MULTIVARIATE TIME SERIES FORECASTING PROJECT
Shahab Yousef-Nasiri

PROMPT:

Assignment: Analysing Statistical Relationships and Feature Engineering

- You have been provided with a dataset of anonymised features on a weekly frequency.
- In the following, you shall investigate the statistical significance of these features on two target variables given in the last two columns, named “Target 1” and “Target 2”.
- Identify which features show reliable statistical significance and which show spurious relationships to each of the targets.
- Pay attention to proper feature engineering.
- You may transform features in any way you see fit.
- Report the most significant features.
- First, focus on individual features, afterwards build a predictive model using your selection of relevant features.
- This task shall be approached from the lens of prediction, i.e., investigate the performance of features observed at time t to predict the target at time t + 1.
- Your solution will be tested on a hidden test set to evaluate the out-of-sample performance of your solution.
- For each step explain your motivation why you chose a feature engineering, data cleaning, or model selection step.
- Please use Python for your analysis.
- Please provide a concise report as well as all used code files.

SOLUTION:

The program is divided into three sections, the first of which houses the key functions responsible for performing data cleaning, statistical tests, filtering, outlier detection and imputing. Section 2 involves the feature engineering methods, ranging from time/seasonality features, cyclical features, statistical and decomposition features and Fourier features. Section 3 houses the XGBoost predictor model, alongside hyper parameterization via Bayesian optimization and feature selection.

All functionality is controlled through the ‘main’ script.

SECTION 1: Cleaning, Outliers, Filters

replace_inf_with_nan: Replaces inf values with nan, which were causing issues. Allows proper functionality of many other methods.

is_stationary: Applies the Augmented Dickey-Fuller (ADF) test to a given time series to statistically determine whether the series is stationary. It tests whether statistical properties of the series such as mean, variance, and autocorrelation is constant over time. This is crucial because most time series modelling techniques (such as lag creation or STL decompositions performed in the feature engineering section) assume or require stationarity.

dropping_columns_rows: The idea behind these data cleaning processes is to ensure that we can maintain a significant level of data integrity and to avoid training models on features with insufficient data. I have chosen to drop the first 310 rows in order to have a ‘full’ data set to analyse and process. The alternative would be to use imputation techniques to backwards fill however none are accurate enough to extend many of these column’s hundreds of times steps back and would introduce too much unnecessary noise into the data, counterproductive for training a model. All in all, features 67-113 are deleted due to either complete/or near complete lack of data.

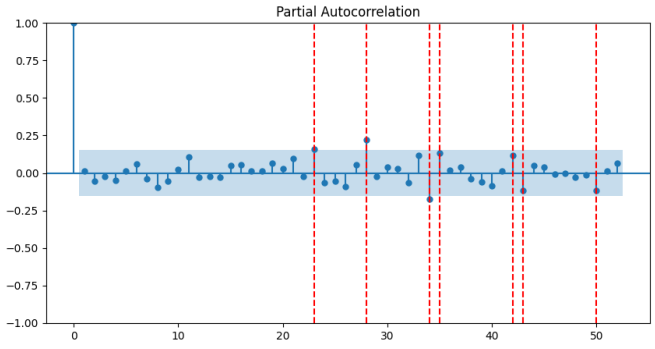
outlier_detection_MAD: This function identifies and handles outliers in the columns by replacing them with NaN values, using the Median Absolute Deviation (MAD) method for z-score calculation. A modified z-score is used here instead of the standard z-score because it is more robust to outliers in the data, making it suitable for skewed distributions often found in real-world data. The choice of MAD for outlier detection is particularly effective because it is less influenced by extreme values than the standard deviation, providing a more reliable measure of variability in datasets with skewed or non-normal distributions.

constant_duplicate_filter: Identifies and removes constant features—columns where the standard deviation is zero. These features do not provide any informative variance for analysis or modelling and can be safely dropped leading to less overfitting. Also removes duplicate features using transposition techniques, eliminating redundant data.

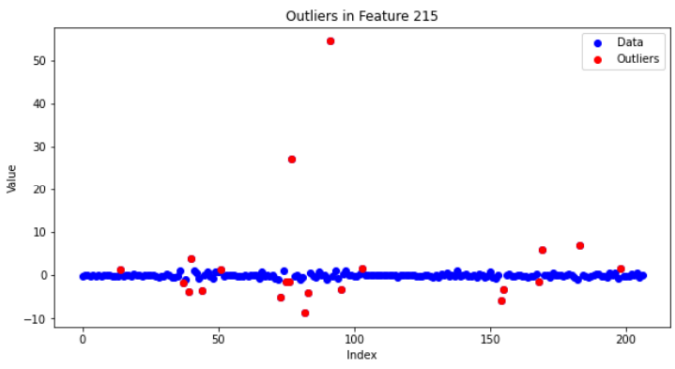
spearman_correlation_filter: Systematically evaluates the relationship between each feature column and a specified target variable using Spearman's rank correlation coefficient. This non-parametric measure assesses how well the relationship between two variables can be described using a monotonic function, not assuming a linear relationship and not sensitive to outliers. Significantly correlated features are removed, as to avoid causing multicollinearity issues in predictive models, potentially skewing the model's performance and interpretability. This filtering is crucial in feature selection to reduce the dimensionality of the dataset, enhance model generalizability and reduce significant overfitting. E.g. Feature 251 is an almost complete copy of target 1 and is removed by this filter as to stop 'answer leakage' and overfitting in the training set.
Features removed with correlation > 0.7: Feature 251 0.993, Feature 266 0.712, Feature 267 0.703

cross_correlation_lag_filter: The function examines the cross-correlation between a feature time series and a target time series across different lag values, using the cross-correlation function (ccf) to compute the correlation values for each lag and identifies which of these lagged correlations are statistically significant. Lags with absolute cross-correlation values exceeding this threshold are considered significant, suggesting that the feature series may have a predictive relationship with the target series at those lags. These lags are then incorporated as extra features in the feature engineering section.

partial_auto_correlation_function_filter: Function investigates a time series to identify significant lags where the partial autocorrelation is distinct from zero, beyond random chance, up to a user-defined maximum number of lags, and within a certain level of statistical confidence. Partial autocorrelation measures the correlation of a time series with its own lagged values, discounting the contributions from intermediate lags, therefore isolating the direct effect of past values at each lag. This is important as it helps differentiate between direct and indirect correlations within the time series data. The idea is used to determine significant target lags that suggest a potentially predictive relationship that can be utilized in model building and feature engineering.



Partial Autocorrelation Function, with vertical lines indicating significant lags.



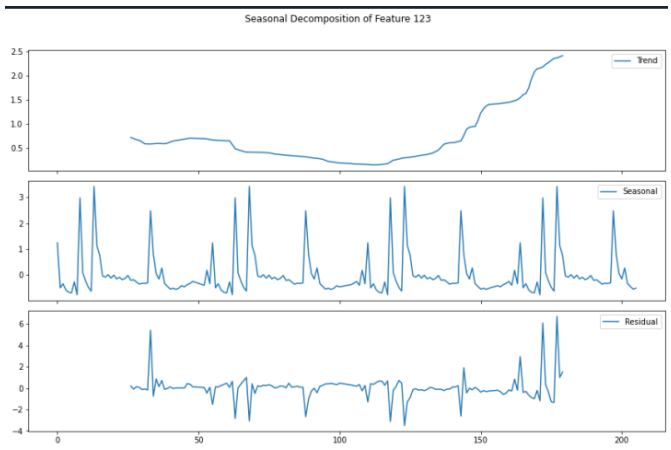
MAD Outlier detection process.

SECTION 2: Feature Engineering

add_time_features: Designed to fill the dataset with additional time-related features from a generated date range, improving the time series model's ability to capture temporal dynamics. Adding such time features is vital in time series analysis because many datasets exhibit seasonality, trends, and cycles that are dependent on time-related components. By including these features, predictive models can more accurately account for periodic changes and also recurring patterns in the data. This process of feature engineering is essential for models that do not inherently account for time such as the XGBoost model that we are trying to take advantage of.

add_cyclical_features: Transforms temporal features into a format that can be properly utilized by machine learning models, particularly ones that do not naturally handle cyclicity. This is an essential step because many algorithms do not recognize the end of a cycle and the start of a new one (e.g., December to January or week 52 to week 1).

auto_decompositions_feature_creation_wiz: This function is aimed at automatically decomposing time series features that are both non-stationary (i.e features that exhibit trends and seasonality) and significantly correlated with the target using the Spearman correlation test. The spearman is appropriate in this case as it does not assume that the data is normally distributed, which can be particularly advantageous if the underlying assumptions of parametric tests (like Pearson's correlation) are violated. Additionally, is also more robust to outliers as it compares the rank orders rather than actual values when it calculates correlation.



Seasonal, Trend and Residual decomposition of non-stationary correlated features.

The function calculates the Spearman rank correlation coefficients between all features and the target. Then, the function checks two conditions before decomposing: if the feature is non-stationary (which could affect model performance if left unaddressed) and if its absolute correlation with the target exceeds a certain threshold. The decomposition process involves breaking down the time series into components such as trend, seasonality, and residual noise. This process is critical in time series analysis as it allows for the isolation and direct modelling of different temporal dynamics, which can significantly improve the predictive power and interpretability of predictive models.

auto_lag_feature_creation_wiz: For each stationary feature the function assesses the cross-correlation between the feature and the target variable up to a specified max_lag. It uses the cross_correlation_lag_filter to find significant lags where the cross-correlation exceeds the statistical threshold set by alpha. These significant lags are then used to create new lagged features for the corresponding original feature. Additionally, the function applies the partial_auto_correlation_function_filter to the target variable itself to identify significant lags that are internally correlated with the target. Lagged features are then created based on these significant lags for the target column.

add_rolling_statistics_features: This function calculates rolling statistics for a specified column across a range of window sizes. It iterates over each window size and statistic type provided in the window_sizes and statistics parameters, respectively. The supported statistics are mean, variance, standard deviation, minimum, and maximum. Rolling statistical features are important in time series analysis because they capture the changing dynamics of the series over time, providing a "moving" insight into the data's behaviour. These features can reveal trends, volatility, and extremes that are not apparent from the raw data alone. Including such features in our dataset can greatly enhance the predictive models, enabling them to account for recent trends and fluctuations and improve forecast accuracy. For instance, rolling means can smooth out short-term fluctuations and highlight longer-term trends, while rolling standard deviation can measure the variability in a given window, which might be indicative of volatility.

add_fourier_features: For each specified period and harmonic, the function generates sine and/or cosine terms based on the DataFrame's index, which represents time or sequence order. Fourier features are particularly valuable in time series modelling because they allow for the capture of more complex, repetitive patterns, especially those that are seasonal in nature. By breaking down these patterns into a series of sine and cosine waves with different frequencies, the model can be made to understand and predict periodic behaviour effectively. Note: very small periods can lead to aliasing issues.

SECTION 3: XGBoost Predictor Model

I've designed this predictive modelling section to be a comprehensive machine learning workflow for regression analysis using XGBoost, which is a powerful gradient boosting algorithm. I've chosen XGboost for a variety of reasons. Firstly, it is robust to overfitting due to built-in regularization. It is also natively able to manages sparse data and missing values, making it versatile, especially for datasets with outlier removal and most importantly, it is capable of capturing complex relationships in data through decision trees making it highly adept at modelling datasets where numerical features may interact in nonlinear ways. These capabilities make it an ideal choice for our predictive modelling needs, promising both accuracy and efficiency.

The process begins with feature selection, where an initial XGBoost model is trained to determine the importance of each feature relative to the target variable. Features that do not contribute significantly to the model's predictive power, measured by an importance multiplier applied to the mean feature importance, are discarded. This step is

crucial for reducing model complexity and computational cost, as well as for potentially improving model performance by eliminating irrelevant or noisy features.

The core of the workflow is the Bayesian optimization, which seeks to identify the most effective hyperparameters for the XGBoost model. This is done by defining a search space of possible hyperparameter values and using the hyperopt library to intelligently explore this space. The objective function guides this exploration by evaluating the XGBoost model's performance, with a given set of hyperparameters, using time-series cross-validation provided by TimeSeriesSplit. This method of validation is particularly important for time series data to preserve the temporal order of observations, which is crucial for avoiding look-ahead bias and ensuring that the model's ability to generalize is accurately assessed.

The perform_bayesian_optimization function manages this optimization process, repeatedly calling the objective function and updating its search based on the results. After a specified number of evaluations, it converges on the best hyperparameters, which are then used to train the final model. The trained model is then evaluated on both the training and test datasets. The mean squared error (MSE) and the coefficient of determination (R^2) are reported for each. Overall, the user has control over the hyperparameter space, max number of evaluations, the threshold multiplier for feature importance and the number of splits/folds in the cross-validation TimeSeriesSplit.

Evaluation Results:

TARGET 1

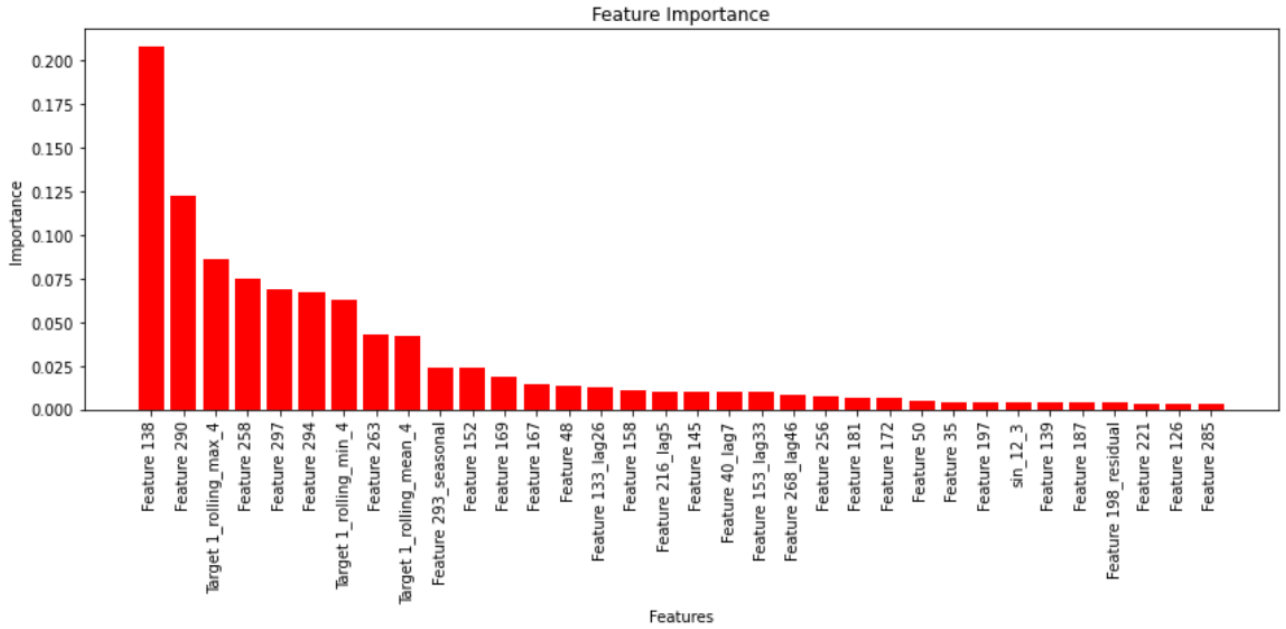
```
100%|| 25/25 [00:13<00:00,  1.90trial/s, best loss: 0.0024238871596489681]
Train MSE: 0.0001228242783025555
Train R^2: 0.9750056832399079
Test MSE: 0.0004406803019372794
Test R^2: 0.8287867596000512
Test Target Variance: 0.00257386812438024
```

TARGET 2

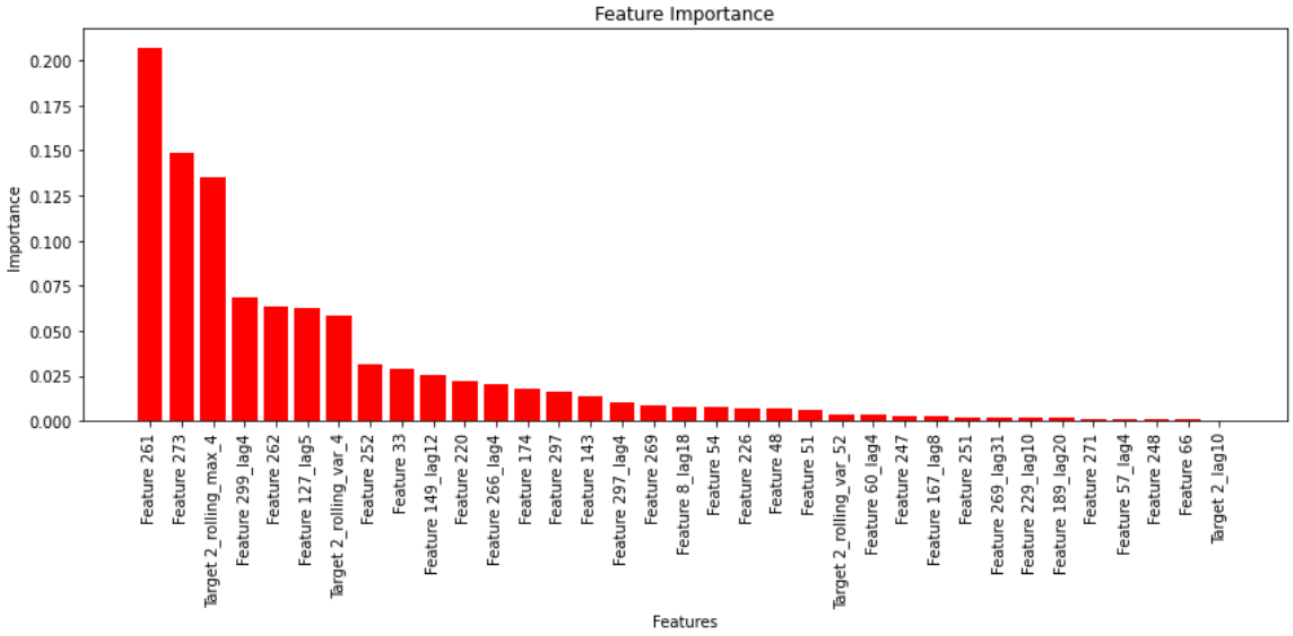
```
100%|| 10/10 [00:07<00:00,  1.38trial/s, best loss: 0.019754001005170434]
Train MSE: 4.586711211661224e-05
Train R^2: 0.9987794980086691
Test MSE: 0.00647551917255127
Test R^2: 0.5007396493414874
Test Target Variance: 0.01297022518153919
```

Statistically Significant Features:

TARGET 1:



TARGET 2:



Bonus:

In the case of predicting on a dataset with missing feature columns, I've created a data leakage function which should processes the test dataset (X_test) to prevent inadvertent inclusion of future information. It does so by applying several rules to nullify values in specific types of columns: i.e. Feature columns are replaced with nan, lags are carried forward into the test set accordingly, etc.

I've also created a univariate time series imputing/forecasting method called feature_imputing_fb to forecast and replace the nan values for the feature columns in the test set, as to allow for more accurate predictions by the XGboost algorithm. This imputation technique incorporates Facebook's Prophet model which allows for the capture and utilization of underlying patterns and trends in the data, leading to potentially more accurate and contextually appropriate imputations than simpler methods such as mean or median imputation. This can be especially important in predictive modelling where the integrity of time series data is crucial for making valid predictions. You are able to play around with the seasonality Fourier order terms for both the monthly and weekly seasonality's in the model.