Implementação e Avaliação de Modelos de Regressão Linear e Florestas Aleatórias Aplicados à Previsão de Séries Temporais

Igor Mol

igor.mol@makes.ai

6th November 2023

O arquivo `sliding_window_v2.py` apresenta o código que define e treina um modelo de regressão linear utilizando a abordagem de janela deslizante para prever séries temporais. Inicialmente, as variáveis de tempo e quantidade são extraídas dos dados de treinamento e transformadas em vetores unidimensionais. Uma função que implementa a abordagem de janela deslizante é definida, treinando o modelo de regressão em cada janela de dados e efetuando previsões para o horizonte subsequente. Esse processo é repetido continuamente, deslocando a janela de treinamento ao longo do conjunto de dados. Em conclusão, o código exibe dois gráficos: um que compara as previsões do modelo com a série temporal original e outro que demonstra a tendência do erro quadrático médio em diferentes janelas.

O script `random_forests.py` emprega a técnica de Floresta Aleatória para prever séries temporais. Primeiramente, os dados são preparados para a previsão por meio da criação de grupos contíguos, denominados 'janelas', e gera vetores de características e alvos. Em seguida, realiza-se uma validação cruzada para dividir a série em conjuntos de treinamento e teste, respeitando a ordem temporal dos dados. A Floresta Aleatória é então iniciada, treinando o modelo e fazendo previsões. Os valores previstos e os dados originais, juntamente com o erro quadrático médio de cada previsão, são armazenados. Um gráfico dos resultados é traçado ao final.

O arquivo `out_of_sample_unit_of_time.py` implementa a função `optimal_time_unit` com o objetivo de encontrar a melhor unidade de tempo para o modelo de série temporal. A função começa com `errors`, uma lista que armazenará os erros quadrados médios, verifica se a unidade de tempo é menor que o total de pontos na série e aplica uma janela deslizante para calcular a média dos pontos de dados. Os conjuntos X e Y são divididos em rotinas de treinamento e testes, um modelo de regressão linear é ajustado, utilizado para fazer previsões e calcular o erro quadrado médio. O erro para cada unidade de tempo é armazenado e, no final, a função retorna a unidade de tempo com o menor erro. Um gráfico é gerado para a observação do erro versus o tamanho da janela, e a

função é invocada considerando unidades de tempo t entre $1 < t < 60$.

O script `out_of_sample_optimal_horizon.py` é utilizado para analisar uma série temporal, encontrar o horizonte de previsão ideal e visualizar os resultados. O primeiro bloco de código define duas funções, `linear_regression_model`, que treina um modelo de regressão linear a partir de um dataframe, e `find_optimal_horizon`, que itera sobre diferentes horizontes de previsão para encontrar aquele que produz o menor erro quadrático médio. Uma terceira função, `plot_model_performance`, é então usada para traçar o erro de previsão versus o horizonte de previsão. A última parte do código aplica essas funções aos dados de treinamento, imprime o horizonte ideal e exibe os resultados.

Já o código–fonte `optimal_unit_of_time.py` realiza uma análise de série temporal com o objetivo de encontrar a unidade de tempo ótima para agrupar os dados e treinar um modelo de regressão linear. Os dados são primeiramente ordenados cronologicamente. Uma função é então definida que agrupa os dados em unidades de tempo especificadas, treina um modelo de regressão linear, faz previsões e calcula o erro quadrático médio (MSE). Um intervalo de unidades de tempo é definido para testar o desempenho do modelo e os resultados são plotados em um gráfico. A função retorna a unidade de tempo que resulta no menor MSE.

O código–fonte `ARIMA.py` apresenta uma implementação do modelo ARIMA para modelar uma série temporal, onde as funções para executar o modelo, plotar o gráfico e fazer previsões estão definidas. O modelo ARIMA é executado nos dados de treinamento e depois usado para prever os próximos 50 passos da série temporal. O resultado então é representado graficamente, mostrando tanto os dados originais como o modelo ajustado e as previsões para futuros passos.

No arquivo `ARIMA_sliding_window.py`, reformulamos o método ARIMA implementado no código–fonte descrito no parágrafo acima segundo um modelo de janelas–deslizantes. O código inicializa duas listas: `history`, que armazena os primeiros elementos da série temporal como determinado pelo tamanho da janela, e `predictions`, que armazena as previsões futuras. Em seguida, iteramos sobre o restante da série temporal, construindo e ajustando um modelo ARIMA por meio da lista `history`, e fazemos uma previsão n–passos adiante, onde $n := $ `horizonte`. Por fim, adicionamos os dados reais após todas as previsões terem sido feitas, calculamos o erro quadrático médio entre os dados reais e as previsões, imprimimos o resultado, e então confrontamos os dados reais e as previsões através de um gráfico.