ARCHITECTURE

DOCUEMNT

CREDIT CARD DEFAULT PREDICTION

SATYAM SHARMA

DATA SCIENTIST

ARCHITECTURE DESIGN: WHY DO WE NEED IT?

Architecture design plays a pivotal role in the successful development and deployment of any software system, including our Credit Card Default Prediction System. It serves as the blueprint that defines the system's structure, components, relationships, and how they function together. An effective architecture design not only provides a clear roadmap for implementation but also ensures scalability, maintainability, and reliability of the system.

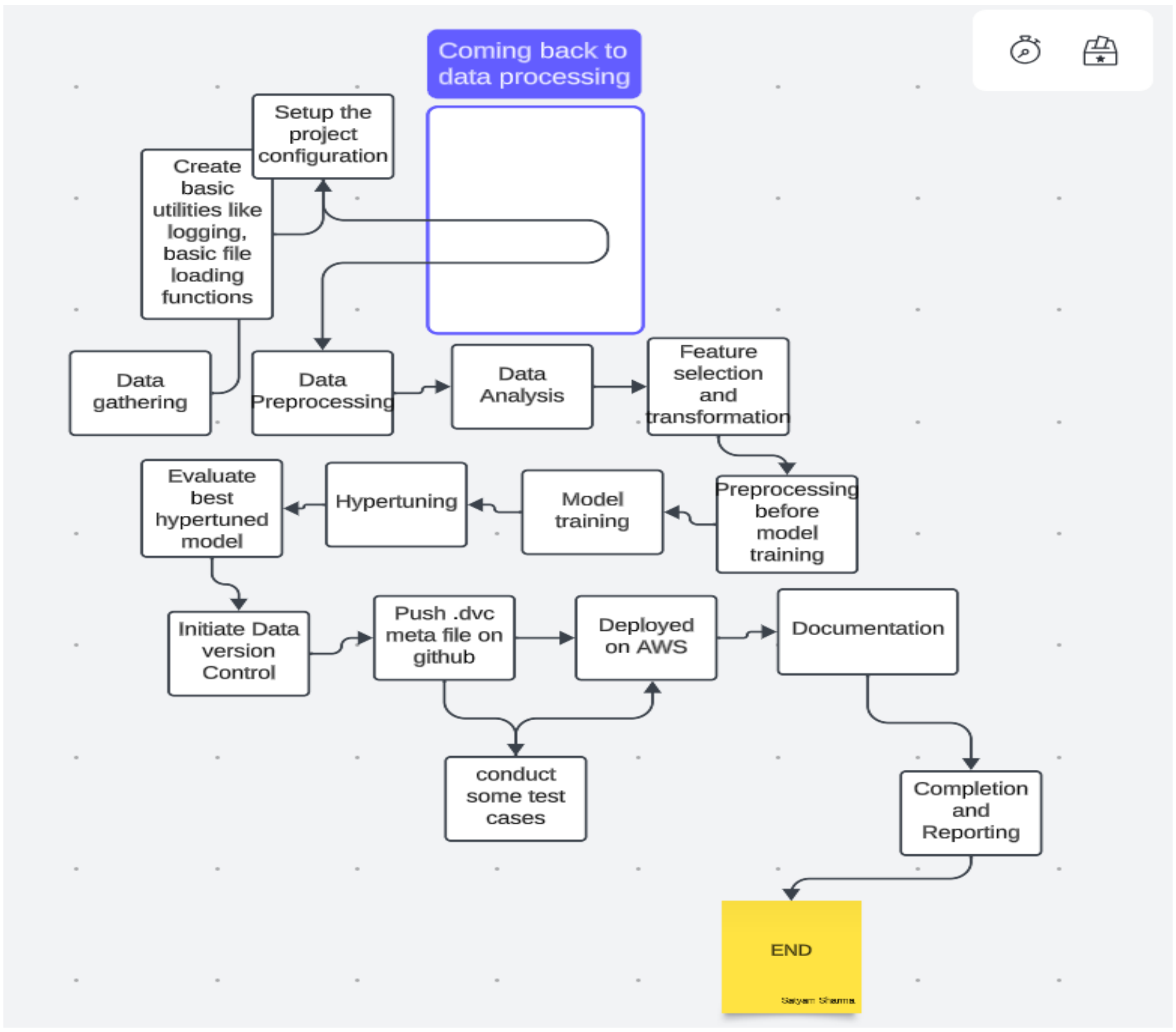
Complexity Management: Our system incorporates multiple components, including data processing, machine learning models, a user interface, and more. A well-defined architecture helps manage this complexity by organizing and structuring these components in a comprehensible manner.

Scalability: As the system's user base and data volumes grow, it must be able to scale efficiently. A well-designed architecture allows for modular growth, ensuring that new features or improvements can be seamlessly integrated.

Performance: The architecture design influences the system's performance. By optimizing the flow of data and processes, we can ensure that predictions are made efficiently and accurately.

Maintainability: Over time, the system will require updates, bug fixes, and enhancements. A well-structured architecture simplifies maintenance by providing a clear delineation of system components and their interactions.

CREDIT CARD DEFUALT ARCHITECTURE


Coming back to data processing
Setup the project configuration
Create basic utilities like logging, basic file loading functions
Data gathering
Data Preprocessing
Data Analysis
Feature selection and transformation
Preprocessing before model training
Model training
Hypertuning
Evaluate best hypertuned model
Initiate Data version Control
Push .dvc meta file on github
Deployed on AWS
Documentation
conduct some test cases
Completion and Reporting
END
Satyam Sharma

PROCESS

1. Data Gathering

Data Acquisition from Kaggle

Kaggle provided us with a dataset that contains a wealth of information relevant to our credit card default prediction model. The dataset encompasses various features, including credit limits, repayment history, bill amounts, payments, and the target variable indicating credit card default status.

2. Create Basic utilities, logging, error handling

Logging Mechanism

A comprehensive logging system is implemented to capture relevant information and events during the execution of our system. Logging is crucial for tracking the system's behaviour, diagnosing issues, and monitoring performance. The logging mechanism records critical information, warnings, errors, and other events, providing a clear audit trail for system activities.

Error Handling

Effective error handling is a critical aspect of ensuring the reliability and robustness of our system. It involves the creation of error handling routines that can gracefully manage exceptions and unexpected situations. Well-structured error handling ensures that the system can recover from errors, provides informative error messages, and minimizes downtime.

Rationale for Process

- *Code Organization*: Developing basic utilities ensures a well-organized and modular codebase, making it easier to manage and extend the system.
- *Transparency and Debugging:* A robust logging system enhances transparency, aiding in debugging, troubleshooting, and tracking the system's performance.
- *Resilience:* Effective error handling mechanisms enhance the system's resilience by preventing crashes and unhandled exceptions.

3. Setup project configuration

    The "Setup Configuration" process is the cornerstone of our Credit Card Default Prediction System's architecture, shaping its adaptability, security, and maintainability. In this expanded process, we emphasize the integration of Python OOP concepts and a modular approach while configuring the entire project in Visual Studio Code (VSCode).

Python OOP and Modular Approach

- In the setup phase, we embrace the power of Python's Object-Oriented Programming (OOP) to create structured and reusable code. OOP allows us to encapsulate data and functionality into classes and objects, promoting code organization and clarity. This modular approach not only enhances code maintainability but also supports scalability and extensibility.

4. Data preprocessing
    - Data preprocessing is a critical phase in our Credit Card Default Prediction System, following the modular coding setup. In this process, we conduct a comprehensive analysis of the dataset, where we diligently examine the data features, rename columns for clarity, and remove unnecessary or unknown categories to enhance the quality and utility of the data.
5. Data Analysis
    - Column Renaming: To improve the clarity of our data, we engage in column renaming. This step involves providing more descriptive and intuitive names to the columns, making it easier to understand and work with the dataset.
    - Useless and Unknown Categories: During the feature analysis, we identify and remove categories or columns that provide no significant information or contain unknown or irrelevant data. This cleansing process streamlines the dataset, improving the efficiency of subsequent data processing.
6. Feature selection and transformation

Feature selection and transformation are pivotal phases following data preprocessing, where we further enhance the dataset's quality and utility. In this process, we apply advanced techniques to merge columns related to dues and previous payments and employ one-hot encoding to handle categorical columns, such as sex, marriage, and education.

Feature Merging

Dues Columns: We consolidate information from different months of dues, merging them into a single feature that represents the overall credit behaviour. This consolidation simplifies the dataset while retaining essential credit history information.

Previous Payments Columns: Similarly, we merge columns related to previous payments into a single feature, creating a comprehensive indicator of the payment history. This transformation offers a consolidated view of payment behaviour.

One-Hot Encoding

Categorical Data: To handle categorical columns like sex, marriage, and education, we apply one-hot encoding. This process converts categorical variables into binary values, enabling machine learning models to utilize these features effectively.

7. Preprocessing before model feeding

Before feeding the data into the predictive models, it is essential to conduct a final round of preprocessing to ensure that the dataset is in an optimal state for model training and evaluation. In this process, we remove the columns that were left unmerged after feature selection and transformation.

Final Data Cleansing

Column Removal: We identify the columns that were not merged during the feature selection and transformation phase and proceed to remove them from the dataset. These columns are typically those that do not provide significant predictive power or may introduce noise.

Final Check for Data Quality: We perform a final check for data quality, ensuring that there are no missing values or anomalies that could affect model training and prediction.

Data Splitting

- To prepare for model training and evaluation, we split the dataset into training and testing subsets. The training set is used to train the model, while the testing set assesses the model's performance on unseen data.

8. Model training

Model training is a pivotal phase in our Credit Card Default Prediction System. In this process, we train predictive models using four distinct dataframes: the simple normal dataframe, the scaled SMOTE balanced dataframe, the scaled balanced dataframe, and the unscaled unbalanced dataframe. The goal is to assess the performance of various models on these datasets to determine the most suitable configuration.

Dataframe Selection

- Simple Normal Dataframe: This dataframe represents the original dataset without any scaling or balancing. It serves as a baseline for model training.
- Scaled SMOTE Balanced Dataframe: This dataframe is a scaled version of the dataset, and SMOTE (Synthetic Minority Over-sampling Technique) has been applied to balance the classes. It aims to evaluate the impact of data balancing and scaling on model performance.
- Scaled Balanced Dataframe: Here, the dataset is scaled without applying SMOTE, and class balance is achieved. This dataframe helps us assess the influence of scaling alone on model outcomes.
- Unscaled Unbalanced Dataframe: This dataframe retains the original data distribution without any scaling or balancing. It is crucial for understanding the effect of class imbalance on model predictions.

Model Choice: In this phase, we train a variety of classification models, including Logistic Regression, Support Vector Classifier, Decision Tree, Random Forest Classifier, XGBoost Classifier, and AdaBoost Classifier. Each model is trained on all four dataframes.

9. Hyper tuning

In the "Model Hyper-Tuning" process, we focus on optimizing the parameters of our selected model, the AdaBoost Classifier. By fine-tuning the hyperparameters, we aim to enhance the model's performance and predictive accuracy.

Hyperparameter Tuning for AdaBoost Classifier

- Parameter Selection: We identify the hyperparameters of the AdaBoost Classifier that can be tuned to improve its performance. Common hyperparameters for AdaBoost include the number of estimators (n_estimators), the learning rate (learning_rate), and the algorithm used for boosting (algorithm).
- *GridSearch:* To systematically explore the hyperparameter space, we employ GridSearch, a technique that performs an exhaustive search over a predefined set of hyperparameter values. GridSearch helps us identify the optimal combination of hyperparameters that results in the best model performance.
- Hyperparameter Combinations: We specify a range of values for the hyperparameters, and GridSearch evaluates all possible combinations. For instance, for "n_estimators," we may consider values like 30, 50, 70, and 100.

10. Model Evaluation

- The "Model Evaluation" process is a pivotal phase in our Credit Card Default Prediction System. In this process, we assess the performance of the tuned AdaBoost Classifier using a combination of evaluation metrics, including ROC (Receiver Operating Characteristic) curves, calibration curves, accuracy scores, and additional metrics such as precision, recall, F1-score, and support.

ROC Curve Analysis

ROC Curves: We plot the Receiver Operating Characteristic (ROC) curves to evaluate the model's ability to discriminate between positive and negative classes. The ROC curve illustrates the trade-off between sensitivity (true positive rate) and specificity (true negative rate).

AUC (Area Under the Curve): We calculate the Area Under the ROC Curve (AUC) to quantify the overall performance of the model. A higher AUC indicates better discrimination.

Calibration Curve

Calibration Plot: We generate a calibration curve to assess the calibration of the model's predicted probabilities. This curve illustrates how well the predicted probabilities align with the observed outcomes. Ideally, the curve should closely follow the 45-degree diagonal line.

Accuracy Score

Accuracy Evaluation: We re-evaluate the model's accuracy score, which is a key metric for assessing the model's overall performance.

Precision, Recall, F1-score, and Support

Precision: Precision measures the model's ability to correctly identify true positives among the predicted positive cases.

Recall: Recall assesses the model's ability to capture all actual positive cases.

F1-Score: The F1-score is the harmonic mean of precision and recall, providing a balanced assessment of the model's performance.

Support: Support indicates the number of instances in each class, which can provide insights into class distribution.

11. Initiate Data Version Control (DVC)

In the "Initiate Data Version Control (DVC)" process, we implement Data Version Control (DVC) to manage and track changes to our dataset. DVC is used to efficiently handle large datasets, ensuring that data remains organized, accessible, and shareable among team members.

DVC Setup

DVC Installation: We install and set up DVC as a version control tool for our dataset. DVC provides a simple and efficient way to manage large files and track changes without storing the actual data in version control.

Repository Initialization: We initialize a DVC repository in the project directory, which serves as a central location for managing dataset versions

DVC Workflow

DVC Meta-Files: DVC creates meta-files that store information about the dataset, including file sizes, hashes, and data file locations. These meta-files are tracked in version control, allowing team members to access them without downloading the entire dataset.

Data Versioning: When changes are made to the dataset, DVC creates new versions and updates the meta-files. This ensures that the dataset's history is preserved, and it is easy to roll back to previous versions if needed.

Collaboration: DVC enables collaboration by allowing team members to access the latest meta-files and data versions. It streamlines the sharing of large datasets without the need to transmit the entire dataset.

12. Push project on github

In the "Pushing the Complete Project to GitHub" process, we ensure that our entire project, including the source code, dataset, and associated files, is uploaded to a GitHub repository. GitHub serves as a centralized platform for version control, collaboration, and project sharing.

13. Deploy on AWS

In the "Deploying on AWS (Amazon Web Services)" process, we take our Credit Card Default Prediction System and make it accessible via Amazon Web Services (AWS). This enables us to host our system in the cloud, making it available for use on the web.

AWS Setup

Account Creation: If not already done, we create an AWS account to access AWS services.

EC2 Instance: We set up an Elastic Compute Cloud (EC2) instance, which will serve as the hosting environment for our system. EC2 instances provide scalable compute capacity in the cloud.

Security Groups and Key Pairs: We configure security groups to control inbound and outbound traffic to the EC2 instance and create key pairs for secure remote access.

Deployment

System Deployment: We deploy our Credit Card Default Prediction System on the EC2 instance. This involves transferring the project files, code, and any necessary dependencies to the instance.

Web Server Setup: We configure a web server (e.g., Apache or Nginx) to host the user interface of our system. The web server listens for incoming requests from users.

14. Documentation

The "Documentation" process is a critical phase in our project, ensuring that all aspects of the system, from its design and functionality to its usage and maintenance, are thoroughly documented. Effective documentation aids in system understanding, support, and future development.

15. Completion and Reporting

Conclusion

In conclusion, our Credit Card Default Prediction System stands as a testament to the seamless integration of financial acumen and cutting-edge technology. As financial institutions navigate the complexities of credit risk management, our system offers a robust, data-driven solution that enhances decision-making and helps mitigate credit card defaults. The successful deployment of the system on AWS, coupled with thorough documentation and version control, ensures that it remains a valuable asset for organizations looking to make informed lending decisions.