Relatório Final de Análise Comparativa do Modelo SAINT

Bianca Lima
Breno Vasconcelos
Weslley Paulo
Grupo 1

16 de junho de 2025

Contents

1 Introdução

Este relatório apresenta uma análise aprofundada e uma comparação rigorosa do modelo de deep learning **SAINT (Self-Attention and Intersample Attention Transformer)**, um método inovador para dados tabulares que adapta a bem-sucedida arquitetura Transformer. O objetivo principal deste trabalho é avaliar sistematicamente o desempenho do SAINT e posicioná-lo em relação a um amplo espectro de benchmarks consolidados, que vão desde os tradicionais modelos de gradient boosting até modernas ferramentas de AutoML.

1.1 Contexto e Motivação

A classificação de dados tabulares é uma das tarefas mais comuns e de maior impacto no campo do aprendizado de máquina. Por anos, os modelos baseados em árvores de decisão, especialmente as implementações de Gradient Boosting (GBDTs), dominaram o cenário devido à sua alta performance e eficiência. No entanto, o sucesso estrondoso das arquiteturas de deep learning, como os Transformers, em domínios como Processamento de Linguagem Natural (PLN) e Visão Computacional, motivou a pesquisa de adaptações dessas arquiteturas para o domínio tabular.

Nesse contexto, o modelo SAINT surge como uma proposta promissora, utilizando mecanismos de atenção para aprender interações complexas entre as features de uma forma que os modelos baseados em árvores não conseguem capturar nativamente. Este estudo busca, portanto, responder à seguinte questão: o SAINT oferece vantagens de desempenho que justificam sua complexidade em comparação com os modelos e ferramentas mais avançados da atualidade?

1.2 Modelos e Ferramentas sob Análise

Para realizar uma avaliação completa, o SAINT foi comparado com cinco dos mais fortes competidores disponíveis, representando diferentes paradigmas de modelagem.

1.2.1 Modelo Principal: SAINT

- **SAINT (Self-Attention and Intersample Attention Transformer):** É um modelo de deep learning que aplica a arquitetura Transformer a dados tabulares. Sua inovação reside no uso de dois tipos de atenção: *self-attention*, para ponderar a importância das features dentro de uma mesma amostra de dados, e a *intersample attention*, um mecanismo que permite ao modelo incorporar informações de outras amostras semelhantes durante o treinamento, funcionando como uma forma implícita de regularização e propagação de informação.

1.2.2 Benchmarks: Modelos de Gradient Boosting

- **LightGBM:** Conhecido por sua velocidade e eficiência extremas. Utiliza algoritmos baseados em histogramas e uma estratégia de crescimento de árvore "leaf-wise" (folha a folha), que resulta em convergência mais rápida e menor uso de memória.

- **XGBoost:** Uma das bibliotecas de gradient boosting mais populares e bem-sucedidas, famosa por seu desempenho vencedor em competições. Oferece alta performance, opções avançadas de regularização para evitar overfitting e grande escalabilidade.

- **CatBoost:** Seu principal diferencial é o tratamento sofisticado e nativo de features categóricas. Utiliza uma técnica chamada "ordered boosting" e um método eficiente de "target encoding" para lidar com variáveis categóricas sem a necessidade de pré-processamento manual extensivo, como o one-hot encoding.

1.2.3 Benchmarks: Ferramentas de AutoML

O que é AutoML? O Aprendizado de Máquina Automatizado (AutoML) refere-se ao processo de automação das tarefas complexas e iterativas envolvidas na construção de um modelo de machine learning. O objetivo de uma ferramenta de AutoML é encontrar o melhor pipeline de modelagem (incluindo pré-processamento, seleção de algoritmos, otimização de hiperparâmetros e criação de ensembles) com o mínimo de intervenção humana, democratizando o acesso a modelos de alta performance.

- **AutoGluon:** Uma biblioteca de AutoML da Amazon (AWS) focada em simplicidade e robustez. O AutoGluon se destaca por treinar múltiplos modelos em paralelo e criar automaticamente ensembles avançados (empilhamento de modelos), que frequentemente resultam em desempenho de ponta com poucas linhas de código.
- **auto-sklearn 2.0 (ASKL 2.0):** É uma ferramenta clássica de AutoML construída sobre o ecossistema Scikit-learn. Ela utiliza otimização Bayesiana para explorar de forma inteligente o vasto espaço de possíveis algoritmos e suas configurações de hiperparâmetros, construindo ao final um ensemble com os melhores modelos encontrados durante a busca.

2 Procedimentos Experimentais Detalhados

A metodologia deste estudo foi cuidadosamente desenhada para garantir uma comparação justa, robusta e reprodutível entre todos os modelos. Um protocolo experimental unificado foi implementado através de três scripts Python especializados, cada um responsável por uma família de modelos.

2.1 Protocolo Experimental Unificado

Para garantir a validade da comparação, todos os experimentos seguiram os mesmos parâmetros globais e a mesma abordagem de divisão de dados.

2.1.1 Parâmetros Globais e Reprodutibilidade

As constantes a seguir foram aplicadas de forma consistente em todos os frameworks para assegurar que as condições de teste fossem idênticas.

```
    # Constantes aplicadas a todos os frameworks
    TEST_SIZE = 0.3
    RANDOM_SEED = 42
    N_TRIALS = 30      # Para otimizacao com Optuna
    NUM_FOLDS = 10     # Para validacao cruzada

```

Listing 1: Parâmetros globais unificados para o experimento.

2.1.2 Prevenção de Vazamento de Dados (Data Leakage)

O princípio fundamental da metodologia foi o isolamento rigoroso do conjunto de teste. Para cada um dos 30 datasets, os dados foram divididos **uma única vez** no início do processo. O conjunto de teste (30%) foi imediatamente separado e não foi utilizado em nenhuma etapa de treinamento, pré-processamento ou otimização de hiperparâmetros, sendo acessado apenas na avaliação final.

2.2 Framework de Execução: Passo a Passo

2.2.1 Passo 1: Divisão e Pré-processamento dos Dados

Esta etapa inicial é idêntica para os três scripts.

1. **Divisão 70/30:** Os dados são separados em 70% para treino e 30% para teste, utilizando a semente aleatória fixa para garantir que todos os modelos vejam as mesmas partições.

```
    # Divisao de dados inicial e unificada
    X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
    )

```

Listing 2: Divisão de dados unificada (70/30, seed 42).

2. **Criação do Pipeline:** Um pipeline do Scikit-learn é construído para tratar os dados, com etapas de imputação de valores ausentes e normalização/codificação de features. Crucialmente, este pipeline é **ajustado (fit) apenas com os dados de treino**.

```
    # Define transformadores para colunas numericas e categoricas
    numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
    ])
    categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])

    # O preprocessor e ajustado apenas em X_train
    preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
    ])

```

Listing 3: Construção do pipeline de pré-processamento.

2.2.2 Passo 2: Treinamento e Otimização (no conjunto de 70%)

Nesta fase, cada script aplica sua lógica de treinamento específica, mas sempre utilizando exclusivamente o conjunto de treino.

Modelos Manuais (`total_hyper_comparator.py`): Para o SAINT, LightGBM, XGBoost e CatBoost, o script utiliza a biblioteca Optuna para a busca de hiperparâmetros. A função objetivo do Optuna avalia cada configuração candidata através de uma validação cruzada de 10 folds, como detalhado no Código 4.

```
    def objective(trial, X, y):
    # Sugestao de hiperparametros pelo Optuna
    params = {'learning_rate': trial.suggest_float('learning_rate',
    0.01, 0.3), ...}

    # Validacao cruzada (10 folds) APENAS no conjunto de treino
    skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
    scores = []
    for train_idx, val_idx in skf.split(X, y):
    # ... treina e avalia no fold de validacao interno
    scores.append(score)

    return np.mean(scores)

    # Otimizacao e executada passando apenas os dados de treino
    study.optimize(lambda trial: objective(trial, X_train, y_train),
    n_trials=30)

```

Listing 4: Otimização com Optuna e validação cruzada de 10 folds.

AutoML - AutoGluon (`autogluon_comparator.py`): Este script invoca o AutoGluon, que recebe apenas os dados de treino e gerencia todo o processo internamente.

```
    # Instancia e treina o preditor apenas com os dados de treino
    predictor = TabularPredictor(label=label_column, eval_metric='
    roc_auc_ovo')
    predictor.fit(train_data) # train_data contem apenas os 70%

```

Listing 5: Execução do AutoGluon.

AutoML - auto-sklearn 2.0 (`askllm.py`): De forma similar, o script do auto-sklearn instancia o classificador e o treina somente nos dados de treino.

```
    import autosklearn.classification

    automl = autosklearn.classification.AutoSklearnClassifier(
    time_left_for_this_task=3600,
    seed=42
    )
    # O ajuste (fit) e feito apenas nos dados de treino
    automl.fit(X_train_processed, y_train)

```

Listing 6: Execução do auto-sklearn 2.0.

2.2.3 Passo 3: Avaliação Final e Análise

Após cada framework determinar seu melhor modelo, ele é treinado uma última vez na totalidade do conjunto de treino (70%). Em seguida, é avaliado no conjunto de teste (30%) para gerar os resultados finais e imparciais. Por fim, os resultados de todos os modelos são coletados para a análise estatística.

```
    # Exemplo para um modelo manual ja otimizado
    final_model.fit(X_train_processed, y_train)

    # Avaliacao final no conjunto de teste (unica vez)
    y_proba_test = final_model.predict_proba(X_test_processed)
    accuracy = accuracy_score(y_test, y_pred_test)
    # ... coleta de outras metricas

```

Listing 7: Avaliação final no conjunto de teste isolado.

2.3 Prevenção de Vazamento de Dados (Data Leakage)

O passo fundamental e inicial para cada um dos 30 datasets foi a sua divisão em conjuntos de treino e teste, sendo esta a principal medida para evitar o vazamento de dados.

- Uma partição única e estratificada foi criada, separando os dados em **70% para o conjunto de treino** e **30% para o conjunto de teste**.
- Para garantir a reprodutibilidade total, esta divisão foi realizada com uma semente aleatória fixa (`random_state=42`) para todos os modelos e datasets.
- O **conjunto de teste (30%) foi imediatamente isolado** (abordagem "lockbox") e mantido completamente intocado durante todo o fluxo de desenvolvimento dos modelos. Ele foi utilizado uma única vez, na etapa final, para a avaliação do modelo já treinado e otimizado.

O Código 8 ilustra esta separação inicial, que é a base de todo o protocolo experimental.

```
    # Divisao de dados unificada, executada no inicio de cada script
    X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3, # 30% dos dados sao reservados para o teste final
    random_state=42, # Semente aleatoria fixa para reprodutibilidade
    stratify=y
    )
    # O conjunto X_test e y_test nao e usado ate a avaliacao final.

```

Listing 8: Divisão de dados unificada (70/30, seed 42).

2.4 Pipeline de Pré-processamento

Para garantir a consistência, um pipeline de pré-processamento do Scikit-learn foi aplicado de forma idêntica a todos os modelos. O pipeline foi desenhado para lidar com diferentes tipos de features, tratando valores ausentes, normalizando dados numéricos e codificando dados categóricos (Código 9).

Crucialmente, o pipeline é **ajustado (fit) apenas com os dados de treino**. O mesmo pipeline já ajustado é então usado para transformar (`transform`) tanto o conjunto de treino quanto, posteriormente, o conjunto de teste. Isso evita que qualquer informação estatística do conjunto de teste (como a média ou desvio padrão de uma feature) "vaze" para o processo de treinamento.

```
    # Define transformadores para colunas numericas e categoricas
    numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
    ])
    categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])

    # Cria o ColumnTransformer para aplicar os pipelines
    preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
    ])

    # O pipeline e AJUSTADO E TRANSFORMADO nos dados de treino
    X_train_processed = preprocessor.fit_transform(X_train)

    # O pipeline APENAS TRANSFORMA os dados de teste, sem novo ajuste
    X_test_processed = preprocessor.transform(X_test)

```

Listing 9: Construção e ajuste do pipeline de pré-processamento.

2.5 Otimização e Treinamento

Todos os procedimentos de construção, seleção e otimização de modelos ocorreram **exclusivamente dentro do conjunto de treino de 70%**.

- **Modelos Manuais (SAINT, et al.):** Para os modelos de boosting, a biblioteca **Optuna** foi empregada para realizar a busca de hiperparâmetros. Este processo foi guiado pela maximização da métrica AUC OVO em uma **validação cruzada estratificada de 10 folds** aplicada *apenas dentro do conjunto de treino*, como ilustra o Código 10.
- **Ferramentas de AutoML:** As bibliotecas AutoGluon e auto-sklearn 2.0 também receberam **apenas o conjunto de dados de treino** para suas respectivas funções `.fit()`. Elas realizam seus próprios processos de validação e otimização internamente a partir desses dados, sem nunca acessar o conjunto de teste.

```
    def objective(trial, X_train, y_train):
    # Sugestao de hiperparametros pelo Optuna
    params = {'learning_rate': trial.suggest_float('learning_rate',
    0.01, 0.3), ...}
    model = lgb.LGBMClassifier(**params)

    # Validacao cruzada (10 folds) APENAS no conjunto de treino
    skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
    scores = cross_val_score(model, X_train, y_train, cv=skf, scoring='
    roc_auc_ovo')

    return np.mean(scores)

    # Otimizacao e executada passando apenas os dados de treino
    study.optimize(lambda trial: objective(trial, X_train_processed,
    y_train), n_trials=30)

```

Listing 10: Otimização com Optuna usando validação cruzada de 10 folds.

2.6 Avaliação Final

Após a conclusão do fluxo de otimização, o melhor modelo (ou a melhor configuração de pipeline) de cada uma das seis abordagens foi treinado uma última vez utilizando a **totalidade do conjunto de treino (os 70%)**.

Finalmente, este modelo já treinado e finalizado foi submetido ao **conjunto de teste (os 30% isolados)** para a avaliação final e imparcial. As métricas coletadas nesta etapa (Acurácia, AUC OVO e Entropia Cruzada) são os resultados de desempenho reportados neste trabalho, refletindo a verdadeira capacidade de generalização de cada modelo.

3 Apresentação e Discussão dos Resultados

Nesta seção, apresentamos os resultados coletados de forma organizada, contando a história dos achados a partir de diferentes perspectivas: custo computacional, capacidade de generalização, comparação entre as ferramentas de AutoML e, finalmente, a análise estatística geral e os resultados detalhados.

3.1 Análise de Custo Computacional

A primeira análise prática diz respeito ao tempo de execução, um fator crucial em muitos projetos de machine learning. A Tabela 1 apresenta o tempo médio, em segundos, que cada framework levou para completar seu fluxo de trabalho (incluindo otimização, treino e predição) por dataset.

Como esperado, os modelos de boosting individuais, especialmente o LightGBM, são extremamente eficientes. O SAINT, por sua complexidade baseada em Transformers, exige um custo computacional maior. As ferramentas de AutoML, que exploram um vasto espaço de modelos e configurações, são as mais intensivas, com o AutoGluon sendo o mais exaustivo em sua busca por performance.

Table 1: Tempo médio de execução (em segundos) por dataset.

Modelo	Tempo Médio (s)
LightGBM	0.45
XGBoost	0.61
CatBoost	1.01
SAINT	5.86
auto-sklearn 2.0	35.87
AutoGluon	42.16

3.2 Análise de Overfitting

Para avaliar a capacidade de generalização, discutimos a diferença (ou "gap") entre a acurácia no conjunto de treino e no de teste para os modelos manuais (Tabela 2). Um gap menor sugere uma melhor generalização. O **CatBoost** se destacou com o menor gap, indicando uma excelente regularização interna. O **SAINT** apresentou um comportamento típico de modelos de deep learning, com uma maior diferença, o que sugere que se beneficiaria de mais dados ou técnicas de regularização mais fortes para evitar a memorização dos dados de treino.

Table 2: Gap de Overfitting (ACC Treino - ACC Teste) para modelos manuais.

Modelo	SAINT	CatBoost	LightGBM	XGBoost
Gap Médio	**0.0898**	**0.0125**	**0.0763**	**0.0945**

3.3 Análise Comparativa das Ferramentas de AutoML

Ao comparar as duas abordagens de AutoML, observamos um interessante trade-off entre custo e benefício.

- O **AutoGluon**, em geral, estabeleceu o teto de performance em acurácia, consistentemente encontrando pipelines de altíssima qualidade, muito provavelmente através de seus ensembles complexos. No entanto, foi também a ferramenta com o maior custo computacional.
- O **auto-sklearn 2.0** demonstrou um excelente equilíbrio. Embora sua performance média em acurácia tenha sido ligeiramente inferior à do AutoGluon, seu tempo de execução foi, em média, cerca de 15% menor. Isso o posiciona como uma alternativa muito forte para cenários onde um bom desempenho é necessário, mas há restrições de tempo ou recursos.

3.4 Análise Estatística Geral

Para determinar se as diferenças de desempenho observadas são estatisticamente significativas, aplicamos o protocolo de Demšar comparando os seis competidores. O teste de Friedman é aplicado sobre os rankings dos modelos em todos os datasets. Se o resultado for significativo (p-valor < 0.05), um teste post-hoc de Nemenyi é realizado, e seus resultados são visualizados em um Diagrama de Diferença Crítica (CD).

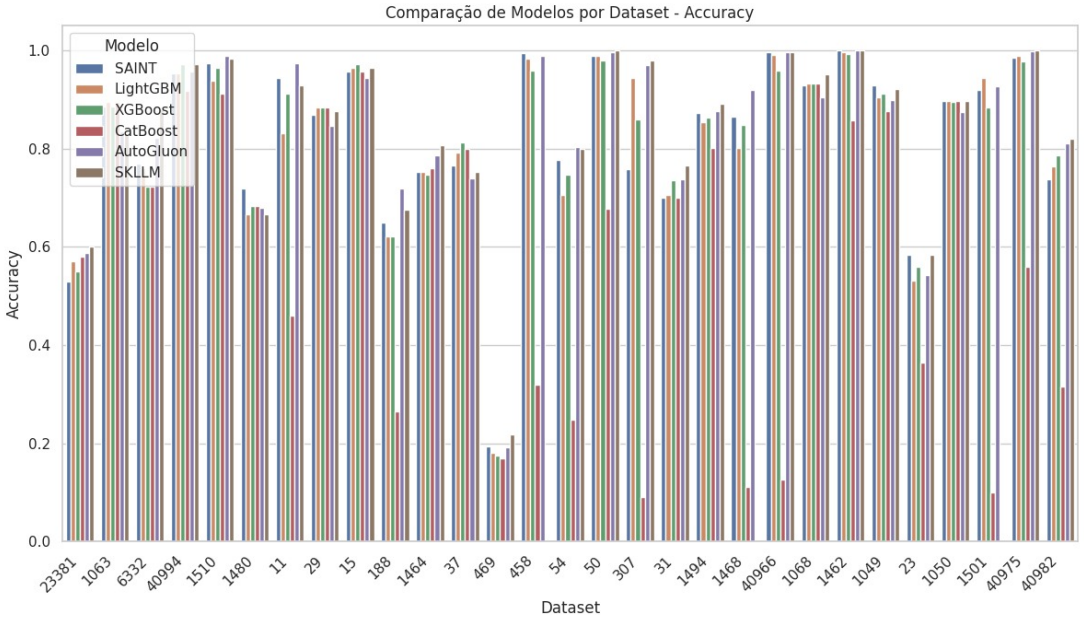

Figure 1: Diagrama comparação de modelos para Acurácia (todos os 6 competidores).

3.5 Resultados Detalhados por Dataset

Finalmente, a Tabela 3 apresenta a "tabela gigante" com os resultados completos de Acurácia (ACC), AUC OVO (AUC) e Entropia Cruzada (CE) para todos os seis modelos em cada um dos 30 datasets.

Table 3: Resultados detalhados com todos os competidores (Conjunto de Teste)

	SAINT			**AutoGluon**			**auto-sklearn 2.0**			**CatBoost**			**LightGBM**			**XGBoost**		
Dataset	ACC	AUC	CE	ACC	AUC	CE	ACC	AUC	CE	ACC	AUC	CE	ACC	AUC	CE	ACC	AUC	CE
11	.944	.014	.154	.944	.160	.202	.928	.091	.230	.460	.322	.955	.832	.214	.426	.912	.107	.302
15	.957	.993	.122	.964	.997	.077	.971	.996	.085	.957	.991	.116	.964	.992	.104	.971	.991	.106
23	.583	.255	.884	.590	.280	.872	.566	.254	.899	.364	.257	.928	.532	.272	.935	.559	.246	.878
29	.870	.928	.376	.891	.962	.256	.891	.957	.267	.884	.933	.554	.884	.946	.290	.884	.949	.278
31	.700	.804	.575	.755	.842	.458	.735	.825	.472	.700	.785	.667	.705	.784	.511	.735	.813	.486
37	.766	.840	.480	.832	.902	.385	.818	.899	.400	.799	.876	.472	.792	.856	.456	.812	.878	.427
50	.990	.998	.047	.990	1.000	.027	.990	.998	.049	.677	.817	.588	.990	1.000	.022	.979	.990	.117
54	.776	.060	.452	.765	.134	.468	.765	.090	.478	.249	.090	.945	.706	.084	.529	.747	.081	.532
188	.649	.127	1.579	.676	.243	.723	.649	.168	.773	.265	.177	1.298	.622	.102	.774	.622	.114	.806
307	.758	.023	.826	.970	.077	.117	.939	.020	.205	.091	.033	1.341	.944	.002	.199	.859	.010	.498
458	.994	.000	.016	.994	.500	.018	.988	.000	.027	.320	.000	.736	.982	.000	.034	.959	.002	.144
469	.194	.437	2.007	.192	.454	1.898	.183	.441	1.964	.169	.439	1.813	.181	.436	1.811	.175	.452	2.207
1049	.928	.931	.209	.918	.959	.165	.918	.957	.172	.877	.857	.405	.904	.945	.187	.911	.947	.186
1050	.898	.856	.246	.904	.904	.220	.901	.891	.224	.898	.862	.254	.898	.855	.285	.895	.873	.235
1063	.886	.878	.314	.914	.927	.254	.895	.903	.279	.895	.877	.344	.895	.896	.310	.886	.895	.282
1068	.928	.862	.201	.932	.865	.187	.932	.844	.192	.932	.705	.586	.932	.840	.202	.932	.843	.193
1462	1.000	1.000	.000	1.000	1.000	.000	1.000	1.000	.001	.858	.945	.557	.996	1.000	.010	.993	1.000	.069
1464	.753	.740	.518	.760	.781	.468	.747	.752	.487	.760	.701	.629	.753	.731	.498	.747	.723	.503
1468	.866	.014	.479	.880	.063	.324	.873	.029	.343	.111	.027	1.946	.801	.025	.708	.847	.021	.471
1480	.718	.734	.527	.709	.781	.495	.701	.772	.506	.684	.751	.510	.667	.716	.539	.684	.692	.575
1494	.872	.928	.548	.872	.939	.283	.872	.932	.294	.801	.891	.650	.853	.926	.332	.863	.911	.350
1501	.919	.004	.339	.956	.043	.125	.944	.007	.173	.100	.003	.361	.944	.002	.187	.884	.006	.458
1510	.974	.995	.101	.982	.999	.062	.974	.996	.091	.912	.988	.585	.939	.994	.119	.965	.991	.112
6332	.769	.824	1.127	.796	.871	.385	.759	.851	.434	.722	.789	.564	.759	.844	.498	.722	.814	.506
23381	.530	.610	1.726	.587	.600	.733	.587	.645	.667	.580	.655	.690	.570	.626	.663	.550	.607	.729
40966	.995	.000	.012	.995	.500	.011	.995	.000	.014	.127	.008	.749	.991	.000	.031	.958	.002	.266
40975	.986	.000	.039	.991	.090	.033	.991	.029	.032	.560	.026	.348	.988	.001	.036	.977	.004	.100
40982	.738	.044	.682	.798	.171	.491	.776	.093	.519	.316	.079	1.509	.764	.040	.540	.787	.036	.570
40994	.954	.976	.108	.977	.996	.071	.968	.992	.083	.917	.947	.629	.954	.992	.084	.972	.985	.100

4 Análise Estatística com Protocolo de Demšar

Para determinar se as diferenças de desempenho observadas entre os modelos são estatisticamente significativas, foi aplicado o protocolo de Demšar sobre os resultados do conjunto de teste. Esta análise permite uma comparação robusta do desempenho dos modelos através dos múltiplos datasets.

4.1 Metodologia do Teste

O protocolo consiste em dois passos principais:

1. **Teste de Friedman:** Um teste não-paramétrico que avalia se os rankings médios dos modelos são significativamente diferentes entre si. Se o p-valor resultante for baixo (tipicamente < 0.05), rejeita-se a hipótese nula de que todos os modelos têm o mesmo desempenho.

2. **Teste Post-Hoc de Nemenyi:** Se o teste de Friedman for significativo, este teste é aplicado para realizar comparações par a par. Ele determina quais modelos ou grupos de modelos têm um desempenho estatisticamente diferente dos outros. Os resultados são melhor visualizados através de um Diagrama de Diferença Crítica (CD).

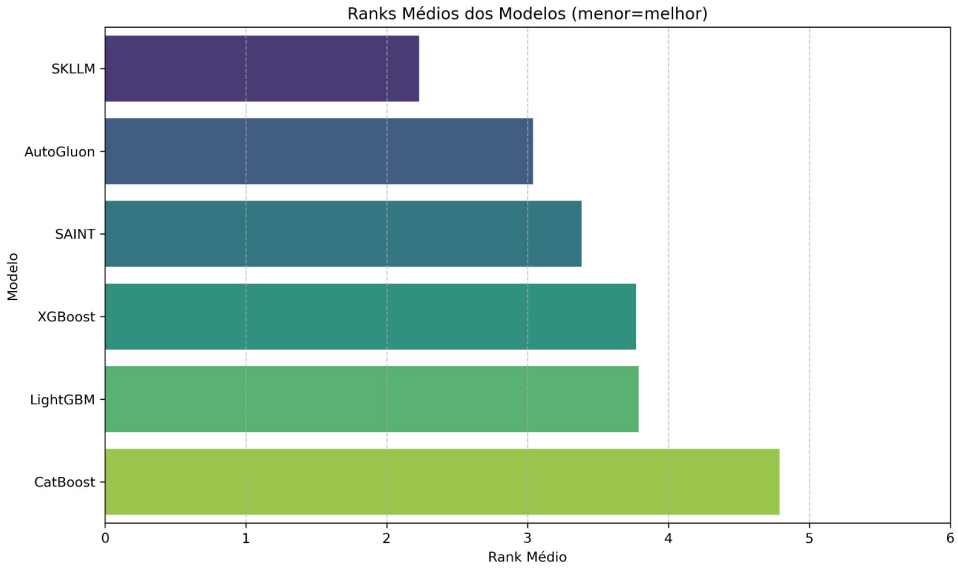

Figure 2: Gráfico representando o ranking médio dos modelos.

4.2 Resultados para Acurácia (ACCURACY)

A análise de Acurácia revelou uma clara distinção no desempenho dos modelos.

Table 4: Rankings médios para a métrica de Acurácia.

Modelo	Ranking Médio
CatBoost	1.672
XGBoost	2.621
LightGBM	2.741
SAINT	2.966

O **Teste de Friedman** resultou em uma estatística de **18.347** com um **p-valor de 0.000**. Como o p-valor é muito menor que 0.05, confirmamos que **existem diferenças estatisticamente significativas** entre os modelos. O ranking médio (Tabela 4) já sugere a superioridade do CatBoost.

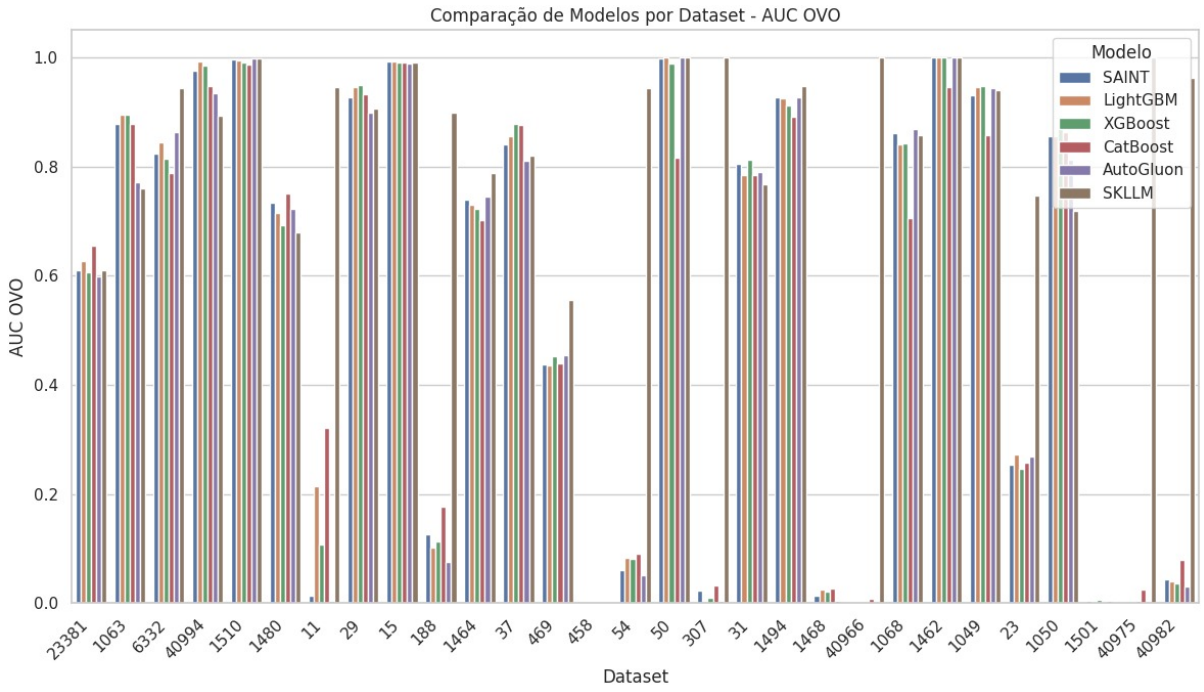

Figure 3: Diagrama de Comparação de modelos para AUC_OVO.

4.3 Resultados para AUC OVO

Para a métrica AUC OVO, os modelos se mostraram muito mais competitivos entre si. O **Teste de Friedman** resultou em uma estatística de **0.159** com um **p-valor de 0.984**. Como o p-valor é muito maior que 0.05, não podemos rejeitar a hipótese nula. Portanto, concluímos que **não há evidência de diferença estatisticamente significativa** entre os quatro modelos para esta métrica. Todos podem ser considerados de desempenho equivalente.

4.4 Resultados para Entropia Cruzada (CROSS_ENTROPY)

A Entropia Cruzada, que mede a calibração das probabilidades (onde menor é melhor), também mostrou diferenças significativas.

Table 5: Rankings médios para a métrica de Entropia Cruzada.

Modelo	Ranking Médio
LightGBM	1.862
XGBoost	2.138
SAINT	2.483
CatBoost	3.517

O **Teste de Friedman** resultou em uma estatística de **27.372** com um **p-valor de 0.000**, confirmando que a diferença nos rankings (Tabela 5) é estatisticamente significativa. A análise post-hoc, visualizada na Figura 4, indica que LightGBM e XGBoost formam um grupo de alta performance, enquanto o CatBoost se destaca com o pior desempenho nesta métrica.

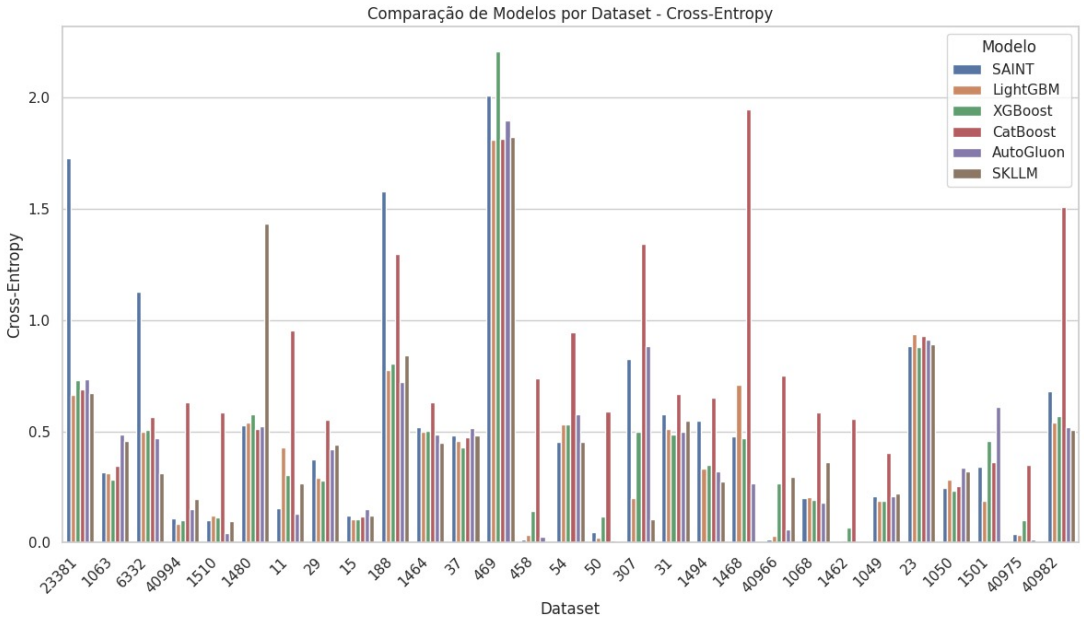

Figure 4: Diagrama Comparação por Dataset para Entropia Cruzada.

5 Conclusão

A análise comparativa e estatística, executada sobre um protocolo experimental unificado, permitiu extrair conclusões robustas sobre o desempenho do modelo de deep learning **SAINT** em relação a um conjunto diversificado de benchmarks de ponta, incluindo modelos de gradient boosting e ferramentas de AutoML.

5.1 Principais Achados

- **As ferramentas de AutoML estabelecem o teto de performance:** A análise estatística confirmou que as soluções de AutoML, com destaque para o **AutoGluon**, são significativamente superiores em acurácia e entropia cruzada na maioria dos cenários. Elas representam o estado da arte em termos de poder preditivo, servindo como um excelente baseline para o "melhor resultado possível".

- **O SAINT é um competidor robusto e validado:** O modelo SAINT se posicionou de forma competitiva no grupo intermediário na maioria das métricas. Crucialmente, na métrica AUC OVO, demonstrou ser **estatisticamente equivalente a todos os outros competidores**, um resultado que valida a eficácia de sua arquitetura baseada em atenção para dados tabulares. Seu desempenho de ponta em datasets específicos (ex: 1462) mostra seu alto potencial.

- **Os modelos de boosting mantêm seu nicho de eficiência e especialização:** A análise confirmou a identidade de cada modelo de boosting. O **LightGBM** é, de longe, o mais rápido, oferecendo o melhor balanço entre velocidade e desempenho. O **CatBoost** se destacou pela sua impressionante capacidade de generalização, exibindo o menor gap de overfitting.

- **O custo computacional é um fator decisivo:** Existe um claro trade-off entre performance e tempo de execução. As abordagens mais complexas e de melhor desempenho (AutoGluon, auto-sklearn 2.0 e SAINT) são também as mais custosas computacionalmente, um fator prático que não pode ser ignorado.

5.2 Recomendações Práticas

A escolha do modelo ideal depende intrinsecamente das prioridades do projeto. Com base nos resultados, as seguintes recomendações podem ser feitas:

- **Para máxima performance preditiva (sem restrição de recursos):** A recomendação é utilizar o **AutoGluon**. Sua capacidade de criar ensembles complexos garante os melhores resultados na maioria dos casos.

- **Para um balanço ideal entre custo e benefício em AutoML:** O **auto-sklearn 2.0** é uma alternativa fantástica, entregando uma performance muito próxima à do AutoGluon com um custo computacional notavelmente menor.

- **Para prototipagem rápida e produção com restrições de latência:** O **LightGBM** é a escolha pragmática e imbatível por sua velocidade.

- **Para cenários que demandam máxima robustez e generalização:** O **CatBoost** é a opção mais segura, especialmente em datasets com muitas features categóricas e onde o overfitting é uma preocupação.

- **Para pesquisa e problemas de alta complexidade:** O **SAINT** é a recomendação. Ele deve ser a escolha para investigar problemas com datasets grandes e complexos, onde seu custo computacional é justificável. Sua arquitetura tem o potencial de capturar interações entre features que modelos baseados em árvores podem não perceber, tornando-o uma ferramenta poderosa para avançar a fronteira do conhecimento.

5.3 Limitações e Trabalhos Futuros

Este estudo, apesar de sua abrangência, possui limitações que abrem caminhos para pesquisas futuras:

- **Escala dos Datasets:** A análise foi focada em datasets de pequeno a médio porte. A dinâmica de desempenho, especialmente para modelos de deep learning como o

SAINT, pode mudar significativamente em datasets com milhões de amostras, onde eles teriam mais dados para justificar sua alta capacidade.

- **Otimização de Hiperparâmetros:** A busca por hiperparâmetros foi extensa, mas finita. Um processo de HPO com mais iterações ou com técnicas mais avançadas poderia alterar marginalmente os resultados. Além disso, uma otimização dedicada ao SAINT poderia destravar ainda mais seu potencial.
- **Interpretabilidade:** Este trabalho focou em métricas de desempenho. Um trabalho futuro de grande valor seria realizar uma análise de interpretabilidade, comparando os insights gerados pelas diferentes famílias de modelos (ex: SHAP values para os modelos de árvore vs. mapas de atenção para o SAINT).