Future of Food Tech Hackathon - COOKR

- **Aparnaa T (21PC04), Vishnupriya R (21PC39)**

Github link- https://github.com/aparnaat/COOKR.git

Problem Statement : Item Categorization

Approach :

We prepare a dataset which contains food items and their corresponding categories. RandomForest algorithm is used to train the dataset. The dataset is considered as the training part and a model is developed using it. When a new food item is given as input, the model predicts its category. If the item is not present in the dataset, the item along with its predicted category is added to the dataset, else the category is just retrieved.

Implementation details :

For Implementation, a multi-label classification model was constructed using a **RandomForest algorithm** to predict the categories of food items based on their name. The data preprocessing was done using **CountVectorizer**, which transforms the food names into numerical features that the machine learning model can analyze. The **MultiLabelBinarizer** was used to handle the multiple categories associated with each food item. The code uses scikit-learn library for machine learning functionalities like data preprocessing, feature selection etc, and pandas for data manipulation.

Challenges Faced :

We couldn't find a proper dataset for this question and ended up creating our own dataset. With a limited dataset, the prediction is not always accurate and leads to overfitting.Our dataset has only main course items listed and not starters or desserts.

Problem Statement : Last Mile Delivery Batching

Approach :

We created a program to manage food orders for delivery. The code efficiently organizes orders, considers various factors like kitchen, customer, and order time, and assigns delivery riders based on few rules. It allows adding of new orders, and aims to use the least number of riders for optimal delivery. It focuses on making the system efficient for different delivery scenarios.

Implementation details :

The code is implemented in Python. The code defines classes for orders and a delivery scheduler, and uses Python's datetime module for time-related calculations. A defaultdict from the collections module efficiently organizes orders based on kitchen, customer, and ready time. The logic for rider assignment considers different rules, such as orders from the same kitchen or customer, ensuring the optimal delivery scenario. The sorting of orders and riders is also handled.

Challenges Faced :

Since we were not provided with a standard dataset, we provided sample data for testing on our own. Making sure to assign riders based on the given rules was definitely a challenge as there were situations where one rule had to be compromised for another. Debugging and refining the logic was also pretty challenging.

Problem Statement : Predictive Maintenance for Infrastructure

Approach :

Predictive Maintenance for Infrastructure can be done using a concept called **AIOps**.

Artificial intelligence for IT operations (AIOps) is an umbrella term for the use of big data analytics, machine learning (ML) and other AI technologies to automate the identification and resolution of common IT issues.

Aim- to provide good prediction performance while requiring significantly less computational power.

Probes are monitoring tools or agents that are deployed within the cloud infrastructure to gather information and assess the health and performance of different components (e.g., servers,
networks). These probes serve both passively and proactively to maintain QoS.

- Passive Monitoring:

Resource Usage Data Collection: Some probes passively collect data on resource usage, such as *CPU and memory utilization*. This information helps in understanding the current load on the system and identifying potential resource bottlenecks.

Performance Measures: Other passive probes monitor performance measures like *response time and throughput*. These metrics provide insights into how well the system is handling user requests and whether there are any performance degradations.

- Proactive Health Checks:

Heartbeats: Probes may proactively check the health of the system by sending periodic *"heartbeats" or signals*. If a component fails to respond within a specified time, it indicates a potential issue.

Sanity Checks: Probes can perform sanity checks, which involve verifying critical aspects of the system's functionality. For example, *checking whether essential services are running* or if the expected dependencies are in place. If any inconsistencies are detected, appropriate actions can be taken to address the problem.

In order to catch early warning signs of these failures, there are additional applications that scan through the monitoring data and proactively generate **alerts**. Some of these alerts are based on thresholds on the collected system behavior data or error logs.

Once the potential failures are predicted, DevOps engineers would first perform **stress testing** that tests the suspected node with synthetic load, in order to validate if the node is indeed failing. Then, DevOps engineers would perform **live migration** to move the jobs from the failed node to a healthy one which has the same configuration. During the live migration, the complete running status of the virtual machines on the failed node is saved and reloaded in the healthy node, so the system can provide identical services after the migration without impacting the experience of the end-users.

Alerts report problems from different sources (e.g., kernels, drivers, software applications, and hardware components). Each alert contains a **timestamp**, a **verbosity level** (critical, unrecoverable, and recoverable), and a **textual message** describing the error.

Critical level alerts - leads to severe issues if no immediate actions are taken
Recoverable level alerts - can be addressed by self-healing strategies
Unrecoverable level alerts - cannot be addressed by self-healing strategies

Alert count- usually higher around the time of node failure.
The first alert occurs about 60 days before the node failure, and the last alert occurs almost right before the node failure.
However, as alerts can happen anytime, only using the alert counts cannot effectively predict node failures.

APPROACH

(1) **feature engineering** processes the monitoring data and produces useful features for the ML models
(2) **model training** trains the ML models based on the features
(3) **model evaluation** examines the effectiveness of the trained ML models in a production-like usage context.

challenges-
complex data format, data skewness, hyperparameter tuning for deep learning models, and the valid evaluation of the AIOps solution.

(1) Feature engineering-
alerts + **spatial data** (location) + **information about the software and hardware configuration** of a node can be used to predict node failures.

The process of extracting and transforming raw data into features, which are inputs to the ML algorithms, is called feature engineering.

3 ML algorithms can be used - **LSTM, MING,** and **random forest**
major challenge - dealing with complex data formats, as they cannot be directly used in some of the ML models.
soln- feature encoding techniques.
All the monitoring data can be streamed to a central repository every hour.
Therefore, we create a data point for each node in each hour.
For each data point for a node, we can calculate the frequency of alerts in each of the previous hours.

(2) Model Training-
- Build a **baseline model** for each type of alert.
- **LSTM** - deep neural network-based ML algorithm commonly used for predicting **time-series data**. We choose LSTM as it is natural to model the problem of predicting node failures using the temporal features (i.e., time-series features). Temporal features are most influential to the model performance.
- **Random forest** - **ensemble**-based classification algorithm. We choose the random forest algorithm because it's **interpretable** and usually achieves the **best performance** among other traditional classifiers.
- **MING** - hybrid approach, which **combines LSTM and random forest**. Inside MING, there are two ML models: an LSTM model, which learns from the

temporal features, and a random forest model, which learns from the spatial features. During prediction, MING combines the intermediate results from the internal LSTM and the random forest model using a Learning to Rank (LTR) model and outputs one unified prediction outcome. (used by Microsoft Azure cloud)

challenges- data skewness, hyperparameter tuning

(3) Model evaluation
A major challenge faced in the model evaluation phase is how to evaluate the performance of ML models in a production-like context.
Evaluation techniques-
1. **Evaluating the prediction for each node**- For each node and each hourly period containing alerts, a prediction is made.
2. **Just-in-time prediction**- predicting a node failure prematurely (several weeks ahead) would lead to financial losses, as currently-healthy nodes will be mistakenly replaced with newly purchased ones. On the other hand, late predictions (ten seconds before the node failure) would be of no value, as DevOps engineers would not have time to react and perform mitigation actions. So, using a prediction window would solve this problem.

ML models should be re-trained periodically to avoid concept drift (every month).

These steps can be used to predict potential failures or issues in infrastructure components (e.g., servers, networks). This could help DevOps teams proactively address issues before they cause downtime or performance degradation.

Problem Statement : Automated Code Reviews and Quality Assurance

Approach :

The Python script employs a regex-based code scanning approach to identify potential vulnerabilities in a PHP code file based on a predefined dataset of vulnerabilities stored in a JSON file. The script loads information about known vulnerabilities, including CVE IDs, threat levels, summaries, and fixed versions, from the JSON file. It then constructs regex patterns based on keywords present in the vulnerability summaries, ensuring that special characters are treated as literals. The PHP code file is read, and the script searches for matches using the constructed patterns. When a match is found, the script records details such as the CVE ID, threat level, line, column, and the matching text. The results are then displayed, highlighting any identified vulnerabilities in the PHP code. This flexible and configurable approach allows for the extension of the dataset and adaptation to different types of vulnerabilities.

Implementation details :

The script loads vulnerability information from the specified JSON file using the **`json.load`** function. The **`“scan”`** function encapsulates the scanning logic, utilizing regex patterns constructed dynamically for each vulnerability. Special character escape is handled with **`re.escape`**, ensuring precision in matching. The script reads the PHP code file, scans it for vulnerabilities using the constructed patterns, and records findings. Results are then displayed, presenting identified vulnerabilities along with associated details.

Challenges Faced :

Creating an automated code reviewer was challenging, and currently, the above requires the user to manually input the file to be scanned. Again the dataset plays a major role here, we were able to collect dataset that consist of vulnerabilities in a PHP program. Thus our program is exclusively used for PHP files.