A list of text on a white background

Description automatically generated

A diagram of a software system

Description automatically generated

A diagram of a process

Description automatically generated

Flow:

* Write down the question
* Clarifying question, Assumptions, and Constraints, Requirements, Scale
  + Ask what possible actions the user can take
  + What are trying to maximize/ minimize
  + Clarify constraints: budget, time, resources
  + Define the scale: number of users, transactions, etc.
* Frame as ML problem
  + What will be the input
  + What will be the output
  + Define the problem type: classification, regression, ranking, clustering, etc.
* Data

**Feature identification and structuring:**

* Different buckets of features (e.g., user, item, context, etc.)
* What type of data will each feature have? (e.g., categorical, numerical, text, images, etc.)
* Convert raw data into meaningful features:
  + Text: TF-IDF, word2vec, transformer embeddings, or pre-trained models
  + Images: CNN feature extraction, embeddings
* **Feature Engineering:**
  + Which features to normalize? (e.g., numerical features like age, salary)
  + Pre-trained models for embeddings
  + Bucketizing and one-hot encoding for categorical data
  + Hashing for large cardinality categorical variables (e.g., user IDs)
  + Feature concatenation to form a composite feature vector
  + **Handling missing values**
* **Feature selection**: Which features will be most important for the model?
  + Data imbalance: How to handle it (e.g., oversampling, undersampling, class weights)
  + How to split the data
* **Model Selection**
* Choose the model type(s) based on the problem (e.g., decision trees, SVM, neural networks, XGBoost, etc.)
* For ranking: pairwise ranking loss, ranking models like RankNet or LambdaMART
* Consider using ensemble methods if appropriate (e.g., Random Forest, Gradient Boosting)
* **Data splitting**
* Train-test split, cross-validation strategies
* Time-series data split (if applicable)
* How to handle data leakage

 **Training Strategy**

* Batch vs. online learning
* Handling model convergence (e.g., learning rate tuning, gradient clipping)
* Hyperparameter tuning (e.g., grid search, random search, Bayesian optimization)
* Cross-validation to prevent overfitting

 **Evaluation**

* **Evaluation metrics:** Define the metric(s) aligned with business goals (e.g., precision for fraud detection, recall for medical diagnoses)
* **Validation:** How will you validate the model on unseen data? (e.g., A/B testing, backtesting)
* **Performance monitoring:** How will the model’s performance be monitored over time?
* **Error analysis:** Identifying failure modes, addressing underperformance

 **Scalability and Robustness**

* **Handling scale**: How will the system scale with growing data or users? (e.g., distributed training, data sharding)
* **Real-time considerations**: How to deploy the model for low-latency predictions (e.g., microservices architecture, batch vs. online serving)
* **Fault tolerance and reliability**: Dealing with partial outages, retries, etc.

 **Deployment & Monitoring**

* Model deployment pipeline (e.g., Docker, Kubernetes, CI/CD)
* Monitoring deployed model performance in production
* Versioning models and datasets (e.g., DVC, MLflow)
* Retraining strategy (how often to retrain, or when the model’s performance drops)
* **Model explainability**: Can the predictions be interpreted? If necessary, use tools like LIME, SHAP

 **Business Impact and Iteration**

* **Business feedback loop**: How will you collect feedback from end-users or stakeholders?
* Iterative improvements: How to iterate on the model once it's deployed (e.g., incorporating new data, retraining)