Universidade Católica de Brasília

Pró-Reitoria Acadêmica
Curso de Ciência da Computação
Novas Tecnologias

AT2/N2: Desafio Titanic

Autor: João Victor Silva Costa
Orientador: Prof. Dr. William Roberto Malvezzi

Brasília - DF
2025

JOÃO VICTOR SILVA COSTA

AT2/N2: Desafio Titanic

Documento apresentado ao Curso de graduação de Bacharelado em Ciência da Computação da Universidade Católica de Brasília, como requisito parcial para obtenção da aprovação na disciplina Novas Tecnologias.

Orientador: Prof. William Roberto Malvezzi

BRASÍLIA

2025

SUMÁRIO

1. INTRODUÇÃO

Este relatório descreve o processo de desenvolvimento de uma solução de Machine Learning para a competição "Titanic - Machine Learning from Disaster", hospedada na plataforma Kaggle. O desafio consiste em criar um modelo preditivo capaz de determinar a probabilidade de sobrevivência de um passageiro com base em um conjunto de atributos individuais e de viagem.

O trabalho foi estruturado seguindo as melhores práticas de engenharia de software e ciência de dados, desde a análise exploratória dos dados e engenharia de features até a otimização, treinamento e avaliação de múltiplos algoritmos. A solução final consiste em um modelo ensemble do tipo Stacking, que combina as predições dos modelos de melhor desempenho para alcançar maior acurácia e robustez. Adicionalmente, foi desenvolvida uma aplicação interativa com Streamlit (app/app.py) para permitir a exploração dos resultados e a realização de predições em tempo real.

2. METODOLOGIA

A metodologia adotada foi além dos requisitos básicos do desafio, implementando um pipeline de modelagem avançado e reprodutível. Todas as constantes, como caminhos de arquivos e modelos a serem testados, foram centralizadas no arquivo src/config/settings.py

2.1. ANÁLISE EXPLORATÓRIA DE DADOS (EDA)

A análise exploratória foi conduzida no notebook notebooks/eda.ipynb e teve como objetivo aprofundar o entendimento sobre as variáveis e suas correlações com a variável alvo (Survived). Foram investigadas as distribuições de dados, a presença de valores ausentes e as relações entre os atributos. Os principais insights que guiaram a etapa de pré-processamento foram:

1. ***Taxa de Sobrevivência por Sexo:*** Confirma-se a hipótese histórica de que passageiras do sexo feminino tiveram uma taxa de sobrevivência expressivamente maior.
2. ***Influência da Classe Social:*** Passageiros da primeira classe (Pclass = 1) tiveram maior probabilidade de sobreviver em comparação com os da segunda e, principalmente, da terceira classe.
3. ***Impacto de Idade e Família:*** Crianças e passageiros acompanhados por familiares apresentaram taxas de sobrevivência distintas, sugerindo que as features Age, SibSp e Parch são preditores importantes.

2.2. PRÉ-PROCESSAMENTO E ENGENHARIA DE FEATURES AVANÇADA

A preparação dos dados foi realizada pela função advanced_feature_engineering no módulo src/processing/preprocessor.py. As seguintes transformações foram aplicadas:

1. ***Tratamento de Dados Ausentes:*** *Age* e *Fare*: Valores nulos foram preenchidos com a mediana da respectiva coluna para minimizar a distorção da distribuição; *Embarked*: Valores nulos foram imputados com a moda (porto de embarque mais comum).
2. **Engenharia de Features**: *Title*: O título de cada passageiro (ex: 'Mr', 'Miss', 'Mrs') foi extraído da coluna Name. Títulos raros foram agrupados na categoria 'Other' para evitar a alta cardinalidade; *FamilySize*: Criada a partir da soma de SibSp (irmãos/cônjuges) e Parch (pais/filhos) mais 1, representando o tamanho total da família do passageiro a bordo; *IsAlone*: Feature binária derivada de FamilySize, indicando se o passageiro viajava sozinho (FamilySize == 1); *AgeGroup* e *FareBin*: As variáveis contínuas Age e Fare foram discretizadas em faixas (bins), transformando-as em variáveis categóricas ordinais para capturar relações não-lineares.

As *features* originais Name, Ticket, Cabin, SibSp e Parch foram descartadas após a extração das novas informações.

2.3. ESTRATÉGIA DE MODELAGEM

A modelagem foi estruturada em um pipeline robusto (*src/models/train_model.py*) que automatiza a otimização de hiperparâmetros, o treinamento e a seleção do melhor modelo.

2.3.1. Otimização de Hiperparâmetros com Optuna

Para cada um dos algoritmos base definidos em settings.py (RandomForestClassifier, GradientBoostingClassifier, XGBClassifier, LGBMClassifier, SVC), foi conduzido um processo de otimização de hiperparâmetros utilizando a biblioteca Optuna. O objetivo era encontrar a combinação de parâmetros que maximizasse a acurácia média em uma validação cruzada estratificada de 5 folds (StratifiedKFold), que preserva a proporção da variável alvo em cada fold. Esse processo foi paralelizado com ThreadPoolExecutor para maior eficiência computacional.

2.3.2. Validação e Construção do Modelo Ensemble

Após a otimização, os três modelos com a maior acurácia na validação cruzada foram selecionados como estimadores de base para o modelo final. O modelo final é um StackingClassifier, uma técnica de ensemble que utiliza as predições dos modelos de base como features para treinar um meta-estimador (neste caso, LogisticRegression). Essa abordagem visa combinar as forças de diferentes algoritmos para produzir predições mais estáveis e precisas.

O StackingClassifier final foi então treinado com todos os dados de treino e salvo como o artefato ensemble_model.joblib, juntamente com o pipeline de pré-processamento e um arquivo JSON (metrics.json) contendo as métricas de performance de todos os modelos avaliados.

3. RESULTADOS E DISCUSSÃO

3.1. PERFORMANCE DOS MODELOS

A performance do modelo foi quantificada pela acurácia na validação cruzada, cujos resultados detalhados foram salvos no artefato *training_metrics.json*. A tabela abaixo resume o desempenho dos principais algoritmos e do modelo ensemble final.

Modelo	***Acurácia Média***	***Desvio Padrão (+/-)***
GradientBoosting	0.8541	0.0103
XGBClassifier	0.8507	0.0166
LGBMClassifier	0.8462	0.0118
Stacking Ensemble (Final)	0.8563	0.0135

O modelo ensemble do tipo StackingClassifier, que utiliza GradientBoosting, XGBClassifier e LGBMClassifier como estimadores de base, obteve a maior acurácia média, com 0.8563. Este resultado valida a abordagem de combinar múltiplos modelos para obter uma predição mais robusta e generalizável.

Após a submissão do arquivo de predições, o score atingido na plataforma foi de 0.77272 e a colocação alcançada no *leaderboard* público foi de 9721.

3.2. DISCUSSÃO

A acurácia de 0.8563 obtida na validação cruzada demonstra a eficácia do pipeline de pré-processamento avançado e da estratégia de ensemble. A pequena margem de melhoria do modelo Stacking sobre o melhor modelo individual (GradientBoosting, com 0.8541) sugere que os modelos base possuem predições correlacionadas, mas a combinação ainda assim foi capaz de agregar valor.

No entanto, a pontuação obtida no leaderboard do Kaggle foi de ***0.77272***, um resultado notavelmente inferior à performance na validação cruzada. Essa discrepância é um sintoma clássico de overfitting: o modelo aprendeu padrões e ruídos específicos do conjunto de treino que não se generalizaram para o conjunto de teste, que era desconhecido.

Os principais desafios enfrentados foram o balanceamento entre complexidade e generalização, onde a utilização de modelos potentes (como GradientBoosting e XGBoost) e um *ensemble* StackingClassifier aumentou a capacidade do modelo de se ajustar aos dados de treino, mas também o risco de overfitting. O desafio foi em encontrar o ponto ótimo que maximize a performance sem decorar os dados de treino.

Os resultados expõem um dos aprendizados mais relevantes em ciência de dados: o paradoxo da complexidade e a importância da generalização. A acurácia de ***0.8563*** obtida na validação cruzada demonstrou, inicialmente, a alta capacidade do *pipeline* de se ajustar aos dados de treino. Contudo, a performance no *leaderboard* do Kaggle, de ***0.77272***, revelou, como dito anteriormente, uma queda significativa, sintoma clássico de *overfitting*.

Tal fenômeno pode ser explicado pelo princípio da ***Navalha de Occam*** aplicado ao Machine Learning, que postula que modelos mais simples frequentemente generalizam melhor. O projeto, em sua busca por uma alta performance, incorporou uma complexidade considerável em duas frentes principais: Engenharia de Features Avançada e Ensemble Sofisticado. A criação de *features* como Title, AgeGroup e FareBin permitiu ao modelo capturar padrões detalhados nos dados de treino. No entanto, essa especificidade fez com que o modelo aprendesse "ruídos" e relações que não eram válidas para o conjunto de teste, prejudicando sua capacidade de generalização. A implementação de um StackingClassifier sobre os três melhores modelos individuais (GradientBoosting, XGBClassifier e LGBMClassifier) representa um aumento substancial na complexidade. A análise das métricas revela o retorno decrescente dessa abordagem: o *ensemble* (acurácia de 0.8563) ofereceu uma melhoria marginal de apenas 0.0022 sobre o melhor modelo individual (GradientBoosting, com 0.8541). Esse pequeno ganho local foi o primeiro indicador de que a complexidade adicionada não se traduziria em robustez, mas sim em um ajuste excessivo aos dados de treino.

4. CONCLUSÃO

Este projeto cumpriu com sucesso o objetivo do desafio Titanic, desenvolvendo um modelo preditivo de alta performance. O processo permitiu aplicar na prática um ciclo completo de ciência de dados, desde a análise inicial até a implementação de um sistema de modelagem avançado e automatizado.

Os principais aprendizados foram a importância da engenharia de features criativa e o poder das técnicas de ensemble para aumentar a robustez e a precisão dos modelos. A estrutura do código, modularizada e configurável, garante a reprodutibilidade dos resultados e facilita a experimentação futura. A solução final não apenas atende aos requisitos do desafio, mas os excede, constituindo um sistema de classificação completo e eficaz.

REFERÊNCIAS

KAGGLE. Titanic - Machine Learning from Disaster. Disponível em: https://www.kaggle.com/competitions/titanic. Acesso em: 12 jun. 2025.