# Implementação e Avaliação de Modelos de Aprendizado Profundo para Otimização de Inventário, Detecção e Tratamento de Anomalias

Igor Mol

igor.mol@makes.ai

13 de novembro de 2023

## Sumário



## 1 Redes Neurais Recorrentes

O código–fonte `RNN.py` visa resolver um problema de otimização de inventário por meio da aplicação de um modelo de aprendizado profundo, utilizando uma Rede Neural Recorrente (RNN) implementada com TensorFlow e Keras. A estrutura do código está modularizada na classe `InventoryOptimizer` para facilitar a compreensão e promover a reutilização.

## Inicialização e Configuração

A classe `InventoryOptimizer` é inicializada com o conjunto de dados de inventário. Os parâmetros essenciais, como o comprimento da sequência (`sequence_length`), um objeto `MinMaxScaler` para normalização dos dados, e o modelo RNN são configurados durante a inicialização.

## Pré-processamento de Dados

A função `preprocess_data` realiza a normalização dos dados e gera sequências para o treinamento da RNN. Os dados são normalizados para o intervalo $[0, 1]$ usando o `MinMaxScaler`. Sequências de entrada e os rótulos correspondentes são criados a partir dos dados normalizados.

## Construção do Modelo RNN

A função `build_model` constrói o modelo RNN utilizando a biblioteca Keras. O modelo consiste em uma camada RNN com ativação ReLU e uma camada densa de saída. A função de perda utilizada é o erro quadrático médio (MSE), e o otimizador é o Adam.

## Treinamento do Modelo

A função `train_model` treina o modelo RNN com os dados de treinamento. Os parâmetros padrão são configurados para 50 épocas e um tamanho de lote de 32. O histórico de treinamento, incluindo as métricas de MSE para o conjunto de treinamento e validação, é retornado.

## Avaliação do Modelo

A função `evaluate_model` avalia o desempenho do modelo nos dados de teste, imprimindo o MSE resultante como uma medida quantitativa da qualidade do modelo em dados não vistos.

## Geração de Previsões

A função `predict` gera previsões para os dados de teste utilizando o modelo treinado. As previsões são desnormalizadas para a escala original dos dados.

## Visualizações e Resultados

As funções `plot_results`, `plot_mse_trend`, e `plot_loss_function` são responsáveis por gerar gráficos que visualizam os resultados do modelo. A função `print_results_table` imprime os resultados da otimização de inventário em uma tabela.

### Execução do Programa

A função `main` carrega os dados do inventário, instancia a classe `InventoryOptimizer`, realiza o pré-processamento dos dados, treina o modelo, avalia o desempenho, gera previsões e cria visualizações dos resultados. Este padrão de execução promove a organização do código e facilita a reutilização em diferentes cenários de otimização de inventário.

## 2 Otimização de Inventário com Redes Neurais Convolucionais (CNN)

O código–fonte `CNN.py` implementa um otimizador de inventário por meio de uma abordagem de aprendizado profundo utilizando Redes Neurais Convolucionais (CNNs). A classe `InventoryOptimizer` encapsula todas as funcionalidades do sistema.

### Inicialização e Carregamento de Dados

O objeto é inicializado com o caminho do arquivo CSV contendo os dados do inventário. A função `load_data` carrega os dados utilizando a biblioteca pandas, retornando um DataFrame.

### Pré-processamento de Dados

A função `preprocess_data` normaliza os dados, escalando as colunas '`time_scale`', '`filial_id`' e '`quant_item`' para o intervalo $[0, 1]$ por meio da classe `MinMaxScaler` do scikit-learn.

### Construção da CNN

A função `build_model` constrói a arquitetura da Rede Neural Convolucional (CNN) utilizando a API do Keras. A CNN é composta por uma camada de convolução unidimensional, uma camada de achatamento (flatten), e duas camadas densas (fully connected).

### Treinamento do Modelo

O método `train_model` realiza o treinamento da CNN, registrando as tendências de Mean Squared Error (MSE) e perda (loss) ao longo das épocas.

### Avaliação do Modelo

A função `evaluate_model` avalia o desempenho do modelo treinado sobre o conjunto de teste, calculando e exibindo o MSE.

### Geração de Previsões

O método `predict` realiza predições utilizando a CNN sobre o conjunto de teste.

### Inversão da Escala de Normalização

A função `inverse_transform` efetua a inversão da escala de normalização, convertendo as predições para a escala original.

### Visualizações e Resultados

As funções `plot_actual_vs_predicted`, `plot_mse_trend`, `plot_loss_trend`, e `print_results_table` geram visualizações dos resultados. A tabela é apresentada utilizando a biblioteca tabulate, seguida por gráficos que ilustram a relação entre as quantidades reais e previstas, a tendência geral do MSE e a evolução da função de perda durante o treinamento.

### Execução do Programa

A função `main` carrega os dados do inventário, realiza o pré-processamento, treina a CNN, avalia o desempenho, gera previsões e cria visualizações dos resultados. Este padrão de execução organiza o código e facilita a compreensão em diferentes cenários de otimização de inventário.

## 3 Otimização de Inventário com Abordagem Híbrida CNN-RNN

O código–fonte Hybrid_CNN_RNN.py apresenta uma solução para o problema de otimização de inventário por meio de uma abordagem híbrida que combina Redes Neurais Convolucionais (CNN) e Recorrentes (RNN). O sistema é encapsulado na classe OtimizadorInventario, projetada para proporcionar modularidade e compreensão do código.

### Inicialização e Carregamento de Dados

A classe é inicializada com informações cruciais, como o caminho do arquivo de dados, o número de passos temporais e o número de recursos. Esses parâmetros oferecem flexibilidade para ajustar o comportamento do otimizador. A função carregar_dados é responsável por ler os dados tabulares contidos no arquivo CSV especificado.

## Normalização dos Dados

A normalização dos dados, realizada pela função normalizar_dados, utiliza a classe MinMaxScaler da biblioteca scikit-learn para dimensionar os atributos do conjunto de dados para o intervalo [0, 1]. Essa etapa é crucial para aprimorar a convergência e o desempenho do modelo durante o treinamento.

## Geração de Sequências

A função criar_sequencias é empregada para gerar sequências temporais apropriadas a partir dos dados. Cada sequência é formada por uma janela deslizante de passos temporais, onde os rótulos representam a quantidade de itens na posição temporal subsequente.

## Preparação das Sequências

A preparação das sequências para treinamento é realizada pela função preparar_sequencias, que extrai as colunas relevantes do DataFrame e ajusta as dimensões das sequências de acordo com as expectativas do modelo híbrido CNN-RNN.

## Construção do Modelo

A função construir_modelo define a arquitetura do modelo híbrido CNN-RNN utilizando a biblioteca TensorFlow e Keras. Essa arquitetura inclui uma camada convolucional 1D seguida por uma camada de pooling, uma camada LSTM e, por fim, uma camada densa de saída.

## Treinamento do Modelo

O treinamento do modelo é executado pela função treinar_modelo, que utiliza o otimizador Adam e a métrica de erro quadrático médio (MSE) para ajustar os pesos do modelo às sequências de treinamento. A função fazer_predicoes gera previsões utilizando o modelo treinado nas sequências de treinamento.

## Avaliação do Desempenho

As métricas de desempenho, como MSE, MAE, R2 Score e Residuais, são calculadas pela função calcular_metricas. Essas métricas proporcionam uma avaliação abrangente do quão bem o modelo está performando.

### Apresentação e Análise dos Resultados

Para uma análise detalhada, a função imprimir_tabela_resultados utiliza a biblioteca tabulate para apresentar os resultados de forma estruturada e legível em formato de tabela. Além disso, são gerados gráficos visuais, como o gráfico de valores reais versus previstos e as tendências do MSE e da função de perda ao longo das épocas de treinamento.

### Execução do Programa

A função main orquestra o processo de otimização de inventário utilizando o modelo híbrido de CNN e RNN. O caminho do arquivo é especificado, e um objeto da classe OtimizadorInventario é instanciado para realizar operações subsequentes, incluindo carregamento de dados, normalização, preparação de sequências, treinamento do modelo, geração de previsões e análise dos resultados. Os resultados finais são consolidados em um DataFrame que encapsula previsões, rótulos reais e métricas de desempenho.

## 4 Solução Híbrida de Aprendizado Profundo para Otimização de Inventário

Este programa Hybrud_RNN_LSTM.py oferece uma solução para o desafiador problema de otimização de inventário, utilizando uma abordagem híbrida de aprendizado profundo. A estratégia adotada combina Redes Neurais Convolucionais (CNN) e Redes Neurais do tipo Long Short-Term Memory (LSTM).

### Inicialização e Preprocessamento de Dados

A classe InventoryOptimizer encapsula as funcionalidades do programa. O construtor inicializa variáveis cruciais, enquanto a função load_data carrega dados tabulares de um arquivo CSV especificado e os ordena com base na coluna 'time_scale'. O preprocessamento inclui a extração de features e rótulos, normalização e divisão dos dados em conjuntos de treinamento e teste.

### Construção do Modelo

A arquitetura do modelo é construída com uma camada Conv1D, seguida por uma camada Flatten, Reshape e uma LSTM. O modelo é compilado utilizando o otimizador 'adam' e a função de perda 'mse' (Mean Squared Error).

### Treinamento e Predição

A função train_model executa o treinamento do modelo com os dados de treinamento, retornando o histórico para análise. A predição é realizada pela função predict, que inverte a escala para obter valores reais.

### Avaliação do Desempenho

Métricas como desvio padrão, erro estatístico e resíduos são calculados pela função calculate_metrics com base nas previsões e nos valores reais.

### Visualização e Análise dos Resultados

Resultados são apresentados de maneira formatada por meio da biblioteca Tabulate. Gráficos de dispersão e tendência do Mean Squared Error (MSE) ao longo das épocas fornecem insights visuais sobre a eficácia do modelo.

### Execução do Programa

Na função principal main, a classe InventoryOptimizer é instanciada, dados são carregados, pre-processados, o modelo é construído e treinado. Previsões são feitas, métricas são calculadas e os resultados são exibidos por meio de tabelas e gráficos, proporcionando uma visão abrangente do desempenho da solução híbrida proposta.

## 5 Otimização de Inventário com Redes Neurais Recorrentes (RNNs) e GRU

O código–fonte `GRU.py` apresenta uma solução para o problema de otimização de inventário, adotando Redes Neurais Recorrentes (RNNs) com uma arquitetura de Unidade Recorrente Gated (GRU).

### Inicialização e Pré-processamento

A classe InventoryOptimizer é projetada para otimizar o inventário, com parâmetros cruciais, incluindo o caminho do arquivo CSV, o número de passos temporais, unidades GRU, épocas e tamanho do lote. A normalização dos dados, realizada pelo método load_data, é essencial para melhorar a convergência do modelo.

### Construção e Treinamento do Modelo

A arquitetura da rede é construída com uma camada GRU e uma camada densa. O treinamento, executado por train_model, utiliza dados de treinamento e validação. O acompanhamento do histórico permite análises detalhadas do desempenho do modelo ao longo das épocas.

### Predição e Avaliação

A função predict gera previsões para os dados de teste. A inversão da transformação normalizada é crucial para comparar valores previstos e reais. Resultados são apresentados de forma estruturada e podem ser salvos em um arquivo CSV para análises posteriores.

### Visualização e Análise de Resultados

A classe oferece métodos para plotar resultados, comparando a quantidade ótima com os dados reais. Gráficos da função de perda durante o treinamento fornecem insights sobre a convergência do modelo.

### Execução do Programa

A função principal main orquestra o processo de otimização de inventário. Os dados são carregados, pré-processados e divididos em conjuntos de treino e teste. O treinamento, predição e avaliação são conduzidos, e os resultados são organizados em estruturas de dados para análise e visualização abrangentes.

## 6 Otimização de Inventário com Floresta Aleatória

O código `Random_Forests.py` apresenta uma solução para o problema de otimização de inventário, utilizando a abordagem de Floresta Aleatória. Essa técnica baseia-se em um conjunto de árvores de decisão para prever a quantidade ótima de itens em estoque, fundamentada em dados históricos de vendas.

### Inicialização e Divisão dos Dados

A classe OtimizadorInventario é inicializada com um dataframe contendo informações relevantes. O método dividir_treino_teste extrai as features e o target, separando os dados em conjuntos de treino e teste, crucial para avaliar o desempenho do modelo.

### Treinamento do Modelo de Floresta Aleatória

A implementação utiliza a classe RandomForestRegressor da biblioteca scikit-learn. O método treinar_random_forest instância e treina o modelo com os dados de treino, permitindo ajustes como o número de árvores na floresta.

### Avaliação do Modelo

O método avaliar_modelo realiza previsões sobre os dados de teste e calcula o desvio padrão e o erro estatístico (MSE). Essas métricas são essenciais para compreender a precisão e a robustez do modelo.

### Visualização e Análise de Resultados

O código fornece funções para visualização, incluindo um gráfico de dispersão comparando valores reais e previstos e a tendência do MSE em relação ao número de árvores na floresta. Essas visualizações oferecem insights sobre a confiabilidade do modelo em diferentes configurações.

### Execução do Programa

O bloco principal orquestra a execução do programa. O arquivo CSV contendo os dados de inventário é carregado, e o modelo é treinado e avaliado. Os resultados são tabulados e visualizados, proporcionando uma análise abrangente e detalhada do desempenho do modelo de Floresta Aleatória.

## 7 Detecção de Anomalias em Inventário com Isolation Forest e CNN

O código `Anomaly_Detection.py` apresenta uma estratégia para a análise de inventário, focando na detecção e substituição de anomalias. Essa abordagem utiliza uma combinação de Redes Neurais Convolucionais (CNN) e Isolation Forest, proporcionando uma solução híbrida para identificar comportamentos anômalos nos dados.

### Inicialização e Carregamento de Dados

A classe InventoryAnalyzer é inicializada com o caminho do arquivo CSV contendo os dados de inventário. O método load_data inicia o processo carregando os dados brutos, um passo crucial para a subsequente análise. Pré-processamento de Dados

O método scale_data padroniza os dados de quantidade do item usando o StandardScaler do scikit-learn. Essa etapa prepara os dados para o treinamento do modelo CNN.

## Construção e Treinamento do Modelo CNN

A implementação inclui métodos para construir e treinar um modelo CNN. A arquitetura do modelo é configurada utilizando a biblioteca Keras/TensorFlow. O treinamento é executado para aprender padrões nos dados padronizados.

## Predição e Cálculo do Erro de Reconstrução

O código realiza previsões usando o modelo CNN treinado e calcula o erro de reconstrução entre as previsões e os dados padronizados. Esse erro é fundamental para a detecção de anomalias.

## Detecção de Anomalias com Isolation Forest

O método apply_isolation_forest utiliza o algoritmo Isolation Forest para rotular anomalias com base no erro de reconstrução. Essa abordagem é eficaz para identificar pontos de dados que se desviam significativamente do comportamento normal.

## Criação de DataFrames de Resultados e Limpeza

Os resultados são organizados em um DataFrame, incluindo informações temporais, filiais, quantidade de item, erro de reconstrução e rótulos de anomalia. Além disso, um DataFrame limpo é criado, substituindo valores de quantidade de item em casos de anomalias.

## Salvamento de Resultados e Visualização Gráfica

O código oferece funcionalidades para salvar dados limpos em um arquivo CSV e gerar gráficos ilustrativos do processo de detecção de anomalias. Os gráficos incluem uma comparação entre inventário limpo e dados reais, a tendência geral do erro médio quadrático (MSE) e a função de perda durante o treinamento do modelo CNN.

## Impressão da Tabela de Resultados

A tabela de resultados é impressa para fornecer uma visão tabulada das anomalias identificadas, incluindo os atributos temporais, filiais, quantidade de item, erro de reconstrução e rótulos de anomalia.