Humayra Azra
Capstone 3 - Webtraffic

Exploratory Data Analysis (EDA) for Web Traffic Forecasting

The primary objective of this Exploratory Data Analysis (EDA) is to explore and understand the structure, behavior, and trends within web traffic data, which is crucial for accurate time-series forecasting. The EDA process, conducted as part of the capstone project *Web Traffic Forecasting*, plays an essential role in uncovering patterns, trends, seasonality, and identifying potential data challenges that may affect forecasting performance.

The first step in this analysis involved loading the dataset using the pandas library. A preliminary inspection was conducted using .head() to view the first few rows and .info() to gather information about the dataset's structure, column names, and data types. This allowed for a quick understanding of the data and its attributes. The dataset was found to include a date field, which was crucial for time-series analysis. Since time-series forecasting models depend on chronological data, this date column was transformed into a proper datetime format, allowing us to set it as the index of the DataFrame. This step enabled us to perform chronological operations, including resampling, rolling averages, and decomposition techniques.

To examine long-term trends and reduce daily fluctuations, the dataset was resampled to a weekly frequency. Weekly aggregation smooths out the short-term noise, revealing more stable patterns over time. This transformation was achieved using .resample('W').sum(), which sums the traffic counts for each week. The line plot of the resampled data highlighted smoother transitions and made it easier to observe the periodic behavior of web traffic. The resampling process suggested that web traffic is subject to seasonal fluctuations, with peaks and valleys corresponding to certain times of the year.

Next, to further analyze underlying trends, a 12-week rolling average was computed and plotted alongside the original weekly data. This smoothing technique helps to reduce the impact of short-term fluctuations and highlight long-term trends, both upward and downward. The 12-week rolling average provided a clearer visualization of consistent cycles, revealing that web traffic exhibited both a trend and a seasonal pattern.

In order to formally decompose the time series and separate its components, a seasonal decomposition was performed using the seasonal_decompose() function. This technique decomposes the time series into three components: trend, seasonal, and residual (or noise). The decomposition confirmed that the dataset exhibited a clear seasonal pattern, with traffic surges and dips occurring at regular intervals. The trend component revealed that traffic was gradually increasing over time. The residuals showed variability that was not explained by trend or seasonality, suggesting that the data contained spikes or random noise, which could be attributed to unexpected events or irregular user behavior.

Autocorrelation and partial autocorrelation plots were also generated using plot_acf() and plot_pacf(). These plots assess the relationship between current values and previous observations at different lags. The autocorrelation function (ACF) revealed strong correlations at weekly lags, further confirming the seasonal nature of the data. The partial autocorrelation function (PACF) helped to identify the optimal lag terms for time-series models, such as ARIMA, that can capture these seasonal patterns.

Since many statistical forecasting models, particularly ARIMA, require stationary data (meaning the statistical properties like mean and variance remain constant over time), it was essential to check for stationarity in the dataset. The presence of a clear trend and seasonality in the raw data suggested non-stationarity. To address this, first-order differencing was applied to the time series using .diff(), which calculates the difference between consecutive observations. This transformation was effective in reducing the trend, resulting in a time series that appeared more stationary.

A second seasonal decomposition was performed on the differenced series to ensure that the trend and seasonality had been adequately removed. The decomposition confirmed that the differenced data had become more stationary, with the residuals now resembling white noise, a key indicator that the differencing step had been successful.

In conclusion, the EDA phase provided valuable insights into the web traffic dataset. It confirmed the presence of strong trend and seasonality, highlighting the need for advanced forecasting methods. The dataset exhibited autocorrelation at predictable intervals, indicating that web traffic patterns are repeatable over time. The differencing process successfully made the data stationary, preparing it for time-series modeling.

These findings lay the groundwork for selecting appropriate forecasting models, including SARIMA (which accounts for both trend and seasonality), Facebook Prophet (which handles seasonality and missing data), or LSTM neural networks (to capture complex, non-linear relationships). The transformed and cleaned dataset resulting from this EDA is now ready for the next step in the capstone project: building, evaluating, and refining predictive models for web traffic forecasting.

Capstone 3: Web Traffic Forecasting – Data Preprocessing and Training Dataset Development Report

The purpose of this phase in the capstone project is to preprocess the web traffic dataset, ensuring it is in the proper format for use with machine learning models. While the EDA phase focused on understanding the patterns, trends, and seasonality within the data, this phase focuses

on transforming the dataset into a machine-learning-friendly format, ensuring that the features and target variable are appropriately structured for model training.

The notebook begins by reloading the time-series dataset and resampling it to a weekly frequency. This was identified in the EDA phase as an effective strategy to reduce noise while preserving key trends and seasonality. The weekly resampling aggregates daily data into weekly totals, ensuring that the forecasting models will be trained on consistent data.

The next step involves the creation of lag features. Lag features are a critical part of time-series forecasting because they allow models to learn patterns based on previous time steps. For example, the weekly traffic data was shifted backward by one, two, three, and four weeks to generate lag features. These lagged values become independent variables, while the current value (the traffic count for the target week) serves as the dependent variable or target. Creating lag features transforms the problem from a time-series prediction task into a supervised learning task, where models can be trained to predict future traffic based on historical values.

After generating the lag features, the dataset is cleaned by removing rows with missing values. The lagging process introduces missing values for the earliest observations, as there is insufficient historical data to create lagged variables for those rows. As a result, these rows are dropped to ensure the dataset's integrity. This is a standard trade-off in time-series forecasting, where some initial data points are sacrificed to enable better forecasting on subsequent data.

Following the cleanup process, the dataset is split into features and target variables. The features consist of the lagged traffic counts (lag_1, lag_2, lag_3, lag_4), and the target variable is the original weekly traffic count. This setup ensures that the forecasting model can be trained to predict future traffic based on historical patterns.

The next step is splitting the dataset into training and testing subsets. To maintain the time-series structure, the data is split chronologically, with the earliest 70% used for training and the remaining 30% used for testing. This method mirrors real-world forecasting scenarios where models are trained on historical data and used to predict future values. This chronological split prevents data leakage and ensures that the model is evaluated on unseen data that mirrors future forecasting conditions.

Before training the models, the features are scaled using MinMaxScaler, which ensures that each feature has a common range (usually between 0 and 1). Scaling is essential for models, especially those sensitive to feature magnitudes, such as gradient-based models or neural networks. Scaling prevents any one feature with a larger numeric range from dominating the model’s learning process.

In conclusion, this phase successfully transforms the raw time-series dataset into a structured, machine-learning-ready dataset. The preprocessing steps, including lag feature creation, handling

missing values, dataset splitting, and feature scaling, prepare the data for training effective forecasting models. With the data now in a suitable format, the next step involves training and evaluating machine learning models, including classical regression models and deep learning models like LSTMs, which will be explored in the following phases.

Capstone 3: Web Traffic Forecasting – Classification with Random Forest and XGBoost

This section of the project focuses on classifying web traffic into high or low categories using two advanced machine learning models: Random Forest and XGBoost. The original dataset contains timestamped traffic counts, and the objective is to classify whether traffic at a given time is above or below the median level. To approach this as a binary classification task, several features were engineered from the timestamp, including the hour of the day, day of the week, and the month. These time-based features help capture common patterns in user web behavior, allowing the model to better predict traffic surges.

The target variable was created by labeling each record as either high traffic (1) or low traffic (0), based on whether the traffic count exceeded the median. This simplifies the classification task, allowing the models to focus on learning patterns that consistently predict high-traffic periods.

Instead of randomly splitting the data, the dataset was sorted chronologically to preserve the time-series structure. Seventy percent of the earliest records were used for training, and the remaining thirty percent were used for testing. This split ensures that the models are trained on historical data and tested on future-like data, closely mirroring real-world forecasting conditions.

The first model evaluated was the Random Forest classifier. Using GridSearchCV for hyperparameter tuning, several parameters were optimized, including tree depth, number of estimators, number of features considered at each split, and minimum samples required per leaf node. The best configuration included a tree depth of 7, 100 estimators, 10 features, and a minimum of 4 samples per leaf, achieving a training accuracy of 96.4% through three-fold cross-validation.

On the test set, the Random Forest model achieved 94% accuracy, with high precision (0.98), indicating a low rate of false positives. The recall was 0.846, meaning the model was able to correctly identify a significant portion of the high-traffic instances. The F1 score, which balances precision and recall, was 0.904, showing strong model performance.

To further validate the approach, the same classification task was performed using XGBoost. GridSearchCV was used again to optimize the learning rate, tree depth, and subsample ratio. The best model had a learning rate of 0.1, tree depth of 3, and a subsample ratio of 0.5, achieving a training accuracy of 96.5%.

On the test data, the XGBoost model also achieved 94% accuracy, with slightly higher precision (0.985), indicating fewer false positives. However, the recall was slightly lower at 0.829, meaning the model missed some true high-traffic instances. The F1 score for XGBoost was 0.897, which was slightly lower than that of Random Forest.

In conclusion, both models performed well, with Random Forest offering a more balanced performance in terms of precision and recall. XGBoost excelled in terms of precision but sacrificed some recall. The choice between the models depends on business objectives, such as whether minimizing false positives or capturing more high-traffic instances is more critical. Future improvements may include incorporating lagged features, event-based data (e.g., holidays), and experimenting with deep learning models like LSTM to capture more complex dependencies.

This project showcases the power of ensemble methods for classification tasks, even with basic features, and highlights the importance of model selection based on business needs.