Aplicação do algoritmo FunSearch na seleção de features para treinamento de Modelos de Predição

1st Luis Carvalho
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
luisfccarvalho00@gmail.com

2nd Adriano Veloso
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
adrianov@dcc.ufmg.br

***Abstract*—This work builds on top of FunSearch [1], an iterative, genetic algorithm, applying it to preprocess and select important features in different types of datasets by using a Large Language Model. One experiment was created for each dataset, which consists of generating an efficient evaluator function and running the algorithm for a specified number of iterations, then comparing with Kaggle user's results for this dataset. For simpler, smaller datasets or with consistent columns, the algorithm performs slightly worse than the Kaggle data scientists, but with consistent improvements. For more complex and poorly cleaned datasets, the amount of features and information consistency poses a challenge to building efficient evaluator functions.**

I. INTRODUÇÃO

A. *Large Language Models*

Ao longo das últimas décadas, diferentes abordagens de aprendizado de máquina foram propostas para processamento de linguagem natural, como N-grams [2], SVMs [3] e redes neurais recorrentes como LSTMs [4]. Mais recentemente, a criação do mecanismo de atenção, que dá origem a arquitetura de Trasformers, gerou avanços significativos na área de processamento natural.

Como apresentado por [5], a Atenção é um mecanismo de associação entre cada palavra e um peso de importancia associado à todas as outras palavras em uma sequencia, de forma que seja possível entender como cada palavra se relaciona com as outras de um texto. O maior avanço deste mecanismo se dá no processamento de todas as informações paralelamente, a partir do uso de matrizes, ampliando as possibilidades de uso para sequências maiores ao utilizar menos memória e diminuir a parte sequencial desta computação, padrão até então.

Os LLMs mais modernos são treinados em corpus robustos e extremamente abrangentes, tipicamente coletados da internet, abrangendo uma vasta gama de fontes como sites, livros, artigos e outras formas de texto digitalizado. Essa ampla base de treinamento proporciona aos modelos um conhecimento de mundo generalizado, permitindo que respondam com certa veracidade questões sobre uma variedade de domínios. Além disso, essa capacidade de aprendizado a partir de dados diversos confere aos LLMs uma grande flexibilidade em sua aplicação, permitindo que eles realizem tarefas complexas como tradução, resumo e geração de texto.

Estudos recentes buscam avaliar e expandir capacidade de resolver problemas conhecidos e mais complexos a partir da aplicação de diferentes métodos de prompting, como Chain of Thought [6], Graph of Thoughts [7] e mesmo Zero-shot [8], que buscam alterar a resposta do modelo ao induzí-lo a "pensar" de formas diferentes. Ainda sim, uma questão fundamental surge: estes modelos conseguem resolver estes problemas porque os métodos para resolvê-los foram aprendidos no seu treinamento ou eles conseguem gerar novas soluções?

B. *Ciência de dados e LLMs*

A Ciência de dados é uma área científica que envolve dados, estatística e o estudo sistemático de organização, propriedades e análise de dados e seu papel na inferencia de informações [9]. Com crescente disponibilidade de dados e valor econômico destes, o principal desafio atual da área é encontrar valor nos dados e formas de criar sistemas capazes de tomar decisões, encontrar relações, resolver problemas e gerar lucros e produtividade [10].

Neste contexto, surge a Exploração de dados, uma tarefa demorada e intensiva que extrai ideias e correlações importantes de conjuntos de dados [11]. A exploração de dados tem várias etapas, como a limpeza de dados e a visualização de dados, e demanda extenso conhecimento do domínio analisado e conceitos estatísticos, além de tipicamente conhecimento de uma linguagem de programação como Python ou R. A partir dessa exploração, é possível encontrar tendências, anomalias e potenciais usos dos dados coletados enquanto avalia a utilidade destes.

A necessidade de conhecimento de domínio e conhecimento estatístico, limita bastante a disponibilidade de profissionais qualificados que possam adequadamente analisar estes dados. Além disso, a natureza iterativa da tarefa faz com que grande parte do tempo destinado à solução de problemas com dados seja gasto nesta etapa, aumentando custos e limitando a produtividade de times que lidam com quantidades massivas de dados.

Assim, o uso de LLMs surge como uma oportunidade de otimização nas mais diversas etapas deste processo, dentre as quais pode-se destacar: a limpeza dos dados, a avaliação conceitual de soluções, geração de visualizações mais detalhadas ou seleção de features utilizadas nos modelos.

C. *FunSearch*

No contexto de geração de novas soluções, o FunSearch [1] surge como uma abordagem inovadora para explorar a capacidade de LLMs de gerar diferentes soluções para problemas, aproveitando a criatividade destes modelos para impulsionar a evolução de uma função inicialmente simples.

O algoritmo apresentado propõe a utilização de LLMs como gerador de soluções diversas, explorando diferentes ângulos dos problemas apresentados por meio de um algoritmo genético com várias ilhas. Embora essa variação seja boa no contexto teórico, existe uma chance de que as soluções se distanciem do problema proposto, fazendo-se necessária a utilização de limitadores, como uma função de avaliação, que pode rejeitar ou avaliar o resultado com uma nota, alterando a probabilidade de novas gerações o utilizarem.

D. *Objetivos*

O presente trabalho buscou construir uma implementação de um algoritmo baseado no FunSearch, em que o código a ser evoluído deve gerar uma seleção de features para um modelo de machine learning, bem como criar problemas base para serem utilizados na construção de novos experimentos. Como avaliação, foi proposto que os resultados gerados pelo modelo sejam comparados com resultados de usuários da plataforma Kaggle.

E. *Estrutura do Trabalho*

Na primeira sessão, é brevemente apresentado o algoritmo e o artigo originais que inspiram os experimentos deste trabalho. Na sessão seguinte, é apresentada a metodologia com 3 fases, sendo a primeira a implementação adaptada do algoritmo FunSearch, a segunda uma rápida abordagem de teste para definir o modelo a ser avaliado e a terceira a execução dos experimentos. Na etapa de resultados, são destacados alguns recortes à cerca das hipóteses feitas na sessão de metodologia e apresentados os resultados quantitativos das variações propostas. Ao fim, há uma breve discussão sobre as conclusões e propostas do trabalho, seguidas de sugestões para melhoria da aplicação no futuro.

II. Referencial Teórico

O referencial teórico vem de duas vertentes principais, sendo a primeira e mais importante o algoritmo FunSearch e a segunda de diferentes abordagens e utilizações de LLMs:

A. *FunSearch*

O algoritmo FunSearch, apresentado por RomeraParedes *et al.* (2023) no trabalho intitulado "Mathematical discoveries from program search with large language models", demonstra a efetividade do uso de LLMs no contexto de explorador de conjunto de soluções matemáticas. O funcionamento básico do algoritmo é descrito abaixo.

São descritos os diversos componentes utilizados para o funcionamento do algoritmo:

- Amostrador (LLM)
- Avaliador
- Banco de programas

O amostrador tem como função escolher quais programas serão retirados do banco de programas e então gerar um novo programa a partir dele, utilizando a LLM. A ideia é que ao combinar diferentes programas em um prompt de exemplo, ocasionalmente novas ideias serão geradas na saída.

O avaliador tem como função rodar o código e avaliar o resultado comparativamente com uma função pré-definida. O avaliador também controla o crescimento do banco de dados com um algoritmo genético, utilizando ilhas de programas e descartando parte delas periodicamente.

Além disso, são descritas as propriedades necessárias para a utilização do FunSearch:

1) Um avaliador eficiente deve existir.
2) A função de feedback deve possibilitar quantificar melhoras.
3) O problema deve poder ser reduzido à uma parte isolada para evolução.

Destaca-se também nesta arquitetura a natureza distribuída dos componentes do FunSearch, de forma que os avaliadores rodam em paralelo, tipicamente em hardware barato de CPUs, enquanto os amostradores rodam em GPUs, em menor quantidade.

Davis, no entanto, levanta pontos interessantes sobre a viabilidade e a inovação trazida por essa solução no review intitulado "Using a large language model to generate program mutations for a genetic algorithm to search for solutions to combinatorial problems: Review of (Romera-Paredes et al., 2023)). O autor argumenta que o FunSearch demonstra mais sobre a estrutura das funções para os problemas apresentados do que sobre a capacidade da LLM, e que o aspecto mais interessante do trabalho foi a interação LLM-Matemático, combinando conhecimento de domínio com o algoritmo [12].

B. *Outras Abordagens*

Rometiolis *et al.* demonstraram a viabilidade do uso de LLMs para avaliação de modelos não supervisionados no contexto de sistemas de recomendação [13], embora a abordagem seja mais focada em texto do que código, apresentando resultados textuais gerados ao LLM.

Conforme apresentado em [14], a capacidade de resolução de tarefas diversas por LLMs pode ser potencializada por interpretadores matemáticos, de forma que a LLM é responsável pelas decisões lógicas de alto nível, enquanto a execução matemática é delegada à outros componentes especializados.

Ainda, em [15], é apresentada uma abordagem promissora que divide o processo de avaliação de dados em três partes: análise, ferramentas e verificação. Neste trabalho, é apresentado o Data Interpreter, que busca melhorar o processo de avaliação ao (1) planejar os passos de acordo com os dados; (2) recomendar diferentes ferramentas para diferentes tipos de tarefas; (3) verificar as soluções por diferentes critérios de confiança.

III. Metodologia

A metodologia utilizada consistiu em 3 fases:

1) Montagem do código base do algoritmo genético e de avaliação.
2) Testes e definição da LLM a ser utilizada.
3) Execução dos experimentos e coleta de dados de execução.

A. Código base do algoritmo

Uma parte significativa do código foi copiado do material auxiliar do paper de apresentação do FunSearch, enquanto outras partes foram adaptadas, reescritas ou implementadas completamente. A implementação final foi disponibilizada no github: [LINK GITHUB].

Destacam-se como mudanças significativas em relação ao original:

1) Implementação em memória:

a) Os experimentos originais, por rodarem até milhões de vezes e lidar com paralelização, precisavam de um mecanismo mais robusto de registro dos programas. O menor tempo de execução e a simplificação de outros componentes possibilitaram uma implementação rápida totalmente em memória, com objetos, arrays, listas e conjuntos.:

b) Para uma exploração mais detalhada dos resultados, foram implementados backups e salvamento de resultados em arquivos CSV a cada alguns minutos. Os backups contam com todo programa que entrou no banco de dados, sua nota e o número da amostra que gerou o programa. Os resultados, salvos a cada 100 amostras, mostram a melhor nota obtida em cada ilha, além do número de programas em cada ilha.:

2) Possibilidade de importação de bibliotecas externas:

a) Os prompts de cada problema encorajavam a utilização e a importação de bibliotecas externas se necessário, que, no caso de serem chamadas no código gerado, também rodavam. Os imports sequenciais funcionaram bem na prática, mas sugere-se que estes sejam movidos para o topo do código do problema antes da execução por motivos de consistência.:

3) Execução síncrona e com único par gerador-avaliador:

a) O FunSearch foi modelado para problemas combinatórios e de avaliação com algoritmos simples, tipicamente executados em multiplas threads de CPUs, presentes em quantidades razoáveis mesmo em sistemas mais antigos. Para uma abordagem de avaliação de modelos, o gargalo passa diretamente à parte de avaliação, já que a geração de modelos de machine learning utilizam predominantemente a GPU, que tem mais poder de processamento, mas em uma única unidade, de custo mais elevado.:

b) Com a baixa capacidade de gerar e avaliar estes modelos, com apenas um avaliador, não faria sentido utilizar mais de um gerador de código, tornando uma solução assíncrona desnecessariamente complicada para os experimentos sugeridos. Além disso, o código auxiliar não continha a implementação, apenas comentários com sugestões de implementação.:

4) Implementação do avaliador como processo:

a) O avaliador foi implementado como um processo secundário na mesma máquina em que o experimento rodava. Cada programa gerado é escrito em um arquivo temporário do tipo py e executado em um subprocesso, e o resultado retornado é a saída padrão do processo (ou seja, os comandos do tipo "print"), que contam com o resultado obtido e as colunas utilizadas.:

b) Por utilizar diversas bibliotecas, como numpy e pandas, em alguns momentos o código rodado pode gerar warnings e erros. Se o resultado obtido for -1 ou for detectado um erro, o programa é rejeitado e não vai para o banco de dados, enquanto warnings não alteram a nota e nem rejeitam o programa. No geral, ler a última linha retornada foi o suficiente para receber as informações necessárias, embora sugere-se que um hash seja inserido na saída do processo e lido no avaliador para maior confiabilidade na estrutura dos resultados do programa.:

5) Mudanças no agrupamento:

a) O resultado obtido por um programa não é mais um número natural, e sim um número real, o que impede a utilização do resultado como identificador de um tipo de soluções em clusters. O agrupamento utilizado foi, inicialmente, o número de colunas retornadas pelo programa, e posteriormente, o nome das colunas ordenadas e concatenadas. Como não foi possível notar nenhum tipo de diferença, o número de colunas foi re-estabelecido em razão da simplicidade.:

6) Contexto da função evolutiva: Para fazer o uso do conhecimento prévio das LLMs, que são altamente dependentes de contexto, é enviado junto ao prompt de cada requisição de código informação sobre as colunas das bases de dados, como nome ou exemplos de potenciais valores, retiradas integralmente do Kaggle.

Por utilizar LLMs de propósito geral, as instruções para geração de código também são adicionadas ao final deste contexto, como mostrado na figura. Naturalmente, o código que faz a avaliação e a função avaliadora não são enviados junto ao prompt pois poderia influenciar nas decisões da LLM.

B. Modelos

Os cinco modelos cotados foram:

1) Llama-2-7b (LOCAL)
2) Deep Seek Coder 6.7B (LOCAL)
3) Mixtral 7B (LOCAL)
4) GPT-3.5 Turbo (externo, da empresa OpenAI)
5) GPT-4 (externo, da empresa OpenAI)

Os requisitos necessários para a utilização de um modelo eram:

1) Capacidade de geração de código (código roda consistentemente)
2) Capacidade de eliminar ou separar texto adicional na resposta
3) Velocidade de geração

Inicialmente, esperava-se utilizar um modelo local de tamanho pequeno, mas houveram diferentes problemas com cada um deles: no Llama-2-7b, o código gerado raramente

A model to detect higher chances of heart attack is being trained on a dataset.
These are the attributes of the dataset:

- age : Age of the patient
- sex : Sex of the patient
- cp : Chest pain type ~ 0 = Typical Angina, 1 = Atypical Angina, 2 = Non-anginal Pain, 3 = Asymptomatic
- trtbps : Resting blood pressure (in mm Hg)
- chol : Cholestoral in mg/dl fetched via BMI sensor
- fbs : (fasting blood sugar > 120 mg/dl) ~ 1 = True, 0 = False
- restecg : Resting electrocardiographic results ~ 0 = Normal, 1 = ST-T wave normality, 2 = Left ventricular hypertrophy
- thalachh : Maximum heart rate achieved
- oldpeak : Previous peak
- slp : Slope
- caa : Number of major vessels
- thall : Thalium Stress Test result ~ (0,3)
- exng : Exercise induced angina ~ 1 = Yes, 0 = No

The target column has been removed from the dataset.
Complete the function to preprocesses the dataset, returning a dataset with up to FIVE important columns.
Process each column in any way that seems plausible.
Import libraries as needed.

Fig. 1: Exemplo de contexto de prompt, adicionado antes das funções.

rodava, menos de 40% das vezes, mesmo com contexto detalhado sobre a função; no DeepSeekCoder e no Mixtral 7B, embora o código tivesse resultados melhores, a capacidade de gerar apenas código ou isolar o código gerado do texto da resposta não foi satisfatória.

Das opções externas, o GPT-3.5 Turbo supriu todos os três requisitos, além de ser um dos modelos mais baratos no tempo em que este trabalho foi feito, portanto foi o escolhido. Foi testado rapidamente o GPT-4, que gerava consistentemente códigos mais complexos para o contexto de teste e também falhava com uma alta taxa, acima de 50% das vezes, além de ter um custo 10 vezes maior que o GPT-3.5 Turbo.

Apesar do custo mais baixo dentre as opções, destaca-se o alto custo de modelos disponibilizados por empresas especializadas como a OpenAI. Como comparação, o custo de todos os experimentos do FunSearch para o modelo Codey (Google) em hardware dedicado na plataforma Google Cloud foi estimado em pouco menos de 1000 dólares, com experimentos de até dois milhões de iterações [1], enquanto o custo dos experimentos deste trabalho foi de 81 dólares para experimentos de até cinco mil iterações.

C. Execução dos experimentos

Foram escolhidos três bases de dados do Kaggle. Todas as bases selecionadas tinham pelo menos 50 notebooks associados, para gerar uma avaliação minimamente consistente, mas outros critérios necessários foram estabelecidos para cada experimento para que o algoritmo pudesse ser experimentado em diferentes configurações.

1) Base 1 - Experimento simples de alvo binário com menos de 15 colunas - https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset:

a) A base escolhida para o problema um deveria ter colunas consistentes e uma variável alvo binária, com menos de 15 colunas. A ideia por trás deste problema era fazer uma prova de conceito com pouco potencial de falha.:

b) Esta base de dados contém informações de pacientes que reportaram algum tipo de dor no coração e o diagnóstico final. As colunas contém informações médicas como idade, sexo, tipo de dor, colesterol, açucar no sangue, taxa de batimentos, etc. A coluna alvo é o diagnóstico final, infarto ou não.:

2) Base 2 - Experimento de alvo binário com mais de 15 colunas -https://www.kaggle.com/datasets/mnassrib/telecom-churn-datasets:

a) A base escolhida para o problema deveria ter colunas consistentes e uma variável alvo binária, mas mais de 15 colunas. A ideia por trás deste problema era fazer uma prova de conceito com um número mais baixo de colunas, de forma à forçar mais variedade de soluções.:

b) Esta base de dados contém informações sobre a atividade de consumidores de uma empresa de telecomunicações. As colunas contém informações de uso dos planos, como tempo de ligação, estado, tipo de plano e horário de utilização. A coluna alvo é o cancelamento do plano, com um desbalanceamento notável (15% cancelados, 85% não cancelados).:

3) Base 3 - Experimento de alvo multiclasse com mais de 15 colunas - https://www.kaggle.com/datasets/parisrohan/credit-score-classification/data :

a) A base escolhida para o problema deveria ter colunas pouco consistentes e uma variável alvo multiclasse, além de mais de 15 colunas. A ideia por trás deste problema era, além de forçar a escolha de um número pequeno de colunas, experimentar o potencial da LLM de limpar os dados das colunas antes da seleção.:

b) Esta base de dados contém o histórico de crédito de diferentes clientes de um banco. As colunas contém informações de utilização de cartões, salário, dívidas passadas e atuais. A coluna alvo é a nota de crédito do usuário. Destaca-se que grande parte das colunas tem diversas categorias e os dados são pouco consistentes, de forma que pode haver strings em colunas numéricas e valores numéricos em colunas que são indicadas como strings.: A função evolutiva de todos os experimentos foi escrita de forma a receber um dataframe e retornar um dataframe, com um esqueleto de código que iniciava com uma coluna escolhida ao acaso.

A função avaliadora de todos os experimentos foi baseada em modelos simples e de rápido treinamento da biblioteca sklearn. Inicialmente, um pedaço da base de dados foi retirado para teste (sempre o mesmo), então a variável alvo é retirada, e só então a função gerada é executada. O resultado a ser retornado, de forma simplificada, é a média dos scores F1 de cada um dos modelos utilizados em relação ao conjunto de testes retirado, e o resultado seria igual a -1 se qualquer erro fosse disparado no código.

O experimento manteve a sugestão de utilização de 4 ilhas para uma maior diversidade de programas, mas todos os outros parâmetros utilizados foram reduzidos do paper original, como a temperatura inicial do sample e o número de iterações para uma maior temperatura. As piores ilhas foram descartadas a cada 15 minutos.

Para cada problema, foram executadas rodadas de teste de até 100 iterações para testar variações de resposta a partir de mudanças no prompt. Para o experimento real, foram executadas até 6 rodadas de 5000 iterações, com o modelo GPT Turbo 3.5, a partir da API da OpenAI.

Ao longo da execução dos experimentos, notou-se uma prevalência grande de programas repetidos no banco de dados. Como a nota de cada programa já é influencia na probabilidade do programa ser escolhido e um prompt que inclui dois programas iguais tem maiores chances de gerar um programa igual, supôs-se que isto seria um problema que levaria à uma explosão no número de programas iguais, acarretando em uma menor variabilidade. Assim, foram executados também experimentos que rejeitavam a inclusão de um programa em uma ilha se a ilha já tivesse um problema idêntico.

Ao longo desta etapa foram feitas também correções gerais no código e variações pequenas para testes, como o limite de tempo e o agrupamento nos clusters. Também foram refinados e implementados novos mecanismos de coleta de resultados ao longo do experimento, como o número de programas repetidos em cada ilha.

IV. Resultados

Os resultados apresentados são os melhores obtidos para cada experimento, separados em tipos, como ”Base”, que é o experimento inicial, e ”Sem Repetição”, que é o experimento excluindo repetições. Nota-se que, ao considerar a natureza aleatória da amostragem de respostas das LLMs e do algoritmo genético, cada experimento, mesmo que utilizando os mesmos parâmetros e iniciando com o mesmo código, pode chegar a resultados significativamente diferentes.

A. *Experimento 1 - Alvo binário com menos de 15 colunas*

No primeiro experimento, o melhor resultado base saltou rapidamente nas primeiras iterações, e manteve um modesto crescimento até atingir um platô após cerca de 2000 iterações, como demonstrado na figura 2.

Ao observar mais claramente o platô pela perspectiva da nota de cada programa não rejeitado, podemos perceber que, além do resultado não melhorar, a média dos resultados começa a cair após alguns milhares de iterações, com a dispersão aumentando e resultados relativamente piores, como mostra a figura 3.

Destaca-se também um comportamento desavantajoso que ocorreu consistentemente após a geração de um resultado muito melhor do que o melhor resultado atual, quando este ocorreu após a metade do experimento: ao longo dos próximos descartes das piores ilhas, este resultado tendia a ser escolhido e replicado as novas ilhas, fazendo com que todas elas empatassem tecnicamente e descartes futuros passassem a funcionar de forma bastante aleatória.

Na versão em que programas repetidos não eram reintroduzidos nas ilhas, a performance melhorou, enquanto a distribuição dos resultados foi alterada significativamente, como indicado nas figuras 4 e 5.

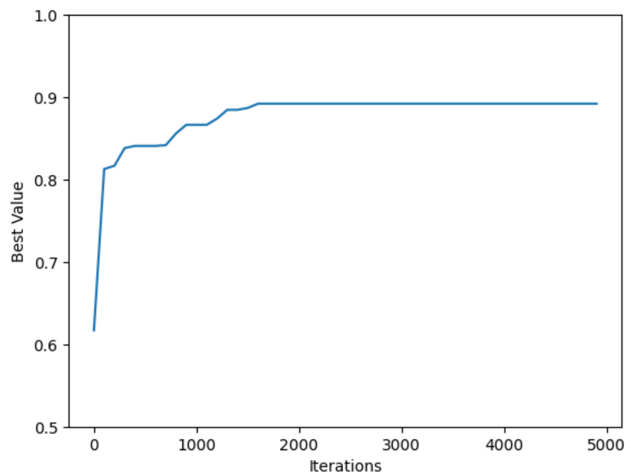


Fig. 2: Melhor Resultado (Base)

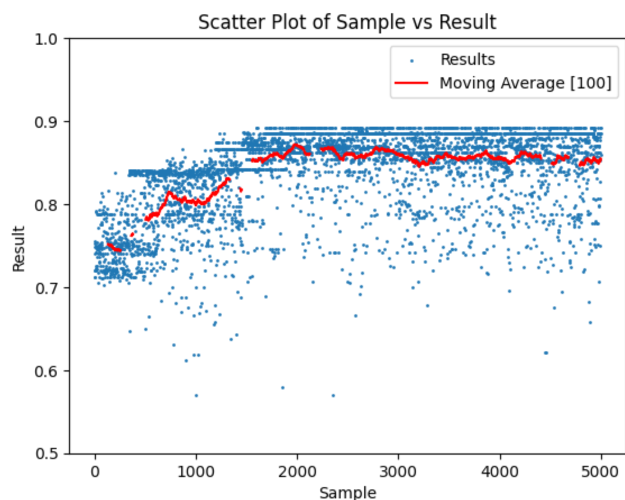


Fig. 3: Resultados ao longo do tempo (Base). A média móvel não é contínua pois ela é calculada nas samples validas, que exclui amostras com resultado igual a menos um; similarmente, nem toda amostra está plotada no gráfico.

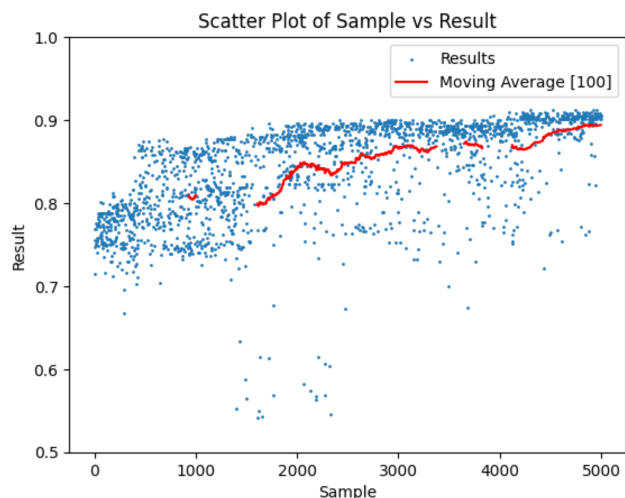


Fig. 5: Resultados ao longo do tempo (Sem Repetição). A média móvel não é contínua pois ela é calculada nas samples validas, que exclui amostras com resultado igual a menos um; similarmente, nem toda amostra está plotada no gráfico.

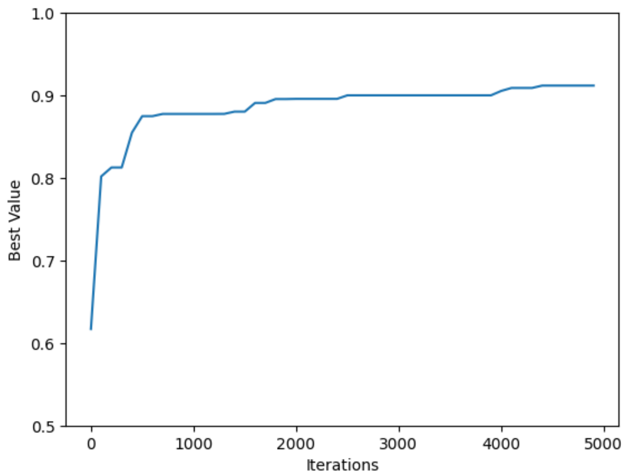


Fig. 4: Melhor Resultado (Sem Repetição)

```
Score:  0.892156009413472
Code:

    df = df[['cp', 'thalachh', 'oldpeak', 'caa']]

    # Binarization based on mean
    df['thalachh_binary'] = np.where(df['thalachh'] > df['thalachh'].mean(), 1, 0)
    df.drop(columns=['thalachh'], inplace=True)

    # Log transformation
    df['oldpeak'] = np.log1p(df['oldpeak'])

    # One-hot encoding for 'cp'
    df = pd.get_dummies(df, columns=['cp'])

    # Binary encoding for 'caa'
    df['caa'] = (df['caa'] > 0).astype(int)

    # Standard Scaling for 'oldpeak' and 'cp_0', 'cp_1', 'cp_2', 'cp_3'
    for col in ['oldpeak', 'cp_0', 'cp_1', 'cp_2', 'cp_3']:
        df[col] = (df[col] - df[col].mean()) / df[col].std()

    return df
```

Fig. 6: Experimento 1 - Melhor código gerado (Base)

Os melhores programa gerados, mostrados nas figura 6 e 7 indica uma capacidade de selecionar colunas relevantes, a partir da escolha de 4 das 13 colunas existentes, além da capacidade de processar as colunas de acordo com o contexto, levando em conta tanto o tipo da variável quanto potenciais padronizações utilizadas por modelos específicos.

Os melhores resultados registrados pelo algoritmo foram então 0.892 e 0.911. Para comparação com o Kaggle, tomando como base o notebook mais votado no qual foi indicado resultados de um modelo de predição, o melhor resultado atingido foi de 0.918 de acurácia, bastante similar ao gerado

```
Score:  0.911897756362651
Code:

    df.fillna(df.median(), inplace=True)
    df['thalachh'] = np.log1p(df['thalachh'])
    df = pd.get_dummies(df, columns=['cp', 'restecg'])
    df.dropna(inplace=True)
    df.drop(columns=['exng', 'fbs', 'oldpeak', 'chol', 'trtbps', 'age', 'slp'], inplace=True)
    df['caa'] = np.sqrt(df['caa'])
    df['thalachh'] = (df['thalachh'] - df['thalachh'].mean()) / df['thalachh'].std()
    df['caa'] = (df['caa'] - df['caa'].min()) / (df['caa'].max() - df['caa'].min())
    df['sex'] = np.where(df['sex'] == 'Male', 1, 0)
    # Additional data processing can be implemented here
    return df
```

Fig. 7: Experimento 1 - Melhor código gerado (Sem repetição)

pelo modelo.

B. Experimento 2 - Alvo binário com menos de 15 colunas

No segundo experimento, o melhor resultado base também saltou rapidamente nas primeiras iterações, mas não parece ter atingido um platô, embora o crescimento tenha sido lento, como demonstrado na figura 8a.

Ao contrário do experimento anterior, quando excluidos os programas com repetição, os resultados tiveram leve piora, com uma distribuição bastante alterada, como mostrado nas figuras 9a e 9b. Embora essa alteração radical nos resultados tenha sido observada em dois experimentos diferentes, em maior ou menor grau, não se descarta a possibilidade do algoritmo apenas ter escolhido um espaço de busca pior, isto é, a mudança não tem significancia estatística clara.

Os melhores resultados base e de repetição, mostrados nas figuras 10 e 11, embora relativamente ruins, demonstram a progressão de diferentes estratégias escolhidas pela LLM, potencializadas pelo algoritmo: enquanto um código combina várias colunas entre si e exponencia algumas delas, o outro tenta codificar as colunas como dummies e fazer operações com os quantis.

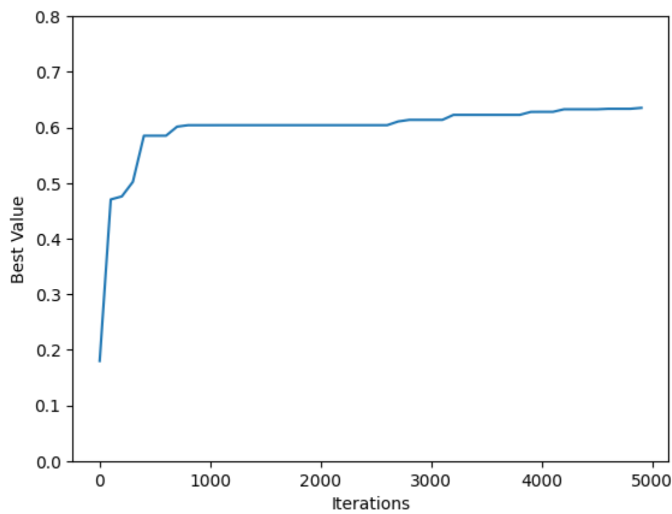
Os melhores resultados registrados pelo algoritmo foram 0.635 e 0.55, respectivamente, nas versões iniciais e sem repetição. Na comparação, o score de todos os melhores notebooks do Kaggle variou entre 0.80 e 0.95 de acurácia, utilizando diferentes abordagens, como XGBoost e KNeighbors com quase todas as colunas.

C. Experimento 3 - Alvo multiclasse com mais de 15 colunas

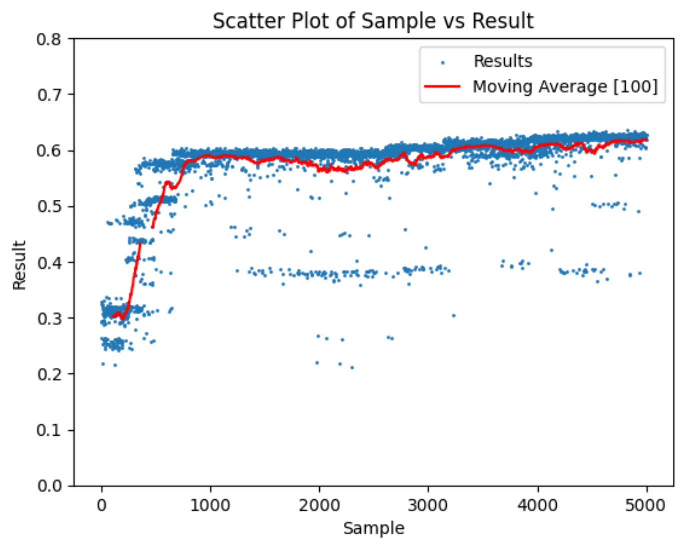
No terceiro experimento, foi descoberta uma grande limitação no algoritmo: quando as colunas são muito inconsistentes, ou seja, tem muitas colunas não estruturadas, mesmo após processamento do LLM, o dataframe final não é robusto o suficiente para treinar uma variedade de modelos. Por este motivo, todos os códigos gerados falham por erro no treinamento, que faz parte da avaliação.

Sem uma função de avaliação funcional, o experimento não progride, já que cada ilha conta com apenas um programa, o original. Foram feitas variações no experimento para tentar mitigar este problema, sem sucesso:

1) Variações no prompt: foram testadas diversas instruções de processamento de colunas, como ”dicas” do que poderia ser feito, contexto adicional sobre os resultados esperados e até mesmo uma lista de potenciais modelos a serem treinados.
2) Variações no contexto: foram testadas diversas variações no contexto da base de dados, como adicionar o tipo esperado da coluna ao final do processamento, exemplos de valores nas colunas e avisos de que tais colunas poderiam conter dados não confiáveis (com diferentes terminologias).
3) Variações na função de avaliação: foram testadas diferentes configurações de modelos, bem como o teste isolado de cada um deles, lançando erros apenas se a maioria falhasse.

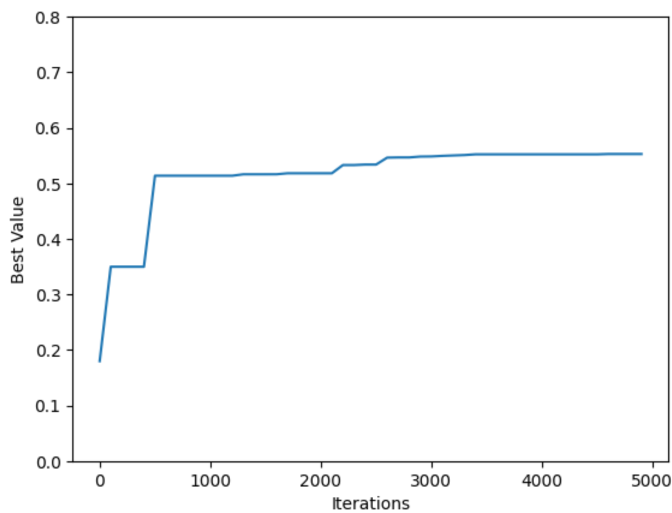


(a) Melhor Resultado (Base)

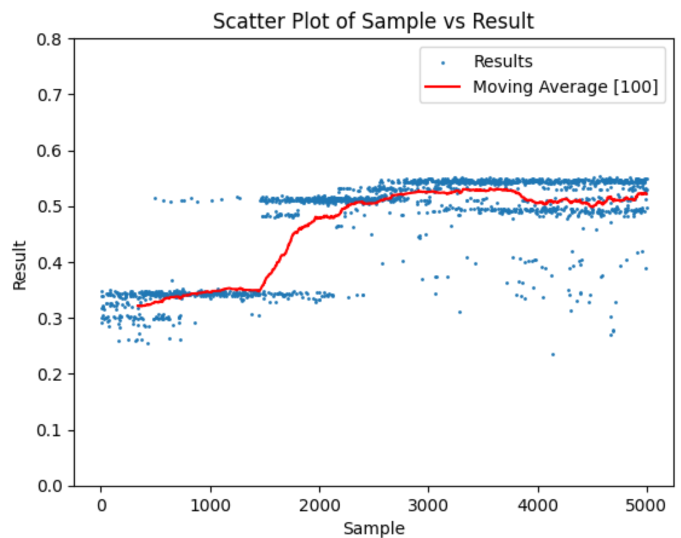


(b) Resultados ao longo do tempo (Base). A média móvel não é contínua pois ela é calculada nas samples validas, que exclui amostras com resultado igual a menos um; similarmente, nem toda amostra está plotada no gráfico.

Fig. 8: Experimento 2 - Resultados Base



(a) Melhor Resultado (Base)



(b) Resultados ao longo do tempo (Base). A média móvel não é contínua pois ela é calculada nas samples validas, que exclui amostras com resultado igual a menos um; similarmente, nem toda amostra está plotada no gráfico.

Fig. 9: Experimento 2 - Resultados sem Repetição

```
Score:  0.635098901881029
Code:

    df = df[['Total day minutes', 'Total eve minutes', 'Customer service calls',
             'Account length', 'Total night minutes', 'International plan',
             'Voice mail plan']]

    df[['International plan', 'Voice mail plan']] = df[['International plan', 'Voice mail plan']].apply(lambda x: x.astype('category').cat.codes)

    for col1, col2 in itertools.combinations(['Total day minutes', 'Total eve minutes',
                                              'Customer service calls', 'Total night minutes'], 2):
        df[f'{col1}_x_{col2}'] = df[col1] * df[col2]

    for col in ['Total day minutes', 'Total eve minutes', 'Customer service calls', 'Total night minutes']:
        df[f'{col}_squared'] = df[col] ** 2
        df[f'{col}_cubed'] = df[col] ** 3

    df = (df - df.mean()) / df.std()

    return df
```

Fig. 10: Experimento 2 - Melhor código gerado (Base)

```
Score:  0.5527332316928217
Code:

    df = pd.get_dummies(df[['Account length', 'Total intl calls', 'International plan', 'Voice mail plan', 'Total day minutes', 'Total eve minutes', 'Total night minutes', 'Total intl minutes'
                        columns=['International plan', 'Voice mail plan'], drop_first=True)

    # Apply Robust Scaling to numerical columns
    numeric_cols = ['Account length', 'Total day minutes', 'Total eve minutes', 'Total night minutes', 'Total intl minutes', 'Customer service calls', 'Total intl calls']
    df[numeric_cols] = (df[numeric_cols] - df[numeric_cols].median()) / (df[numeric_cols].quantile(0.75) - df[numeric_cols].quantile(0.25))

    return df
```

Fig. 11: Experimento 2 - Melhor código gerado (Sem repetição)

V. Conclusões

Os experimentos realizados demonstram que a aplicação de LLMs em Ciência de Dados apresenta um potencial significativo do ponto de vista de otimização do tempo gasto com análises, mas também desafios notáveis, principalmente em relação à formulação do problema e avaliação das ideias geradas. Além disso, a natureza aleatória do algoritmo e dos LLMs atrapalham significativamente a reprodutibilidade e validação de mudanças na estrutura dos experimentos.

No primeiro experimento, os resultados sugerem que o algoritmo apresentado pode identificar colunas relevantes para modelos preditivos de forma eficiente. No entanto, como mostrado na figura 3, ainda faz-se necessária a criação e aplicação de mecanismos adicionais que evitam estagnação e deterioração da performance após uma quantidade alta de iterações. Na comparação com usuários da plataforma Kaggle, as notas obtidas pelo algoritmo eram comparáveis a dos usuários, embora as soluções mais votadas da plataforma fossem bastante simples.

O segundo experimento revelou que a eficiência dos LLMs em contextos de maiores bases de dados ou com maior variabilidade é menos consistente e/ou requer uma maior quantidade de iterações para atingir resultados razoaveis. Na comparação com usuários da plataforma Kaggle, as notas obtidas pelo algoritmo figuravam muito abaixo da média das melhores soluções da plataforma, indicando que transformações simples nos dados não são comparáveis à combinação de técnicas mais avançadas de machine learning.

O terceiro experimento destacou uma limitação crítica do algoritmo, quando aplicado a bases de dados mais complexas e heterogêneas, com muitas colunas não estruturadas. A incapacidade do algoritmo de gerar dataframes robustos para treinamento eficiente de modelos sugere que faltaram mecanismos adequados no algoritmo para lidar com grandes inconsistências nos dados. Propõe-se para estudo futuro que um passo intermediário seja aplicado nestes casos, quando muitos erros na avaliação sejam detectados em um curto intervalo de tempo e o banco de dados tenha apenas o código inicial em cada ilha. A sugestão é que este passo intermediário seja um micro-experimento com um número bastante reduzido de iterações, adicionando parte dos erros gerados ao prompt, de forma parecida ao apresentado no Data Interpreter [15].

Sobre a exclusão de programas repetidos, parece haver algum atributo inerente a base de dados, a formulação do problema ou até a melhor solução atual gerada pela LLM, que acarreta numa melhor ou pior performance desta mudança. Recomenda-se uma avaliação detalhada do contexto e dos códigos gerados ao longo de uma iteração de teste para decidir se esta parte da implementação deveria ser utilizada.

Outra limitação a ser destacada diz respeito a formulação das funções avaliadoras quando aplicadas a dados. Ao utilizar modelos avaliadores, quaisquer que sejam seus métodos de treinamento, propriedades ou premissas, insere-se um viés na avaliação: não necessariamente a otimização da função feita, isto é, as colunas utilizadas e o processamento de cada um delas, se generaliza quando aplicada a outros modelos. Em outras palavras, recomenda-se cuidado na escolha dos modelos escolhidos na avaliação.

Destaca-se ainda que o algoritmo é apresentado como uma solução de otimização, e não de eliminação do conhecimento de domínio: a formulação do problema e a avaliação prática das soluções ainda requerem conhecimento da base de dados e de código. Os resultados obtidos então, principalmente os principais códigos gerados, servem como soluções base a serem avaliadas por alguém com conhecimento de domínio e então utilizadas ou não na implementação de modelos preditivos reais.

References

[1] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi, "Mathematical discoveries from program search with large language models," *Nature*, 2023.

[2] W. B. Cavnar, J. M. Trenkle *et al.*, "N-gram-based text categorization," in *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, vol. 161175. Las Vegas, NV, 1994, p. 14.

[3] A. Lorena and A. Carvalho, "Uma introdução às support vector machines," *Revista de Informática Teórica e Aplicada; Vol. 14, No 2 (2007); 43-67*, vol. 14, 12 2007.

[4] G. Murthy, S. R. Allu, B. Andhavarapu, M. Bagadi, and M. Belusonti, "Text based sentiment analysis using lstm," *Int. J. Eng. Res. Tech. Res*, vol. 9, no. 05, 2020.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[6] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023.

[7] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, and T. Hoefler, "Graph of thoughts: Solving elaborate problems with large language

models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, p. 17682–17690, Mar. 2024. [Online]. Available: http://dx.doi.org/10.1609/aaai.v38i16.29720
[8] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," 2023.
[9] V. Dhar, "Data science and prediction," *Commun. ACM*, vol. 56, no. 12, p. 64–73, dec 2013. [Online]. Available: https://doi.org/10.1145/2500499
[10] F. J. Ohlhorst, *Big data analytics: turning big data into big money*. John Wiley & Sons, 2012, vol. 65.
[11] J. Devore, "Making sense of data: A practical guide to exploratory data analysis and data mining," *The American Statistician*, vol. 61, pp. 370–371, 02 2007.
[12] E. Davis, "Using a large language model to generate program mutations for a genetic algorithm to search for solutions to combinatorial problems: Review of (romera-paredes et al., 2023)." *University of New York*, 2024. [Online]. Available: "https://cs.nyu.edu/~davise/papers/FunSearch.pdf"
[13] K. Roumeliotis, N. Tselikas, and D. Nasiopoulos, "Precision-driven product recommendation software: Unsupervised models, evaluated by gpt-4 llm for enhanced recommender systems," *Software*, vol. 3, pp. 62–80, 02 2024.
[14] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "Pal: Program-aided language models," 2023. [Online]. Available: https://arxiv.org/abs/2211.10435
[15] S. Hong, Y. Lin, B. Liu, B. Liu, B. Wu, D. Li, J. Chen, J. Zhang, J. Wang, L. Zhang, L. Zhang, M. Yang, M. Zhuge, T. Guo, T. Zhou, W. Tao, W. Wang, X. Tang, X. Lu, X. Zheng, X. Liang, Y. Fei, Y. Cheng, Z. Xu, and C. Wu, "Data interpreter: An llm agent for data science," 2024. [Online]. Available: https://arxiv.org/abs/2402.18679