupby('user_id')['rating'].mean().rename('media_rating_usuario')
num_avaliacoes_por_usuario = df_train.groupby('user_id')['rating'].count().rename('num_avaliacoes_usuario')
df_users = pd.DataFrame(media_rating_por_usuario).join(num_avaliacoes_por_usuario)
```

Criando Features para conseguir a “idade” da receita

Depois de criar essas features, principalmente a que calcula a média eu pensei em adicionar mais algumas para nos ajudar e para deixar o modelo mais completo, coluna `submitted` (data de publicação da receita) usa-la para pegar a “idade” da receita, porque receitas mais antigas podem ser clássicos bem avaliados ou pode ser algo ruim também ou também receitas podem ser mais populares em certa época do ano.

```
print("Criando features temporais...")
df_receitas['submitted'] = pd.to_datetime(df_receitas['submitted'])
df_receitas['ano_publicacao'] = df_receitas['submitted'].dt.year
df_receitas['mes_publicacao'] = df_receitas['submitted'].dt.month
data_recente = df_receitas['submitted'].max()
df_receitas['idade_receita_dias'] = (data_recente - df_receitas['submitted']).dt.days
```

Criação de Features de texto atraves das Tags

As tags descrevem características específicas das receitas (ex: '30-minutes-or-less', 'desserts', 'healthy'). Nossa estratégia é selecionar as tags mais populares e convertê-las em colunas binárias (one-hot encoding). Isso porque as preferências dos usuários geralmente estão relacionadas às categorias das receitas – um usuário que gosta de sobremesas, por exemplo, provavelmente avaliará bem outras sobremesas.

```
print("Criando features de tags...")
df_receitas['tags_limpas'] = df_receitas['tags'].apply(lambda x: ' '.join(ast.literal_eval(x)))
vectorizer = CountVectorizer(max_features=30, binary=True)
tags_matrix = vectorizer.fit_transform(df_receitas['tags_limpas'])
df_tags = pd.DataFrame(tags_matrix.toarray(), index=df_receitas.index, columns=['tag_' +
tag for tag in vectorizer.get_feature_names_out()])
df_receitas = pd.concat([df_receitas, df_tags], axis=1)
```

Tratando a Feature de Nutrição

A tabela contém uma coluna chamada `nutrition` que precisamos transformar. Se a mantivermos como está, o modelo a interpretará como um texto único, essa coluna hoje está assim:

recipe_id	nutrition
1	'[487.8, 16.0, 283.0, 13.0, 52.0]'
2	'[250.0, 8.5, 150.0, 9.0, 30.0]'
3	NaN (vazio)

Este processo é basicamente uma tradução: vamos transformar os dados originais, que o modelo não consegue interpretar, em valores numéricos que ele pode entender. Isso vai reestruturar a coluna "nutrition" da seguinte forma:

recipe_id	nutri_0	nutri_1	nutri_2	nutri_3	nutri_4
1	487.8	16.0	283.0	13.0	52.0
2	250.0	8.5	150.0	9.0	30.0
3	NaN	NaN	NaN	NaN	NaN

Feature que captura o “Humor” do cliente

O objetivo dessa feature é calcular o desvio da avaliação de um cliente em relação à sua própria média. Por exemplo, quando um cliente que costuma dar apenas nota 5 atribui uma

nota 4 ou 3, isso indica uma avaliação negativa. Por outro lado, quando um cliente mais crítico que geralmente dá nota 3 atribui uma nota 4 ou 5, isso representa uma avaliação positiva.

```
train_final['rating_desvio_usuario'] = train_final['rating'] - train_final['media_rating_usuario']
validation_final['rating_desvio_usuario'] = validation_final['rating'] - validation_final['media_
rating_usuario']
test_final['rating_desvio_usuario'] = test_final['rating'] - test_final['media_rating_usuario']

print("\nFeature de desvio do usuário criada:")
print(train_final[['rating', 'media_rating_usuario', 'rating_desvio_usuario']].head())
```

Etapa 3: Modelos de IA

Como definido inicialmente, nosso objetivo é prever a nota de 1 a 5 para diferentes tipos de refeição, caracterizando um problema de REGRESSÃO. Para solucionar esta questão, selecionei 3 modelos de IA distintos:

Dummy Regressor

O DummyRegressor não é um modelo "inteligente", mas sim um modelo simplificado que serve para estabelecer a **baseline** do projeto. No meu caso, utilizei a estratégia `'mean'`, que faz com que o modelo sempre preveja o `rating` médio de todas as avaliações do conjunto de treino para qualquer receita. O propósito é garantir que qualquer outro modelo mais complexo que eu desenvolva tenha um erro RMSE menor que o DummyRegressor.

Random Forest Regressor

O Random Forest Regressor, em vez de utilizar apenas uma árvore de decisão para fazer as previsões, constrói uma "floresta" com várias árvores. Cada árvore é treinada de maneira ligeiramente diferente, tanto em termos de dados quanto de características. Para fazer a previsão final, o modelo calcula a média das previsões de todas as árvores individuais.

Esse modelo foi escolhido por ser amplamente reconhecido, oferecer robustez contra overfitting, estabilidade em diversos conjuntos de dados e consistentemente apresentar bom desempenho em problemas de regressão.

XGBoost

O XGBoost também é um modelo baseado em árvore de decisão, mas a diferença é que ele utiliza uma técnica de reforço. Ao invés de construir árvores independentes como o Random Forest, o XGBoost constrói árvores de maneira sequencial. Cada árvore nova é treinada para corrigir os erros da árvore anterior, resultando em um modelo extremamente preciso.

Selecionei este modelo por ser amplamente utilizado para este tipo de problema, apresentar maior precisão que o Random Forest, ser extremamente eficiente e rápido, além de oferecer diversos hiperparâmetros que permitem ajustes específicos para melhorar os resultados.

Etapa 4: Avaliação

Nesta seção vamos analisar os resultados dos modelos treinados, com o objetivo de medir a precisão das previsões e verificar se os modelos apresentam **overfitting ou underfitting**. Para este problema de regressão, a métrica escolhida para avaliar o desempenho dos modelos foi o **RMSE (Root Mean Squared Error ou Raiz do Erro Quadrático Médio)**.

Esta métrica foi selecionada porque o RMSE penaliza erros maiores de forma mais significativa que erros pequenos. Por exemplo, errar por 2 estrelas é consideravelmente mais grave que errar por 0,5 estrela, e o RMSE captura bem essa diferença.

Análise do Random Forest

```
RMSE de Treino do RandomForest: 0.0066
RMSE de Validação do RandomForest: 0.1241

Lembre-se: O RMSE do Baseline é: 1.3468
```

Analisando os dados de treino, observamos que o Random Forest apresenta um erro de treino de apenas `0.0066`, indicando que o modelo aprendeu quase perfeitamente os dados de treino, conseguindo "memorizar" as respostas vistas durante o treinamento.

Já o erro no conjunto de validação aumentou para `0.1241`. Embora este valor não pareça muito alto, a diferença significativa entre o desempenho de treino e validação revela claros sinais de **overfitting**. O modelo memorizou excessivamente os dados de treino, por isso, quando exposto a dados novos, sua capacidade de previsão diminuiu consideravelmente.

Análise do XGBoost

```
RMSE de Treino do XGBoost: 0.0204
RMSE de Validação do XGBoost: 0.0772

Lembre-se: O RMSE do Baseline é: 1.3468
```

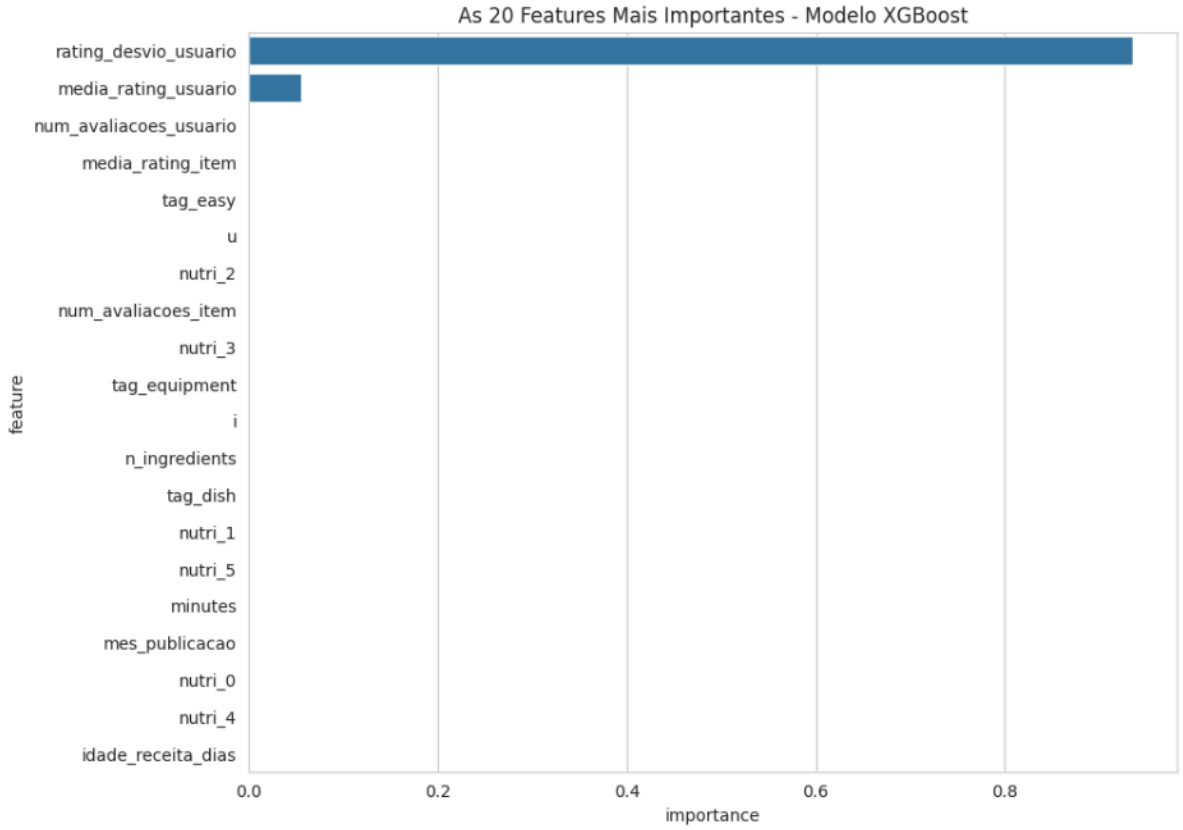
Analisando os dados de treino do XGBoost, observamos que o erro de treino é de apenas `0.0204`, o que indica que o modelo foi capaz de aprender bem os padrões dos dados de treino. Quanto aos dados de validação, o erro é de `0.0772`, valor que não apenas é baixo, mas também consideravelmente menor que o do Random Forest (`0.1241`). Além disso, a diferença entre os erros de treino e validação é muito menor.

Concluímos que o XGBoost é um modelo muito mais equilibrado e com maior capacidade de generalização. Mesmo existindo um pequeno grau de overfitting, o "gap" é controlado, o que demonstra que o XGBoost captura os padrões mais relevantes dos dados em vez de simplesmente memorizá-los

Etapa 5: Interpretação

Após treinar e validar nosso modelo campeão (XGBoost), vamos utilizar o **Feature Importance** para calcular a pontuação de cada feature com base no quão útil ela foi na construção das

árvores. Em outras palavras, o gráfico de Feature Importance mostra as informações que o modelo mais considerou para chegar a um resultado.



Analisando o resultado desse gráfico, podemos ver que o modelo XGBoost depende quase exclusivamente da feature **rating_desvio_usuario**, responsável por cerca de **90% do poder preditivo**. Isso demonstra que o modelo prevê melhor a nota de um usuário ao analisar como ele se desvia do seu próprio padrão de avaliações.

Features como **media_rating_usuario** e **num_avaliacoes_usuario** ajudam a contextualizar esse perfil, mas têm impacto significativamente menor. Já as características das receitas — como tempo de preparo, ingredientes, nutrição e popularidade — tiveram importância quase nula porque seus efeitos já estavam incorporados no padrão de comportamento do usuário.

Essa análise revela que o modelo é altamente personalizado e que as futuras melhorias devem focar no comportamento individual dos usuários, em vez das características das receitas.

Etapa 6: Refinamento

Após a avaliação do ponto anterior, realizamos uma etapa de refinamento com o objetivo de otimizar os hiperparâmetros dos modelos, buscando corrigir o overfitting do Random Forest e extrair uma performance melhor do XGBoost.

Refinamento do Random Forest

O modelo original do Random Forest apresentou um overfitting severo, com um erro de treino de `RMSE=0.0066` drasticamente inferior ao de validação `RMSE=0.1241` , para diminuir esse overfitting

a estratégia é fazer a arvore de decisão ser menos complexa, a principal alteração foi no hiperparâmetro `min_samples_leaf`, que define o número mínimo de amostras em um nó folha, aumentando o seu valor a gente impede que o modelo crie regras muito especificas nos dados.

Hiperparâmetro	Modelo Original	Modelo Refinado
min_samples_leaf	15	40
RMSE Validação	**0.1241**	0.1542

A tentativa de regularização parece que não deu muito certo em termos de precisão, o RMSE de validação piorou, aumentando de 0.1241 para 0.1542. Isso mostra que ao forçar o modelo a ser mais simples para combater o overfitting a gente acabou simplificando demais e caindo em um underfitting e perdendo nossa capacidade de predição.

Refinamento do XGBoost

O modelo atual ja possui uma excelente performance e uma boa generalização, o objetivo aqui é fazer apenas um ajuste fino para buscar uma melhoria na precisão.
Foi adotada uma estratégia de aprendizado mais gradual, reduzindo a `learning_rate` para passos menores e compensando com um maior número de árvores `n_estimators`, o que favorece uma convergência mais estável e potencialmente melhor do modelo.

Hiperparâmetro	Modelo Original	Modelo Refinado
n_estimators	200	500
learning_rate	0.05	0.02
RMSE Validação	0.0772	0.0771

O refinamento reduziu o RMSE de 0.0772 para 0.0771, mostrando um ganho marginal que confirma a eficácia do aprendizado mais lento, mas indicando que o modelo já está próximo do desempenho ótimo e que maiores melhorias dependerão da criação de novas features. O refinamento confirmou o **XGBoost** como modelo superior, com RMSE de 0.0771, consolidando-o como escolha final do projeto, enquanto o ajuste no Random Forest reduziu sua precisão.

Conclusão

O projeto na minha visão foi um sucesso, a gente conseguiu desenvolver um sistema de recomendação baseado em XGBoost que teve um alto pode de previsão e com as features personalizadas, principalmente a `rating_desvio_usuario` como a principal feature que captura as preferências do usuário.

Entretanto ainda temos alguns problemas, como o *cold start* (novos usuários e receitas), o modelo ser estático (sem adaptação a mudanças a longo prazo) e a baixa relevância de algumas features criadas.

Para evoluções futuras:

- Criar estratégias para cuidar do *cold start*
- Explorar o deep learning para padrões mais complexos