TEAM NAME: ChillyFlakes
TEAM MEMBERS:

Thirusha M
Akshith Mohamed M

Statement - Item Categorization.
Colab link:

cookr1.ipynb

Description: Create a model or research the necessary steps to create a model for categorizing items. When the cook adds an item to their kitchen, it should be automatically categorized into multiple categories. We can provide the sample data for this to train the model. For instance:
• Idly - South Indian, Protein Rich, Breakfast, Baked Items etc.
• Chicken Vindaloo - North India, Punjabi, Non-Veg, Chicken, Protein Rich etc.
• Ragi Dosa - South Indian, Diabetic Friendly, Millet Based, Pregnancy friendly
etc.

Approach & Implementation details:

Preprocess Dish Names:

- Uses `TfidfVectorizer` from scikit-learn to convert dish names into numerical features (TF-IDF representation). The transformed features are stored in `X`.

Separate Features and Target:

- Splits the dataset into features (`X` - TF-IDF representation of dish names) and the target variables (`y` - Cuisine, Taste, Meal_Type, Diet_Type).

Train-Test Split:

- Splits the data into training and testing sets (80% training, 20% testing) using the `train_test_split` function.

Train the Random Forest Classifier:

- Creates and trains a Random Forest classifier (`rf_classifier`) using the training data (`X_train` and `y_train`).

Attributes of the dataset:

- Dish
- Cuisine
- Taste

- Meal_Type
- Diet_Type

Challenges:

Data Acquisition Complexity:

- Sourcing an appropriate dataset with a comprehensive set of attributes for food prediction proved intricate and demanded extensive research efforts.

Vectorization Efficiency Concerns:

- *Challenge:* The process of word count vectorization for multiple attributes posed a computational bottleneck, leading to time-intensive operations.So, Implemented an alternative vectorization approach to streamline the word count process, optimizing efficiency and reducing computational overhead.

Accuracy Refinement Endeavors:

- *Challenge:* Achieving optimal predictive accuracy necessitated meticulous tuning and refinement of the model.So,employed systematic tuning strategies to enhance the model's accuracy, ensuring more precise attribute predictions.

Web Integration Complexity:

- Seamlessly integrating the model into the web application presented a multifaceted challenge.
- Applied robust integration techniques to ensure the model functioned seamlessly within the framework of the web application.

Modules used:

Pandas:

Pandas is a popular Python library that is widely used for data manipulation and analysis. It provides high-performance, easy-to-use data structures, and data analysis tools. We have used pandas to read and manipulate the dataset.

TfidVectorizer:

The dataset contains textual information about food items, including dish names and ingredients.Machine learning models, including Random Forest classifiers, require numerical input features.TfidfVectorizer is a text vectorization technique that converts a collection of raw text into a matrix of TF-IDF (Term Frequency-Inverse Document

Frequency) features.This transformation allows the model to learn patterns from text data, enabling efficient and accurate attribute predictions.

RandomForestClassifier:

The RandomForestClassifier module is employed to train ensemble classifiers for predicting various attributes based on TF-IDF features extracted from dish names and ingredients. Its use of multiple decision trees enhances predictive accuracy and robustness, making it suitable for handling complex relationships within the dataset.

train_test_split:

The `train_test_split` module is employed to partition the dataset into training and testing sets, enabling the evaluation of the Random Forest classifiers' generalization performance. This practice helps assess the model's ability to make accurate predictions on new, unseen data by simulating real-world scenarios and preventing overfitting during the training process.

Statement - Predictive Maintenance for Infrastructure

Predictive Maintenance.ipynb

Description: Create AI models or research the necessary steps to create AI models that can predict potential failures or issues in infrastructure components (e.g., servers, networks) based on historical data. This could help DevOps teams proactively address issues before they cause downtime or performance degradation.
For instance:

AI models are trained on historical data regarding cluster performance, node utilization, and workload patterns. These models analyze metrics such as CPU usage, memory consumption, and network traffic to predict potential failures or performance degradation. For instance, if the model detects a gradual increase in CPU usage on certain nodes, it may predict an impending failure due to resource exhaustion. The DevOps team receives automated alerts, allowing them to proactively address the issue by scaling up resources or reallocating workloads before any downtime occurs. This predictive maintenance approach helps optimize the reliability and stability of the EKS environment, ensuring continuous availability of applications running on Kubernetes.

Approach & Implementation details:

Data Loading and Preparation:

- Load the dataset ('simulated_real_time_data_with_failures.csv') into a Pandas DataFrame.
- Define input features (`X`) by excluding timestamp and failure-related columns.
- Set the target variable (`y`) as the 'failure' column.

Data Splitting:

- Split the data into training and test sets using `train_test_split`, reserving 20% for testing.

Baseline Model Training:

- Create a basic Random Forest classifier (`rf`) and train it on the training data (`X_train`, `y_train`).
- Evaluate the model on the test data, calculating and printing the accuracy.

Tree Visualization:

- Visualize the structure of the first three trees in the forest for interpretability.

Hyperparameter Tuning:

- Define a parameter distribution (`param_dist`) for `RandomizedSearchCV` including the number of estimators and maximum depth.
- Utilize `RandomizedSearchCV` to find the best hyperparameters within the specified distribution.

Best Model Evaluation:

- Get the best model from the search (`best_rf`) and print the best hyperparameters.
- Evaluate the performance of the best model on the test data, calculating and printing the accuracy.

Challenges:

Dataset Simulation Constraints:

- Securing a suitable dataset for simulating real-time failure scenarios proved challenging, so explored sources to procure or generate authentic and representative data.

Model Integration with Web Application:

- The seamless integration of the failure prediction model into a web application was a key challenge, demanding expertise in web development and API integration for cohesive architecture.

Live Data Feeding for the Model:

- Establishing a mechanism for continuous live data feeding to the model emerged as a pivotal challenge, requiring the implementation of robust data streaming

Accuracy Tuning for the Model:

- Achieving optimal accuracy by tuning the model's hyperparameters was hard.So,Utilized techniques like RandomizedSearchCV to fine-tune hyperparameters, optimizing the model's performance and enhancing its ability to accurately predict failures.

Modules used:

Numpy:

The numpy module is used to handle numerical operations efficiently in the provided code. It supports array manipulations and computations, aiding tasks such as data splitting, defining hyperparameter distributions, and evaluating model performance, contributing to the overall efficiency and functionality of the machine learning workflow.

RandomForestClassifier:

The RandomForestClassifier module from scikit-learn is utilized to create a Random Forest classifier, a robust ensemble learning algorithm. It is employed for both a basic model and hyperparameter-tuned model training, facilitating accurate classification tasks on the given dataset, while offering versatility and scalability in handling complex relationships within the data.

Accuracy_score:

The accuracy_score module from scikit-learn is employed to quantify the classification accuracy of the Random Forest models on the test data. It provides a reliable metric for evaluating the model's overall predictive performance, measuring the proportion of correctly predicted instances, and aiding in the assessment of model effectiveness during future analyses or comparisons.

Confusion_matrix:

The `confusion_matrix` module from scikit-learn is utilized to assess the performance of the Random Forest models by providing a detailed breakdown of true positive, true negative, false positive, and false negative predictions. It aids in understanding the model's ability to correctly classify instances and is valuable for future evaluations and diagnostic analysis of classification results.

RandomizedSearchCV:

The `RandomizedSearchCV` module from scikit-learn is employed to conduct a randomized search for optimal hyperparameters for the Random Forest classifier. This technique efficiently explores a defined hyperparameter space, helping identify the best model configuration and enhancing the classifier's performance. The module streamlines the hyperparameter tuning process, providing a systematic approach for future model refinement and improvement.

Train_test_split:

The `train_test_split` module from scikit-learn is utilized to partition the dataset into separate training and test sets, allowing the assessment of the Random Forest classifier's performance on unseen data.

Randint:

The `randint` module from the `scipy.stats` library is employed to define a parameter distribution for hyperparameter tuning during the randomized search. Specifically, it generates random integers within specified ranges, facilitating a comprehensive exploration of hyperparameter values and aiding in the optimization of the Random Forest classifier for future model enhancement.

Selectfrommodel:

The `SelectFromModel` module from scikit-learn is used to perform feature selection by leveraging the importance scores assigned to features by a trained Random Forest model. It helps identify and retain the most influential features, contributing to model interpretability, reducing dimensionality, and potentially improving overall model performance in future analyses.

Export_graphviz:

The `export_graphviz` module from scikit-learn is used to generate Graphviz DOT format representations of decision tree structures. This visual representation helps in comprehending and interpreting individual trees within the Random Forest, fostering better understanding and insights into the model's decision-making process for future analysis and communication.

Image:

The `Image` module from IPython.display is used to display images, such as the visualized decision trees, directly within the Jupyter Notebook environment. This enhances the interpretability of the Random Forest models by providing a graphical representation of individual trees during analysis and model development, contributing to improved insights and documentation for future reference.

Graphviz:

The `graphviz` module is used to visualize decision trees from the Random Forest using the Graphviz library. This facilitates a clear interpretation of individual tree structures, aiding in understanding the model's decision-making process. The visualizations serve as valuable insights for model analysis and potential optimizations in future iterations.