Universidade de Brasília

Faculdade UnB Gama

FGA0083 - Aprendizado de Máquina	**Semestre**: 2025.1
Docente: Sérgio Antônio Andrade	**Turma:** 01
Grupo 05	
Integrantes: Harryson Campos Martins Pedro Henrique Muniz de Oliveira Flávio Gustavo Araújo de Melo Leandro de Almeida Oliveira Jefferson Sena Oliveira José André Rabelo Rocha	**Matrícula:** 211039466 200059947 211030602 211030827 200020323 211062016

Mini Trabalho 8:

Lançamento, monitoramento e manutenção do sistema

Introdução

Este trabalho tem como objetivo apresentar as etapas de lançamento, monitoramento e manutenção do sistema desenvolvido durante o semestre. O sistema proposto visa prever os resultados das partidas do Campeonato Brasileiro a partir de dados históricos e informações sobre o desempenho recente das equipes.

A escolha do modelo Random Forest se deu por sua robustez, boa capacidade de generalização, resistência ao overfitting e desempenho consistente frente a dados numéricos e categóricos.

Este relatório descreve o processo de implantação da solução em um ambiente produtivo, detalha os mecanismos de monitoramento para garantir a estabilidade do modelo ao longo do tempo, e propõe estratégias de manutenção preventiva e adaptativa. Tais estratégias são essenciais para assegurar a eficácia preditiva do sistema diante de mudanças no comportamento dos dados e no contexto competitivo do campeonato.

Lançamento da Solução

O projeto desenvolvido tem como objetivo prever os resultados das partidas do Campeonato Brasileiro com base em dados históricos e desempenho recente das equipes. O modelo escolhido foi o Random Forest, devido à sua robustez, capacidade de lidar com dados categóricos e numéricos, resistência a overfitting e excelente desempenho em múltiplas métricas.

Etapas consideradas para o lançamento em ambiente produtivo:

- **Preparação do ambiente:** A implementação utiliza bibliotecas consolidadas como pandas, scikit-learn, matplotlib, seaborn e numpy, garantindo estabilidade, compatibilidade e suporte contínuo na maioria dos ambientes de produção Python.
- **Organização do pipeline:** O código está modularizado e organizado em blocos bem definidos: carregamento e validação de dados, engenharia de atributos (médias móveis, histórico recente), codificação, treinamento, avaliação e ajuste fino do modelo.
- **Separação treino/teste temporal:** O conjunto de dados foi separado respeitando a ordem cronológica, utilizando anos anteriores a 2023 para treinamento e o ano de 2023 para teste, simulando um fluxo real de predição futura.
- **Validação cruzada (K-Fold):** Foi implementada validação cruzada com 5 divisões para garantir a estabilidade e confiabilidade do modelo sob diferentes amostras, reduzindo o risco de overfitting e maximizando a generalização.

- **Preparação para integração:** O código é adaptável para encapsulamento em APIs REST (usando Flask ou FastAPI), e o modelo pode ser exportado com joblib para uso em sistemas externos, como dashboards ou aplicações web.

Plano de Monitoramento

Após o lançamento do sistema, é fundamental garantir a continuidade da qualidade preditiva do modelo. Para isso, o plano de monitoramento contempla:

- **Avaliação contínua com métricas-chave:** Métricas como Accuracy, Precision, Recall e F1-Score são calculadas durante e após o treinamento, permitindo acompanhar o desempenho geral do modelo ao longo do tempo.
- **Geração e análise da matriz de confusão:** Usada para verificar padrões de erro e possíveis tendências de enviesamento do modelo em certas classes (ex: superprevisão de vitórias).
- **Acompanhamento da importância das variáveis:** O monitoramento das features mais relevantes permite identificar alterações no padrão dos dados, auxiliando na detecção de data drift e na reavaliação do modelo.
- **Sugestão de ferramentas de monitoramento:** Para ambientes produtivos, recomenda-se o uso de ferramentas como MLflow, Evidently AI ou Prometheus/Grafana para registro contínuo de métricas e alertas automáticos de degradação.

Manutenção e Atualização Contínua

Para garantir a longevidade, eficácia e adaptabilidade da solução, foram definidas estratégias claras de manutenção com base nos modelos de Random Forest, Regressão Logística e SVM desenvolvidos para previsão de resultados do Campeonato Brasileiro, assim apresentamos quatro estratégias principais para garantir a eficácia e adaptabilidade dos modelos ao longo do tempo:

1. ATUALIZAÇÃO INCREMENTAL COM NOVOS DADOS DE TEMPORADAS

Estratégia: Implementar um pipeline automatizado de atualização do modelo ao final de cada rodada do campeonato, incorporando os novos resultados ao conjunto de treinamento.

Implementação:

- Criar um script de extração de dados que coleta automaticamente os resultados das partidas recentes das fontes mencionadas (Kaggle/GitHub).

- Executar a padronização e limpeza dos novos dados conforme o pipeline atual (limpeza_e_padronizacao_dos_dados.py).
- Retreinar os modelos (RandomForest.py, RegressaoLogistica_otimizado.py, svm.py) incorporando os dados mais recentes, com maior peso para temporadas recentes.
- Aplicar validação cruzada temporal (usando temporadas anteriores para prever a atual) a fim de verificar se o modelo mantém sua acurácia com os dados mais recentes.
- Documentar métricas de desempenho para monitorar possíveis deteriorações do modelo.

Benefícios:

- Adaptação contínua a mudanças no estilo de jogo, desempenho dos times e tendências do campeonato.
- Aumento gradual da base de dados de treinamento, melhorando a capacidade preditiva.
- Redução do "concept drift" (mudanças nas relações entre variáveis ao longo do tempo).

2. REVISÃO E ENGENHARIA DE FEATURES SAZONAL

Estratégia: Realizar uma análise profunda e revisão dos modelos ao final de cada temporada, com foco na engenharia de features e ajuste de hiperparâmetros.

Implementação:

- Conduzir uma análise estatística comparativa entre as previsões da temporada e os resultados reais.
- Identificar padrões emergentes e novas variáveis potencialmente preditivas (ex: mudanças de técnicos, contratações importantes, lesões).
- Criar novas features derivadas que capturem tendências recentes (ex: desempenho nos últimos N jogos, aproveitamento contra adversários de níveis específicos).
- Realizar nova otimização de hiperparâmetros nos três modelos (Random Forest, Regressão Logística e SVM).
- Implementar técnicas de ensemble adaptativas que possam ajustar dinamicamente o peso de cada modelo base conforme seu desempenho recente.

Benefícios:

- Adaptação estrutural do modelo a mudanças fundamentais no campeonato.
- Incorporação de novos conhecimentos do domínio e variáveis emergentes.
- Melhoria contínua através da evolução do conjunto de features.

3. RECALIBRAÇÃO DE HIPERPARÂMETROS PERIÓDICA

Estratégia: Estabelecer um cronograma de recalibração sistemática dos hiperparâmetros dos modelos, independente das mudanças de temporada, para garantir a otimização contínua.

Implementação:

- Definir janelas de tempo específicas (por exemplo, a cada 10 rodadas) para reavaliação automática dos hiperparâmetros.
- Implementar técnicas de otimização bayesiana ou algoritmos genéticos para busca eficiente no espaço de hiperparâmetros.
- Manter um histórico de configurações de hiperparâmetros e seus respectivos desempenhos para identificar tendências.
- Criar um pipeline automatizado que execute grid search ou random search em paralelo, sem interferir nas predições em produção.
- Implementar técnicas de transferência de aprendizado para aproveitar conhecimentos de modelos anteriores na calibração.

Benefícios:

- Manutenção da eficiência computacional dos modelos com configurações sempre otimizadas.
- Adaptação a mudanças sutis nas relações entre variáveis que ocorrem sem alterações estruturais visíveis.
- Prevenção contra overfitting gradual que pode ocorrer com o acúmulo de dados históricos.

4. DETECÇÃO DE DEGRADAÇÃO DE PERFORMANCE

Estratégia: Implementar um sistema de monitoramento contínuo que identifique proativamente quedas de desempenho nos modelos antes que afetem significativamente as predições.

Implementação:

- Desenvolver um dashboard de monitoramento com alertas automáticos quando métricas-chave (acurácia, F1-score, log loss) caírem abaixo de limiares predefinidos.
- Implementar testes estatísticos (como teste de Kolmogorov-Smirnov) para comparar distribuições de previsões recentes com distribuições históricas.
- Criar um pipeline de backtesting que simule o desempenho dos modelos em diferentes janelas de tempo para identificar degradações progressivas.
- Estabelecer métricas específicas para subgrupos de dados (jogos entre grandes times, clássicos, jogos decisivos) para detectar degradações em contextos específicos.
- Desenvolver um sistema de versão/rollback que permita reverter rapidamente para uma versão anterior do modelo em caso de degradação severa.

Benefícios:

- Identificação precoce de problemas antes que afetem significativamente o desempenho preditivo.
- Análise específica de contextos onde o modelo pode estar perdendo capacidade preditiva.
- Capacidade de resposta rápida a mudanças bruscas no padrão do campeonato (como ocorreu durante a pandemia).

A combinação destas quatro estratégias - atualização incremental, revisão sazonal, recalibração periódica e detecção de degradação - forma um sistema completo de manutenção que aborda aspectos preventivos, corretivos e adaptativos, garantindo que os modelos se mantenham relevantes e precisos ao longo do tempo, adaptando-se tanto a mudanças graduais quanto a transformações mais significativas no contexto do Campeonato Brasileiro.

Essas estratégias devem ser acompanhadas de um registro sistemático das métricas de desempenho para documentar a evolução dos modelos e identificar pontos de melhoria contínua.

Integração com Sistemas Existentes

A solução foi pensada para ser facilmente integrável a sistemas externos, tais como:

APIs REST: O modelo pode ser encapsulado via Flask ou FastAPI, permitindo seu uso por aplicações web ou mobile em tempo real.

Exportação de modelo treinado: Com uso de joblib, o modelo pode ser salvo e carregado facilmente em sistemas externos, sem necessidade de retreinamento, acelerando a entrega do serviço.

Automatização do fluxo: Com ferramentas como Airflow, cron ou scripts agendados, é possível automatizar a ingestão de novos dados, reavaliação do modelo e atualização de métricas.

Testes de Estabilidade e Segurança

A solução foi pensada para ser facilmente integrável a sistemas externos, tais como:

Testes de Estabilidade

Testes de estresse: Simulações com grande volume de requisições à API do modelo para avaliar o comportamento sob carga.

Testes de robustez: Inserção de dados malformados para verificar se o sistema lida corretamente com entradas inesperadas.

Monitoramento de tempo de resposta: Avaliação do desempenho do sistema sob diferentes volumes e condições de uso.

Segurança do Sistema

Validação de entrada: Filtragem rigorosa das requisições para evitar entradas inválidas ou perigosas.

Controle de versões: Histórico de versões do modelo para permitir rollback rápido em caso de falha.

Logs e auditoria: Registro detalhado das interações e atualizações para rastreamento e análise de erros.

Autenticação (opcional): Restrição de acesso às APIs via tokens ou chaves para ambientes mais controlados.

Conclusão

A implementação do sistema preditivo para prever os resultados do Campeonato Brasileiro demonstrou-se eficaz tanto em termos de desempenho quanto de escalabilidade. No entanto, mais do que o desenvolvimento inicial do modelo, é fundamental garantir sua continuidade operacional por meio de estratégias robustas de monitoramento e manutenção.

As abordagens propostas como a atualização incremental com novos dados, a revisão sazonal das features, a recalibração periódica dos hiperparâmetros e a detecção proativa de degradação, dessa maneira formando um ciclo completo que assegura a adaptabilidade e longevidade da solução.

Além disso, a capacidade de integração com APIs e ferramentas de automação torna o sistema apto para ambientes produtivos reais. Dessa forma, este trabalho evidencia não apenas a construção de um modelo preditivo funcional, mas também a importância de práticas contínuas de supervisão e evolução, essenciais em projetos de aprendizado de máquina aplicados a contextos dinâmicos.