Proactive Network Congestion Prediction & Bandwidth Management (Smart Network Optimization for Enhanced Connectivity)

This project implements a machine learning system to proactively predict network congestion on a set of routers and provide automated, actionable recommendations for bandwidth allocation. It leverages XGBoost models trained on time-series network data to forecast congestion probabilities and drive a rule-based recommendation engine.

working with the final deliverable of the repository

the dashboard can be asscessed by the link- https://genesishackathon-hackstreetboys.streamlit.app/#congestion-predictions

for sample csv's you can use:

- notebooks/data/model_ready_dataset_p.csv
- notebooks/data/model_ready_dataset_g.csv files

and also visualize them accordingly.

Approach (baseline)

- Supervised **classification** for *congestion in next interval*: features include traffic volume, latency, bandwidth used/allocated, utilization ratio, time-of-day/week, lags, number of users, application usage and many more.
- Simple **reallocation policy**: shift bandwidth from underutilized to overutilized devices/links based on predicted risk and a utilization threshold.

Project Goal

The primary objective is to move from a reactive to a **proactive** network management strategy. By anticipating congestion before it occurs, the system aims to:

- Improve network performance and reliability.
- Optimize resource allocation by adjusting bandwidth dynamically.
- Reduce manual intervention and operational overhead.

System Architecture & Data Pipelines

The project is built around two distinct data pipelines to ensure robust development, testing, and validation:

1. **Pipeline P (Processed Data)**
    - **Source:** Raw, historical log files from problem statement itself.

- **Process:** The `cleaning_given_data.ipynb` notebook ingests, cleans, merges, and features-engineers this data.
- **Output:** `model_ready_dataset_p.csv`. This dataset represents the merged data of all the different raw files that were given.

2. **Pipeline G (Generated Data)**

- **Source:** Synthetically generated from scratch.
- **Process:** The `dataset_generator.ipynb` notebook creates a clean, well-structured dataset with realistic, simulated network patterns (e.g., peak business hours, evening usage spikes).
- **Output:** `model_ready_dataset_g.csv`. This dataset provides a controlled environment for model testing and serves as a performance baseline.

Notebooks

This repository contains the following notebooks, which form the core of the project.

1. `cleaning_given_data.ipynb`

This notebook handles the entire ETL (Extract, Transform, Load) process for the real-world data pipeline.

- **Purpose**: To process and merge multiple raw data sources into a single, model-ready dataset.
- **Inputs**:
  - `Router_A_router_log_15_days.csv`
  - `Router_B_router_log_15_days.csv`
  - `Router_C_router_log_15_days.csv`
  - `application_usage.csv`
  - `user_activity.csv`
  - `external_factors.csv`
  - `configuration_history.csv`
- **Key Steps**:
  1. **Load & Consolidate**: Loads all raw CSVs and combines the individual router logs.
  2. **Aggregate**: Computes daily summaries from detailed logs (e.g., `total_peak_app_traffic`, `total_logins`, `Num_Config_Changes`).
  3. **Merge & Clean**: Merges all data frames on `Date` and/or `Device Name` and fills any resulting `NaN` values.
  4. **Export**: Saves the final, sorted dataset as `final_model_ready_dataset.csv`.

2. `dataset_generator.ipynb`

This notebook generates a high-quality, synthetic dataset for controlled model training and evaluation.

- **Purpose**: To synthetically generate a complete and realistic dataset that mimics real-world network behavior.
- **Key Steps**:
  1. **Simulate Router Logs**: Generates hourly log data, programmatically increasing traffic and latency to simulate **peak business hours** and **evening usage spikes**.

2. **Simulate Daily Summaries**: Creates synthetic daily data for application usage, user activity, and external events (e.g., outages, maintenance).
3. **Integrate & Model Events**: Merges all synthetic data and simulates the impact of events (e.g., doubling latency during a network outage).
4. **Create Target Variable**: Applies a rule-based function (`create_new_flag`) to label congestion events.
5. **Export**: Saves the final dataset as `new_realistic_dataset_v2.csv`.

Of course. Here is a more detailed breakdown of the machine learning and recommendation components from the `model+recommendation` notebooks, formatted for your README file.

---

3. The Model & Recommendation Engine

The `model+recommendation_p.ipynb` and `model+recommendation_g.ipynb` notebooks are the heart of this project. They execute a sophisticated workflow to move from historical data to future predictions and actionable advice. Here's a detailed look at each stage.

1. Feature Engineering: The Sliding Window

To predict the future, the model must understand the recent past. This is achieved using a **sliding window** approach, which transforms the time-series data into a format that a classification model can understand.

- **Concept**: To predict congestion for a specific hour (let's call it `T`), the model analyzes the network's behavior over the **previous 12 hours** (from `T-12` to `T-1`). This 12-hour block is the "window."
- **Vector Construction**: The function `create_training_samples` flattens this 12-hour window into a single, comprehensive feature vector. It takes 9 key metrics (like `Traffic Volume`, `Latency`, `Bandwidth Used`, `total_logins`, etc.) for each of the 3 routers at each of the 12 hours.
- **Result**: This process creates one training sample with a feature vector of size $12 \text{ hours} \times 3 \text{ routers} \times 9 \text{ metrics} = 324$ features. This vector provides a rich, time-aware snapshot of the network's state leading up to a potential congestion event.

2. Model Training: A Multi-Model Strategy

Instead of a single, general-purpose model, this project uses a more effective, specialized approach.

- **Algorithm**: The core algorithm is the **XGBoost Classifier**, a powerful gradient-boosting model known for its high accuracy and ability to capture complex, non-linear relationships in data.
- **Specialized Models**: Three independent XGBoost models are trained—one for each router (`Router_A`, `Router_B`, and `Router_C`). This **multi-model strategy** allows each model to become an expert on its assigned device, learning its unique traffic patterns and congestion triggers far more effectively than a single "one-size-fits-all" model could.

3. Prediction: From Data to Probability

The `predict_congestion_proba` function operationalizes the trained models, turning historical data into a forward-looking risk assessment.

- **Input**: A target timestamp for which a prediction is needed.

- **Process**: The function automatically constructs the 12-hour feature vector preceding the target time.
- **Output**: It returns a **congestion probability** for each router—a precise score between 0.0 (no risk) and 1.0 (very high risk). This probabilistic output is more valuable than a simple "yes/no" prediction because it quantifies the level of risk.

4. Recommendation Engine: From Probability to Action

The `bandwidth_recommendation` function acts as the final decision-logic layer, translating the model's risk scores into concrete, actionable advice. It uses a decision matrix that weighs both the **congestion probability** and the current **bandwidth utilization**.

- **CRITICAL (`prob >= 0.8`)**: The risk of congestion is imminent.
  - **Action**: `increase_bandwidth`. The amount is aggressive (+25% to +40%) to prevent an outage.
- **MODERATE RISK (`prob >= 0.6`)**: The system is showing signs of strain.
  - **Action**: `increase_bandwidth` (+20%) if utilization is also high, otherwise `monitor_closely`.
- **PREVENTIVE (`prob >= 0.4`)**: A proactive adjustment may be needed if the network is busy.
  - **Action**: A small `increase_bandwidth` (+10%) is recommended only if utilization is already high (>85%).
- **OPTIMIZE (`prob <= 0.2`)**: The risk is very low, indicating potential over-provisioning.
  - **Action**: If utilization is also low (<40%), `decrease_bandwidth` (-15%) to improve efficiency and reduce costs.
- **STABLE/NORMAL (all other cases)**: The network is operating within acceptable parameters.
  - **Action**: `maintain` current allocation or `monitor` for changes.

Of course. Here is a comprehensive `README.md` file for your GitHub repository that explains the project's inputs, outputs, and visualizations based on the provided code.

Proactive Network Congestion Dashboard

In dashboard directory there is the python script of streamlit application, which defines the dashboard of project over streamlit, utilizing the models in the corresponding directories.

Input: What the Dashboard Needs

To function correctly, the application requires two main types of input: a primary dataset and pre-trained machine learning assets.

1. Primary Data Source

The core input is a **time-series dataset** of network metrics, typically in a CSV file (or a Pandas DataFrame). This dataset must contain the following columns for the feature engineering process to work:

- `Timestamp`: The date and time of the data record (datetime object).
- `Device Name`: The identifier for the network device (e.g., 'Router_A', 'Router_B').
- `Traffic Volume (MB/s)`: The volume of traffic passing through the device.

- `Latency (ms)`: The network latency at that time.
- `Bandwidth Used (MB/s)`: The amount of bandwidth currently in use.
- `Bandwidth Allocated (MB/s)`: The total bandwidth allocated to the device.
- `total_avg_app_traffic`: A feature representing the average application traffic.
- `total_peak_app_traffic`: A feature representing the peak application traffic.
- `Impact_encoded`: A **numerical** representation of congestion impact (e.g., 0 for 'None', 1 for 'Low', etc.).
- `total_peak_user_usage`: A feature for peak usage by users.
- `total_logins`: The total number of logins.

2. Pre-trained Machine Learning Assets

The application relies on assets generated from a separate model training process.

- **XGBoost Models:** For each router (`A`, `B`, `C`), a pre-trained XGBoost model must be saved as a `.pkl` file using `joblib`. These files should be named `modelA_p.pkl`, `modelB_p.pkl`, etc., and placed in a `models/` directory located one level above the script's folder.
- **Label Encoder:** A fitted `LabelEncoder` object from scikit-learn, saved as `label_encoder_impact.pkl`. This is used to consistently transform the `Impact` feature. The code includes a fallback to create a default encoder if this file is missing.

Output: What the Dashboard Provides

The dashboard processes the input data and produces two key outputs: numerical predictions and actionable recommendations.

1. Congestion Probabilities

For each router, the primary output of the machine learning model is a **congestion probability**. This is a floating-point number between `0.0` and `1.0`, where:

- A value close to `0.0` indicates a **very low risk** of congestion.
- A value close to `1.0` indicates a **very high risk** of congestion.

These probabilities are the core data points used for all subsequent analysis and recommendations.

2. Bandwidth Recommendations

The dashboard translates the raw probabilities into clear, human-readable recommendations for each router. The recommendation engine considers the congestion probability, current bandwidth utilization, and latency to generate a structured output containing:

- **`action`**: A suggested course of action (e.g., `'increase_bandwidth'`, `'monitor_closely'`, `'decrease_bandwidth'`).
- **`amount`**: The recommended change in bandwidth in MB/s. This will be positive for an increase and negative for a decrease.
- **`reason`**: A plain-English explanation for the recommendation, providing context for the administrator (e.g., `'CRITICAL: High congestion probability (0.92) with 95% utilization'`).

How It Visualizes Data

The dashboard uses the Plotly library to create three main interactive visualizations that allow for easy interpretation of the network's health.

1. Traffic Volume Over Time

This is a **line chart** that plots the `Traffic Volume (MB/s)` for each router over the selected time period.

- **Purpose**: To show trends, patterns, and spikes in network traffic.
- **Features**:
  - Each router is represented by a different colored line.
  - A **vertical dashed red line** is drawn on the chart to mark the specific "Prediction Point," giving crucial context to the other visualizations.

2. Congestion Probability (%)

This is a **bar chart** that provides an at-a-glance summary of the current congestion risk.

- **Purpose**: To immediately identify which routers are at risk.
- **Features**:
  - Each bar corresponds to a router, with the bar's height representing the predicted congestion probability (scaled to 100).
  - The bars are **dynamically color-coded** based on risk level:
    - **■ Green:** Low risk (Probability < 40%)
    - **■ Orange:** Medium risk (Probability 40% - 70%)
    - **■ Red:** High risk (Probability > 70%)

3. Current Bandwidth Utilization

This **bar chart** shows how efficiently the allocated bandwidth is being used for each router.

- **Purpose**: To diagnose if high congestion is related to high resource usage.
- **Features**:
  - Each bar shows the utilization percentage ($$\frac{\text{Bandwidth Used}}{\text{Bandwidth Allocated}} \times 100$$).
  - The bars are colored along a **continuous green-yellow-red gradient**, providing a smooth visual indicator of utilization levels.

THE VISUALIZATION SECTION IS UNDER DEVLOPMENT RIGHT NOW.