Implementação e Avaliação de Modelos de Aprendizado Profundo Híbridos e Transformadores na Modelagem de Séries Temporais, Detecção e Tratamento de Anomalias

Igor Mol

igor.mol@makes.ai

10 de novembro de 2023

Sumário

Parte I

Modelagem de Séries Temporais

1 Rede Neural de Convolução (CNN) e Suavização Exponencial

O código–fonte `Exponential_Smoothing_CNN_v2.py` apresenta uma implementação de um modelo de aprendizado profundo para a análise de séries temporais. O roteiro desse código apresenta uma abordagem híbrida de aprendizado profundo para modelagem de séries temporais. O objetivo é formar um modelo que possa prever de forma eficaz dados futuros com base na tendência e padrão temporal dos dados de séries temporais fornecidos. Após a importação dos pacotes necessários, o procedimento de preparação de dados é executado, que organiza os dados de entrada para o modelo de Rede Neural de Convolução (CNN). Posteriormente, a função responsável pela criação do modelo CNN é executada. Depois, é aplicado o método de suavização exponencial aos dados de treinamento. Na sequência, os dados são normalizados no intervalo $[0, 1]$ utilizando a função `MinMaxScaler` do pacote `sklearn`. O tamanho da janela para a preparação dos dados para o modelo CNN é definido e a função de preparação de dados é executada novamente. Uma instância do modelo CNN é criada e treinada nos dados preparados. O modelo é usado então para produzir previsões, que são transformadas inversamente para coincidir com a escala dos dados originais. Finalmente, o código apresenta uma comparação visual entre os dados originais da série temporal e as previsões do modelo por meio de um gráfico.

2 Rede Neural Recorrente (RNN) e Suavização Exponencial

O arquivo `Exponential_Smoothing_RNN_v2.py` implementa um método de aprendizado profundo híbrido combinando suavização exponencial e uma rede neural recorrente (RNN), para modelar uma série temporal chamada `training_data`. Ele começa importando as bibliotecas necessárias, como numpy, pandas, tensorflow, entre outras. As funções são definidas para executar suavização exponencial (`aplicar_suavizacao_exponencial`), preparar os dados para a RNN (`preparar_dados_rnn`) e criar e configurar o modelo RNN (`criar_rnn_model`). Após a definição das funções, a suavização exponencial é aplicada à coluna "`sum_quant_item`" do dataframe "`training_data`", seguida da nor-

malização dos dados suavizados para o intervalo $[0, 1]$ usando o `MinMaxScaler`. Os dados são então formatados para a RNN, definindo "`look_back`" igual a 1. Uma RNN é então criada e ajustada aos dados preparados durante 200 épocas com o tamanho do lote igual a 1. Uma vez ajustado, o modelo é usado para gerar previsões dos dados de treinamento, que são então transformados de volta para a escala original. O erro do modelo é calculado como a raiz quadrada do erro quadrático médio (RMSE) entre as predições e os valores verdadeiros. Após a computação, o gráfico é estilizado usando a biblioteca seaborn, e um gráfico de linha é gerado para comparar os dados originais e as previsões do modelo. O gráfico é aprimorado com legendas, título e rótulos de eixo para uma apresentação clara e profissional. O gráfico resultante é então exibido com `plt.show()`.

3 Mecanismo de Atenção: Transformadores

O código–fonte `Transformers.py` é um exemplo de um modelo de aprendizado profundo para série temporal usando Transformadores. Primeiro, ele normaliza os dados originais usando `MinMaxScaler` para reduzir a variação da série temporal. Em seguida, ele gera uma nova série temporal adequada aos Transformadores por meio da função "`create_dataset`". O argumento "`look_back`" especifica quantos passos de tempo anteriores o modelo deve considerar para prever o próximo valor. O próximo passo é definir a arquitetura do modelo Transformer. O modelo é construído utilizando blocos encoder de transformers que contém uma camada de normalização, uma camada de auto-atenção multi-cabeça e duas camadas de convolução $1D$. Dentro do modelo Transformer, a atenção dos cabeçotes múltiplos permite ao modelo considerar diferentes aspectos da entrada simultaneamente.mDepois que o modelo é definido, dividimos os dados em conjuntos de treinamento e teste. Em seguida, reformulamos esses conjuntos para se adequar à entrada esperada do modelo Transformer.

A seguir, especificamos os hiperparâmetros do modelo, como o tamanho da cabeça, o número de cabeças, a dimensionalidade da camada feed forward, a taxa de dropout e o número de blocos transformadores. Esses hiperparâmetros controlam a complexidade e capacidade do modelo. Depois de definir a arquitetura e os hiperparâmetros do modelo, o compilamos com uma perda de Erro Quadrático Médio e um otimizador Adam, que são padrão para tarefas de regressão. Em seguida, treinamos o modelo usando os dados de treinamento e avaliamos a perda de treinamento e validação em cada época, plotando-a em um gráfico.

Finalmente, o modelo é usado para fazer previsões sobre o conjunto de teste. As previsões são então inversamente transformadas para estar na mesma escala que os dados originais e, em seguida, são plotadas para comparação visual com os verdadeiros valores da série temporal.

4 Modelagem de Séries Temporais com Abordagem Híbrida ARIMA e RNN

O código–fonte `ARIMA_RNN.py` implementa uma abordagem avançada para a modelagem de séries temporais, integrando métodos clássicos de análise de séries temporais, representados pelo ARIMA (Médias Móveis Integradas AutoRegressivas), com técnicas de aprendizado profundo, especificamente as Redes Neurais Recorrentes (RNN). O objetivo principal deste código é realizar a previsão de valores futuros em uma série temporal, representada pelos dados contidos em "`training_data`". A estratégia adotada envolve a utilização do ARIMA para capturar padrões temporais de curto prazo e a implementação de uma Rede Neural Recorrente (LSTM) para modelar padrões de longo prazo e relações não lineares complexas.

Os dados são divididos em conjuntos de treino e teste, e em seguida, são normalizados usando a técnica `Min-Max Scaling` para garantir a estabilidade no treinamento dos modelos. O ARIMA é ajustado aos dados de treino para capturar padrões temporais. A ordem do modelo ARIMA é determinada automaticamente pela biblioteca `pmdarima`. Uma LSTM é construída e treinada usando as previsões do ARIMA. A escolha de uma LSTM visa explorar relações temporais mais complexas e a capacidade da rede de aprender dependências temporais de longo alcance. As previsões obtidas pelo ARIMA e LSTM são comparadas visualmente com os valores reais, proporcionando uma análise detalhada do desempenho dos modelos.

Esta abordagem híbrida busca combinar as vantagens de métodos tradicionais de séries temporais com as capacidades de aprendizado profundo. A combinação de ARIMA e LSTM possibilita a captura eficiente de padrões temporais em diferentes escalas, melhorando a capacidade preditiva do modelo em comparação com métodos isolados.

5 Modelagem de Séries Temporais com Abordagem Híbrida CNN e GRU

O arquivo `Hybrid_CNN_GRU.py` apresenta uma implementação de modelagem de séries temporais utilizando uma abordagem híbrida, combinando Redes Neurais Convolucionais (CNN) e Unidades Recorrentes de Gated (GRU). Esta técnica é empregada para prever futuros valores em uma série temporal representada pelos dados em "`training_data`".

Os dados de treinamento são extraídos e convertidos em uma matriz para facilitar o processamento. A normalização é realizada utilizando a técnica `Min-Max Scaling` para restringir os dados ao intervalo $[0, 1]$. Uma função é desenvolvida para criar pares de entrada e saída para o treinamento do modelo. Cada entrada é formada por uma sequência de observações temporais, e a saída corresponde aos valores futuros a serem previstos. Uma arquitetura de modelo é definida sequencialmente, começando com uma camada de Convolução 1D (CNN) para extração de características. Em se-

guida, são adicionadas camadas de Unidades Recorrentes de Gated (GRU), uma para sequências e outra para consolidar as informações. A camada de saída é composta por uma única unidade, refletindo o horizonte de previsão desejado. O modelo é compilado com o otimizador Adam e a função de perda sendo o erro quadrático médio. O treinamento é conduzido por 100 épocas, utilizando um tamanho de lote de 32. O histórico do treinamento é armazenado para análise posterior. Após o treinamento, o modelo realiza previsões nos dados de treinamento. As previsões são transformadas de volta para a escala original dos dados antes da normalização. Os resultados reais e previstos são visualizados graficamente para uma avaliação qualitativa do desempenho do modelo.

Esta abordagem híbrida, combinando características aprendidas por meio de convoluções com mecanismos recorrentes, é particularmente útil para capturar padrões temporais complexos em séries temporais.

Parte II

Detecção e Substituição de Anomalias

6 Redes Neurais Recorrentes

6.1 Abordagem Monolítica

O código–fonte `Anomaly_Detection_RNN.py` emprega uma abordagem de aprendizado profundo utilizando uma Redes Neurais Recorrente (RNN) para detectar e substituir anomalias em uma série temporal. Para a construção e treinamento da RNN, é utilizada a função `create_and_train_RNN` que inicialmente normaliza os dados com a classe `StandardScaler` do pacote `sklearn`. Os dados normalizados são então formatados em sequências de entrada compostas por cinco elementos consecutivos da série temporal. O modelo de RNN é montado com três camadas de memória de curto e longo prazo (LSTM) através da biblioteca Keras, intercaladas por camadas de regularização ou "`Dropout`". Em seguida, a detecção de anomalias é realizada pela função `anomaly_detection`, que procede na identificação de elementos nos dados originais que divergem significativamente das previsões do modelo treinado. Se identificada uma diferença maior do que um determinado limiar, o ponto é marcado como uma anomalia. Na parte principal do código, inicialmente é feita uma preparação dos dados do dataframe "`training_data`", seguida pelo treinamento do modelo de RNN. Posteriormente, é realizada a detecção de anomalias sobre esses mesmos dados. Uma vez identificadas, as anomalias são removidas do dataframe original, originando um novo conjunto de dados sem as anomalias. Por fim, os valores originais, valores previstos e as anomalias identificadas são visualizadas por meio de um gráfico. O código, portanto, emprega técnicas modernas de aprendizado profundo para detecção e tratamento de anomalias em séries temporais.

6.2 Abordagem Híbrida com Duas Camadas RNN/LSTM

O código–fonte `Anomaly_Detection_Hybrid_RNN_LSTM.py` apresenta uma implementação de uma técnica de aprendizagem profunda para modelar uma série temporal e detectar e substituir anomalias na mesma. A função `create_train_model(data)` utiliza uma rede neural recorrente LSTM (Long Short-Term Memory) para modelar a série e retorna o modelo treinado. Já a função `detect_replace_anomalies(model, series, window_size, sigma=1.0)` detecta as anomalias nos dados, considerando anomalias pontos que se encontram além de um número σ de desvios padrão da média em uma janela deslizante. As anomalias detectadas são substituídas pelos valores previstos pelo modelo treinado. A função `plot_data(orig_data, cleaned_data, anomalies)` cria um gráfico para visualizar os dados originais, os dados limpos e as anomalias substituídas. No código principal, a série de dados é extraída do dataframe "`training_data`", o modelo é treinado e, em seguida, é usado para detectar e substituir as anomalias. Finalmente, os dados originais e as anomalias são plotados e os dados limpos são salvos em um arquivo CSV chamado "`cleaned_data.csv`". Portanto, este código apresenta uma abordagem eficaz para a detecção e substituição de anomalias em dados de séries temporais utilizando redes neurais LSTM.

6.3 Arbodagem Híbrida com Suavização Exponencial

No âmbito da análise de séries temporais, o código–fonte `Anomaly_Detection_Exponencial_Smoothing_RNN.py` descreve um método híbrido que integra suavização exponencial e redes neurais recorrentes do tipo LSTM para detecção e tratamento de anomalias. Inicialmente, a função `suavizacao_exponencial` é utilizada para aplicar o método de suavização exponencial `SimpleExpSmoothing` aos dados da série temporal, visando suavizar flutuações e destacar tendências mais persistentes. Opta-se por fixar o parâmetro de suavização (alpha) em $\alpha \equiv 0.2$, garantindo, desta forma, que os valores mais recentes têm um peso adequado no cálculo da média móvel ponderada exponencialmente.

A função `preparar_dados` transforma o conjunto de dados da série temporal para o formato `float32` e depois os redimensiona a uma estrutura bidimensional, para submetê-los a normalização via `MinMaxScaler`. Este escalonamento normaliza o intervalo dos valores, garantindo que conflitos de escala não comprometam o desempenho do modelo neural. A subsequente função `separar_dados` divide o dataset normalizado em subconjuntos de treino e teste, provendo a base para uma validação rigorosa do modelo.

A função `criar_dataset` manipula os conjuntos de treino e teste, estruturando os dados em uma sequência em que a entrada para a LSTM consiste em `look_back` períodos temporais e a saída corresponde ao período seguinte. Este rearranjo é imperativo para treinar a rede a reconhecer padrões sequenciais e realizar previsões futuras. O procedimento de remodelação subsequente transforma os dados em um formato de três dimensões expositoras, compatível com as expectativas da rede neural recorrente.

O modelo LSTM, que é gerado pela função `criar_modelo`, consiste em uma camada LSTM

com quatro unidades seguida por uma camada densa de projeção única, todo o sistema sendo compilado com a função de perda de erro quadrado médio e o otimizador `adam`. Após a construção, o modelo é treinado com base no conjunto de treino, usando o número de épocas e o tamanho do lote definidos como parâmetros. As previsões realizadas pelo modelo nos subsets de treino e teste são transformadas inversamente para a escala original, permitindo uma comparação realística com os valores efetivos da série.

A remoção de anomalias é realizada pela função `remover_anomalias`, que estabelece um limiar estatístico baseado no desvio padrão das diferenças entre os valores previstos e os reais. Os dados que excedem tal limiar são classificados como anomalias e suprimidos do conjunto de dados. Finalmente, a função `plotar_grafico` exibe uma representação gráfica abarcando os dados originais, as previsões do modelo e o conjunto resultante após a depuração de anomalias, fornecendo um mecanismo visual de confirmação da eficácia do processo.

6.4 Abordagem Híbrida com Média Móvel

O código `Anomaly_Detection_Moving_Average_RNN.py` aqui apresentado propõe um mecanismo de detecção e tratamento de discrepâncias em dados sequenciais temporais, aplicando o conceito de médias móveis em conjunto com uma rede neural recorrente, especificamente a Long Short-Term Memory (LSTM). Inicialmente, as dependências e bibliotecas necessárias são importadas, com destaque para as destinadas ao processamento de dados e construção de redes neurais. O tratamento dos dados inicia-se pela sua normalização, de maneira a adequar a amplitude das variáveis de entrada para um intervalo padrão que otimiza a performance do modelo de aprendizagem profunda.

Posteriormente, a função 'prepare_data' é encarregada de organizar a série temporal em porções sequenciais que servirão de entrada para o treinamento da LSTM, cada segmento é acompanhado pelo valor subsequente da série que servirá como alvo a ser predito pelo modelo. A LSTM, uma estrutura projetada para captar dependências de longo prazo em dados sequenciais, é treinada com as sequências modeladas, induzindo-a a reconhecer padrões subjacentes na série temporal. O treino é realizado através da função 'train_lstm', que configura e executa o processo de otimização dos parâmetros internos da rede.

Com a LSTM devidamente ajustada, avança-se para a etapa de detecção de anomalias com a função 'detect_anomalies', que, por meio do cálculo de erros entre previsões e valores reais, identifica pontos discrepantes com base em um limiar estatístico. Esses pontos são classificados como anômalos caso seu erro supere a média acrescida de um múltiplo do desvio padrão. Dessa forma, é possível isolar as observações que desviam do comportamento padrão aprendido pela LSTM.

Por fim, exclui-se do conjunto original os dados identificados como anormais, resultando em uma série temporal limpa, desprovida de flutuações atípicas que poderiam mascarar ou distorcer análises futuras. As séries, original e depurada, são então representadas graficamente através da função 'plot_data', que ilustra ambas numa mesma figura para fácil comparação visual. Esse

processo de limpeza de dados é fundamental para garantir a integridade e confiabilidade de modelos preditivos subsequentes ou para a correta interpretação da série temporal em análises exploratórias ou inferenciais.

6.5 Abordagem Híbrida com ARIMA

O programa contido no arquivo `Anomaly_Detection_ARIMA_RNN.py` integra técnicas de aprendizado profundo para processar e depurar séries temporais, com o objetivo de identificar e tratar anomalias. Utilizando o Python como linguagem de programação, o código começa pela importação de pacotes indispensáveis, que fornecem ferramentas para manipulação de dados, modelagem de autoencoders, normalização de dados, construção de redes neurais recorrentes (RNN), e visualização gráfica.

A primeira fase do processo consiste em normalizar os dados através da função `standardize_data`, que aplica uma transformação `MinMaxScaler` para reescalar os dados, garantindo que os valores de entrada da rede neural variem $[0, 1]$. Essa padronização é fundamental para uma convergência mais eficiente durante o treinamento do modelo de autoencoder, que é realizado na sequência pela função `model_autoencoder`. Neste estágio, a rede neural dimensiona e modela a informação para identificar padrões considerados normais no conjunto de dados.

A função `mask_anomalies` é a responsável por aplicar o modelo de autoencoder treinado, identificando desvios que superam um limiar pré-estabelecido. Estes são marcados como anomalias e substituídos por valores nulos, isolando-os da série temporal para análises futuras. A limpeza dos dados prepara o terreno para a aplicação de uma rede neural recorrente LSTM, elaborada pela função `lstm_model`. A LSTM é especializada na detecção de padrões em sequências temporais e é adequadamente estruturada para prever valores em séries temporais depuradas de inconsistências.

Após a execução das funções de limpeza, os dados são visualizados por meio de um gráfico, gerado pela função `plot_data`, que exibe a comparação entre os dados originais e os dados já processados e limpos. Essa representação gráfica permite avaliar a eficácia da limpeza de anomalias e confirma a melhoria na homogeneidade da série temporal.

Por fim, a função `clean_data` arquiteta a orquestração de todo o procedimento de limpeza, iniciando pelo escalonamento dos dados, seguindo pela detecção de anomalias, e concluindo pela visualização comparativa dos dados. Os dados limpos, juntamente com o modelo LSTM treinado, são devolvidos, fornecendo uma base confiável e aprimorada para futuras análises e predições sobre a série temporal em foco.

7 Redes Neurais de Convolução

7.1 Abordagem Monolítica

O código–fonte `Anomaly_Detection_CNN.py` implementa uma abordagem de aprendizado profundo empregando redes neurais convolucionais (CNNs). A normalização das variáveis de interesse é

realizada por meio do `StandardScaler`, que ajusta os dados a uma distribuição de média zero e variância unitária. Posteriormente, a modelagem convolucional demanda a reestruturação dos dados em formato bidimensional, compatível com as operações da rede neural.

A construção do modelo é efetuada utilizando o framework Keras, com a sequência de passos que começa pela adição de uma camada convolucional, habilitada a capturar dependências locais. A camada `Flatten` subsequente, lineariza as saídas para que sejam processadas por uma camada densamente conectada, que projetará a saída final do modelo. A função de ativação 'relu', selecionada para a camada convolucional, introduz não linearidades essenciais ao aprendizado, enquanto a camada 'Dense' proporciona a predição do valor subsequente na série temporal.

A compilação do modelo utiliza o otimizador "`adam`", um algoritmo eficiente para o ajuste dos pesos sinápticos, e a função de perda "`mse`", que orienta a otimização no sentido da redução do erro quadrático médio. O treinamento da rede é realizado em bateladas unitárias ao longo de dez épocas, enquanto métricas de desempenho são registradas para análises futuras através de `TensorBoard`. Com o modelo treinado, procede-se ao processo de detecção de anomalias, definindo como critério de detecção valores residuais que se desviem significativamente, especificamente mais de três desvios padrões do esperado.

As anomalias identificadas são então corrigidas, substituindo-as pela mediana do conjunto de dados normalizados. Uma transformação inversa é aplicada a estes valores corrigidos para realinhá-los à escala original, utilizando-se a inversão do escalonamento feito pelo '`StandardScaler`'. O resultado final é um conjunto de dados limpo, livre de anomalias, que representa mais fidedignamente a série temporal analisada.

Visualizações são geradas para ilustrar tanto a série temporal original quanto a tratada, assim como para descrever a evolução da perda durante o treinamento e a tendência geral do erro quadrático médio ao longo das épocas. Portanto, o uso de CNNs para a detecção e correção de anomalias em séries temporais estabelece um método poderoso, capaz de melhorar a qualidade e a confiabilidade dos dados para análises subsequentes.

7.2 Abordagem Híbrida com Suavização Exponencial

O código–fonte apresentado no arquivo `Anomaly_Detection_Exponential_Smoothing_CNN.py` conjuga técnicas de suavização exponencial com uma rede neural convolucional para a detecção e tratamento de anomalias em séries temporais. Primeiro, os dados são obtidos e carregados de um arquivo CSV através da função `read_csv` da biblioteca `pandas`. Em seguida, são fornecidas informações estruturais sobre os dados carregados e o conteúdo dos mesmos é exibido, proporcionando uma visão geral do conjunto de dados disponível para análise.

A etapa seguinte consiste em executar uma suavização exponencial sobre os dados, técnica na qual as observações recentes recebem maior ponderação. Tal suavização é calculada pela função `exponential_smoothing`, que utiliza o parâmetro alpha para determinar o nível de suavização

aplicado à série. O produto dessa suavização é uma série temporal com diminuição de ruídos e possivelmente mais representativa da tendência subjacente dos dados.

Após a suavização, procede-se à normalização desta série alisada, empregando o `MinMaxScaler` do `sklearn.preprocessing`, o qual reajusta os valores de maneira que fiquem dentro de um intervalo pré-definido. Os dados normalizados são então estruturados em uma forma compatível para a aplicação de uma CNN, sendo configurada para processar sequências dos últimos dez pontos temporais.

O modelo da CNN é definido através da função `cnn_model`, que estabelece uma arquitetura formada por uma camada convolucional, uma camada de pooling para redução dimensional, seguida de uma camada de achatamento e duas camadas densas para a realização da tarefa de regressão. Após o treinamento do modelo na série temporal, os valores preditos são comparados com os valores reais, identificando-se discrepâncias significativas que caracterizem as anomalias.

Estas anomalias são então eliminadas do conjunto de dados, e as informações de suavização e normalização são removidas, retornando assim uma série temporal depurada. O código culmina com a geração de um gráfico para a visualização comparativa dos dados originais e dos resultados após a limpeza das anomalias. Com isso, o código propicia uma ferramenta para assegurar a integridade e a precisão de análises subsequentes nas séries temporais.

7.3 Abordagem Híbrida com ARIMA

O código–fonte `Anomaly_Detection_Hybrid_ARIMA_CNN.py` exemplifica uma metodologia inovadora para a detecção e manipulação de anomalias em séries temporais combinando modelos ARIMA com redes neurais convolucionais (CNN). Inicialmente, o código importa as bibliotecas necessárias para a manipulação de dados, processamento estatístico, modelagem de aprendizado de máquina e visualização gráfica. A estacionariedade da série é verificada através do teste Dickey-Fuller aumentado, sendo um pré-requisito para a aplicação eficaz do modelo ARIMA.

Após assegurar que a série é estacionária, o modelo ARIMA é ajustado automaticamente à série utilizando a função auto_arima, que seleciona os parâmetros ótimos. Utilizando as previsões geradas pelo modelo ARIMA, calculam-se os resíduos, que são então analisados em busca de desvios que caracterizam as anomalias, através de um limiar estabelecido pelo escore Z. Normaliza-se a série temporal para treinar a rede neural convolucional, permitindo que a rede processe os dados de forma mais eficiente.

O modelo de CNN é composto por uma camada convolucional, uma camada de achatamento e uma camada densa, e é compilado com um otimizador e uma função de perda específicos para tarefas de regressão. Após o treinamento da CNN, fazem-se previsões sobre os dados normalizados. Novamente, as anomalias são detectadas comparando as previsões da CNN com os dados originais, levando em conta um múltiplo do desvio padrão dos resíduos.

O programa realiza uma combinação das anomalias detectadas tanto pelo modelo ARIMA quanto pela CNN, criando um conjunto unificado de pontos anômalos. Subsequentemente, os dados são

limpos pela remoção das referidas anomalias. Finalmente, o código provê a visualização gráfica da série original e da série já tratada, permitindo a comparação entre ambas e a avaliação dos resultados do processo de limpeza.