Photo by Alessandro Russo on Unsplash

Member-only story

Machine Learning System Design Interview Cheat Sheet-Part 1

Theory, Concepts, Processes, Tools, and Libraries

Senthil E · Follow

Published in Analytics Vidhya · 47 min read · Apr 24

Introduction:

Machine learning system design is a crucial aspect of developing effective AI solutions. It encompasses the entire process of creating, deploying, and maintaining machine learning models, ensuring they perform optimally and meet specific goals. This cheat sheet offers a concise and easy-to-understand guide to the key components of machine learning system design, including data preparation, feature engineering, model selection, optimization, and deployment. Let's dive in.

Image by the Author

Problem definition:

Clearly understand the problem to be solved and its scope.

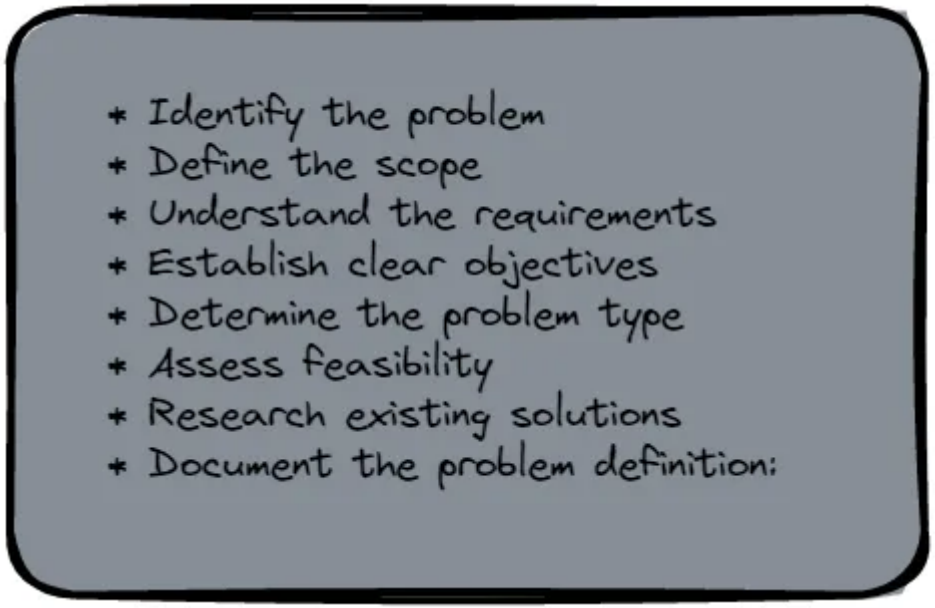


Image by the Author

Objective:

Define the objective, success criteria, and key performance indicators (KPIs).

Image by the Author

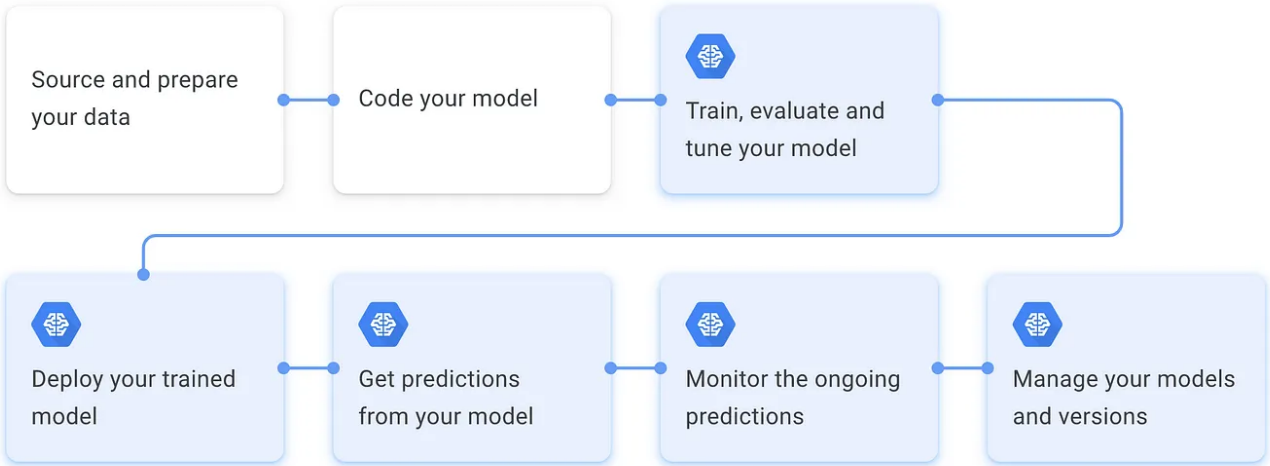


Image credit Google Cloud Documentation

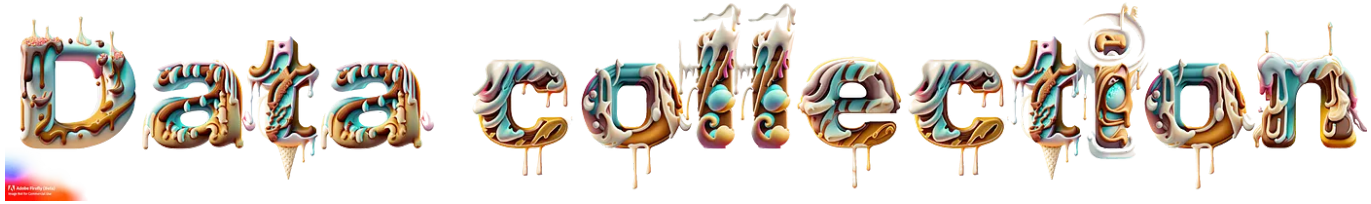


Image by the Author-Adobe Firefly

Identify relevant data sources, collection techniques, and storage.

Data Sources:

Image by the Author

Airtable Download CSV View larger version

Data Formats:

Airtable Download CSV View larger version

Image by the Author

Tools:

- *Google Cloud Storage*
- *Amazon S3*
- *Azure Blob Storage*
- *PostgreSQL*
- *MySQL*
- *MongoDB.*

Image by the Author — Adobe Firefly

Missing values:

- Missing values are data points that are absent for one or more features.

- They can occur due to various reasons, such as data entry errors, equipment malfunctions, or incomplete data collection.
- Identify the missing values and their patterns in the dataset to determine their impact on the analysis and the appropriate handling method.
- Example: In a medical dataset, some patients might not have blood test results due to their unavailability during data collection. Missing values can be identified using functions like `isnull()` in pandas or `isnan()` in NumPy.

Outliers:

- Outliers are data points that deviate significantly from the overall distribution or pattern of the data.
- They can be caused by data entry errors, measurement errors, or genuine extreme observations.
- Identify and analyze outliers to determine their cause and decide whether to retain, correct, or remove them from the analysis.
- Example: In a dataset of house prices, a mansion with an unusually high price may be a genuine outlier. Outliers can be detected using

techniques like box plots, Z-score, IQR, or Tukey's fences.

Duplication:

- Duplicate records are instances that appear more than once in the dataset, either as exact copies or with minor variations.
- Duplicates can occur due to data entry errors, merging datasets, or other data processing issues.
- Identify and remove duplicates to avoid biased estimates and overfitting in the machine learning model.
- Example: In a customer dataset, a person might appear twice with slightly different names due to a data entry error. Duplicates can be identified and removed using functions like `duplicated()` and `drop_duplicates()` in pandas.

Data inconsistencies and errors:

- Data inconsistencies and errors are discrepancies or inaccuracies in the data that can affect the quality and reliability of the analysis.
- These can include typos, formatting errors, incorrect units of measurement, or inconsistent encoding of categorical features.

- Identify and correct data inconsistencies and errors to ensure the data is accurate, consistent, and suitable for analysis.
- Example: In a survey dataset, a categorical feature like ‘gender’ might be inconsistently encoded with values like ‘M’, ‘F’, ‘Male’, ‘Female’, ‘m’, and ‘f’. These inconsistencies can be corrected by standardizing the encoding to a consistent format, such as ‘M’ and ‘F’.

Image by the Author-Adobe Firefly

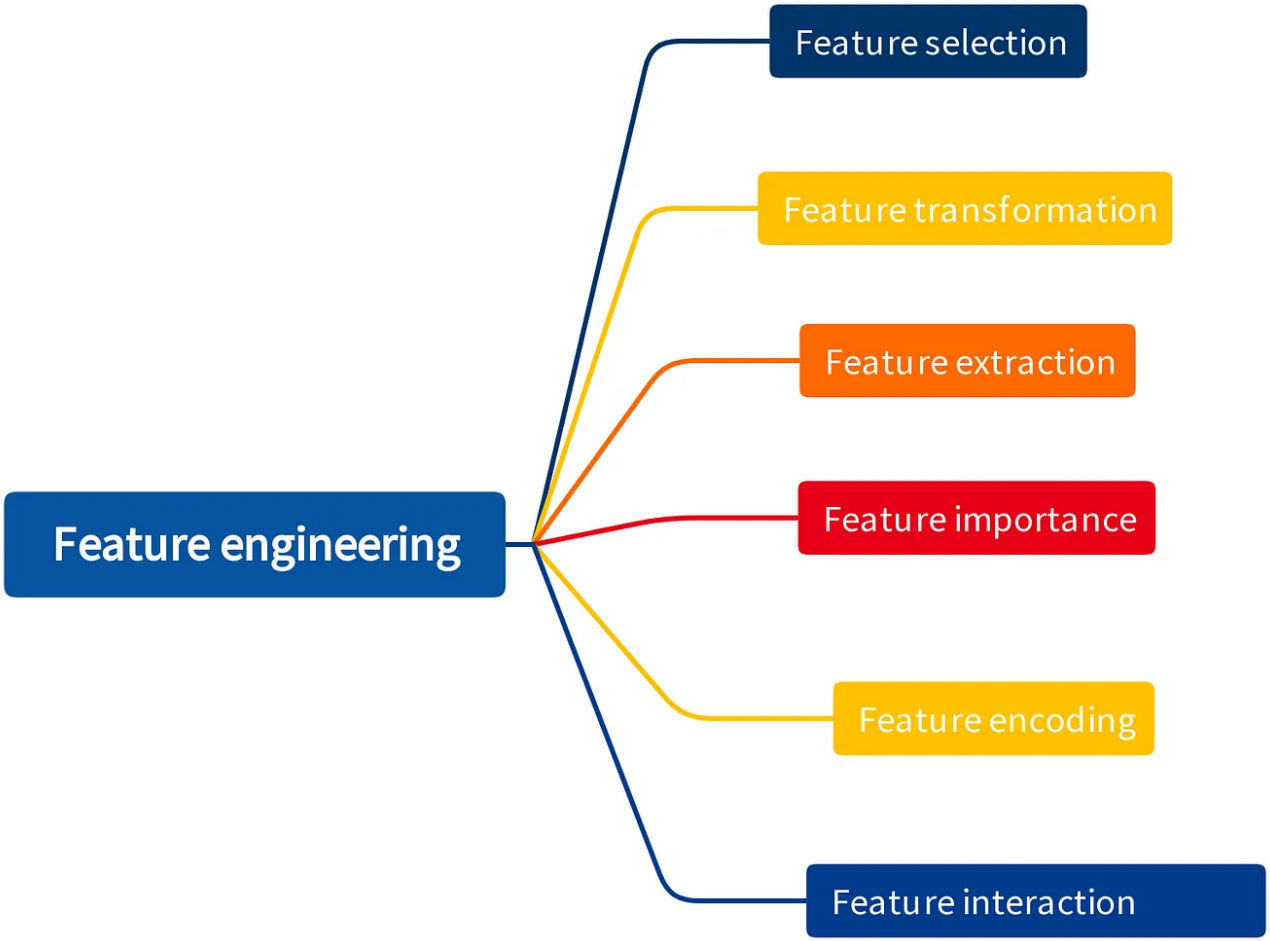


Image by the Author

Feature selection:

- The goal is to identify and retain the most relevant features while removing irrelevant or redundant ones.

- Techniques include filter methods (e.g., correlation-based feature selection, mutual information), wrapper methods (e.g., recursive feature elimination, forward/backward selection), and embedded methods (e.g., LASSO, Ridge Regression).
- Example: Removing features with near-zero variance, as they don't contribute much to the model's predictive power.

Feature transformation:

- Transforming features can help improve model performance by making data more suitable for the learning algorithm or by revealing hidden patterns.
- Common transformations include normalization (e.g., Min-Max scaling, Z-score standardization), log transformation, power transformation (e.g., Box-Cox, Yeo-Johnson), and discretization (e.g., binning, quantile-based discretization).
- Example: Log-transforming a feature with a skewed distribution to make it more normally distributed and improve linear model performance.

Feature extraction:

- Extracting new features from existing ones can help capture additional information or simplify the feature set.
- Techniques include dimensionality reduction (e.g., PCA, t-SNE, UMAP), feature aggregation (e.g., combining related features), and extracting features from raw data (e.g., text, images, time series).
- Example: Extracting the length and the number of vowels from a text feature to create two new numerical features for a spam classification task.

Feature encoding:

- Encoding is used to convert non-numerical features (e.g., categorical, ordinal) into a numerical format suitable for machine learning algorithms.
- Techniques include label encoding, one-hot encoding, ordinal encoding, target encoding, and binary encoding.
- Example: Using one-hot encoding to convert a categorical feature like ‘color’ with values ‘red’, ‘green’, and ‘blue’ into three binary features (‘is_red’, ‘is_green’, ‘is_blue’).

Feature interaction:

- Creating new features by combining or interacting with existing features can help capture complex relationships in the data.
- Techniques include adding interaction terms (e.g., multiplying two features), polynomial features (e.g., raising features to a power), and using domain knowledge to create meaningful combinations.
- Example: For a real estate price prediction model, create a new feature called 'price_per_square_foot' by dividing the price by the area of the property.

Feature importance:

- Identifying important features can help in understanding the underlying patterns and relationships in the data, as well as in simplifying the feature set.
- Techniques include permutation importance, feature importance from tree-based models (e.g., Random Forest, XGBoost), and coefficients from linear models.
- Example: Using permutation importance to rank features based on their impact on the model's performance, and then selecting the top k

most important features.

Image by the Author-adobe Firefly

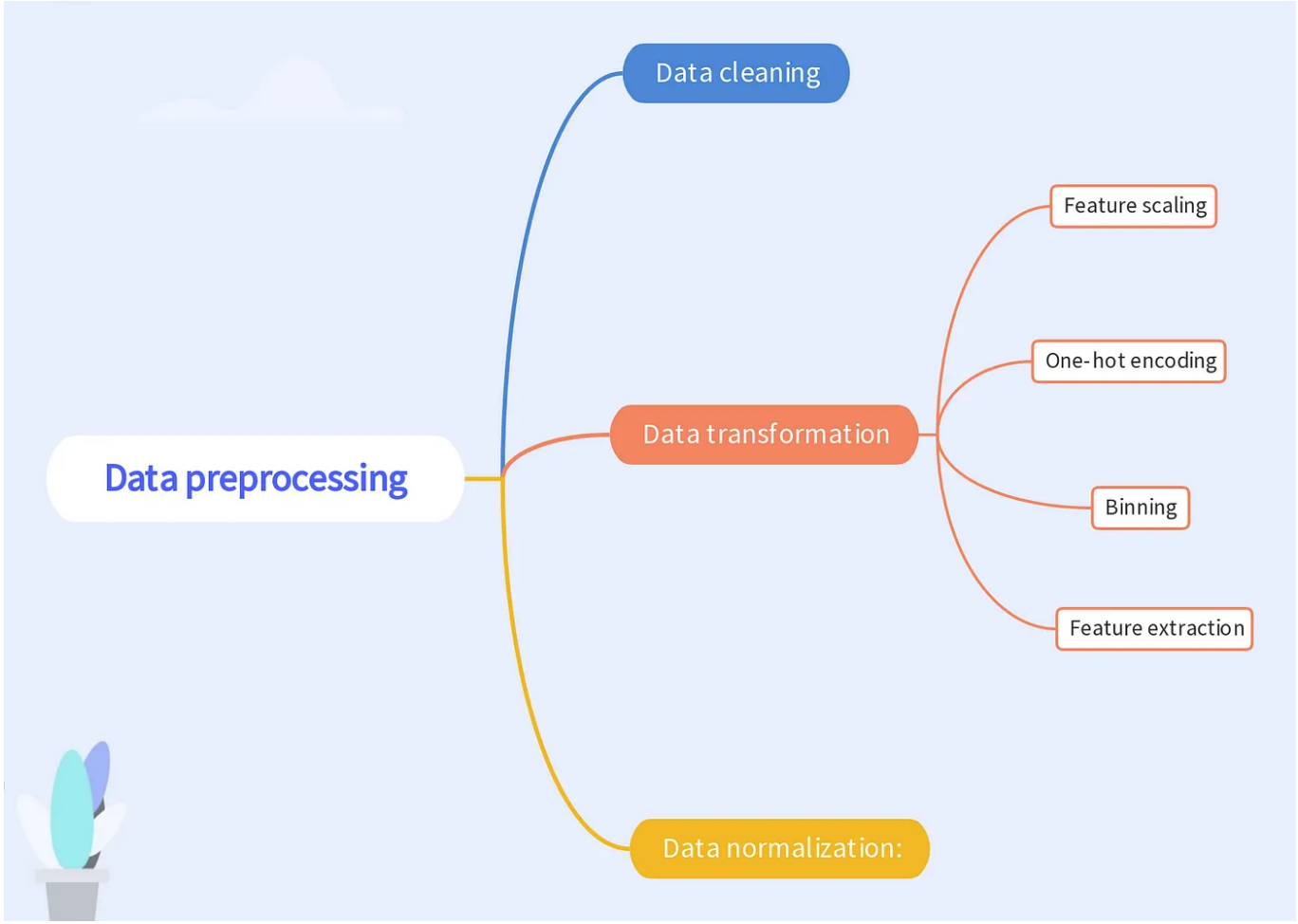


Image by the Author

Data cleaning:

- Address missing values by either removing instances with missing data, filling in the missing values with a suitable method (e.g., mean,

median, mode, or interpolation), or using a model to predict the missing values.

- Remove or correct outliers if they are due to errors or if they negatively impact model performance. Alternatively, use robust algorithms that are less sensitive to outliers.
- Remove duplicates to avoid overfitting and biased estimates.

Data transformation:

- **Feature scaling:** Scale numerical features to ensure that they have similar ranges and units, which helps improve model convergence and performance. Common scaling methods include min-max scaling, standardization (z-score normalization), and log transformation.
- **One-hot encoding:** Convert categorical features into binary dummy variables to ensure compatibility with machine learning algorithms that require numerical input.
- **Binning:** Convert continuous features into discrete bins to reduce noise, handle outliers, or simplify the representation.

- **Feature extraction:** Create new features from the existing ones to improve model performance or reduce dimensionality. Examples include polynomial features, interaction terms, or applying domain-specific transformations.

Data normalization:

- Normalize the data to ensure that it follows a standard distribution (e.g., Gaussian distribution) if required by the chosen machine learning algorithm. Techniques include log transformation, Box-Cox transformation, or Yeo-Johnson transformation.
- Note that not all machine learning algorithms require normalized data, and normalization might not always be necessary.

Image by the Author-Adobe Firefly

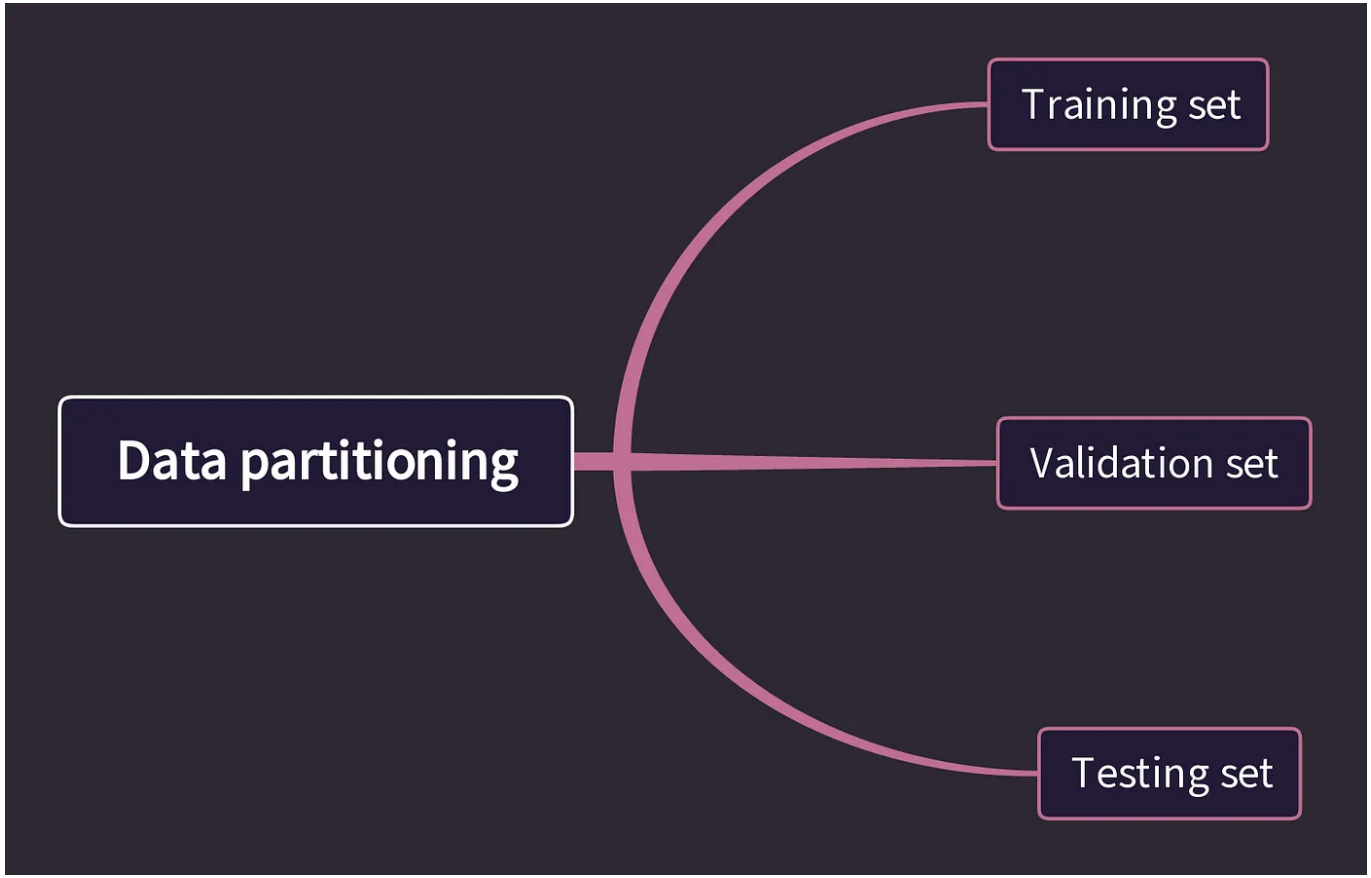


Image by the Author

Training set:

- The training set is the largest portion of the dataset, used to train the machine learning model.
- It helps the model learn patterns and relationships in the data.

- Typically, the training set comprises around 60–80% of the entire dataset.

Validation set:

- The validation set is used to evaluate the model's performance during the model selection and hyperparameter tuning process.
- It helps identify overfitting, underfitting, and the best model architecture or hyperparameters.
- The validation set usually constitutes around 10–20% of the dataset.

Testing set:

- The testing set is used to evaluate the final model's performance, providing an unbiased estimate of its generalization ability on unseen data.
- The testing set should only be used once after the model has been selected and tuned using the training and validation sets.
- The testing set typically comprises around 10–20% of the dataset.

To ensure proper data partitioning, consider the following best practices:

1. *Random sampling:*
2. *Stratified sampling*
3. *Time-based partitioning*
4. *K-fold cross-validation*

Model selection:

Choose the appropriate ML model based on problem type and data.

- Model evaluation, comparison, and selection based on performance metrics

Model training:

Train the selected model using the training dataset.

Model evaluation:

Evaluate model performance using validation and test datasets.

Image by the Author

Hyperparameter tuning:

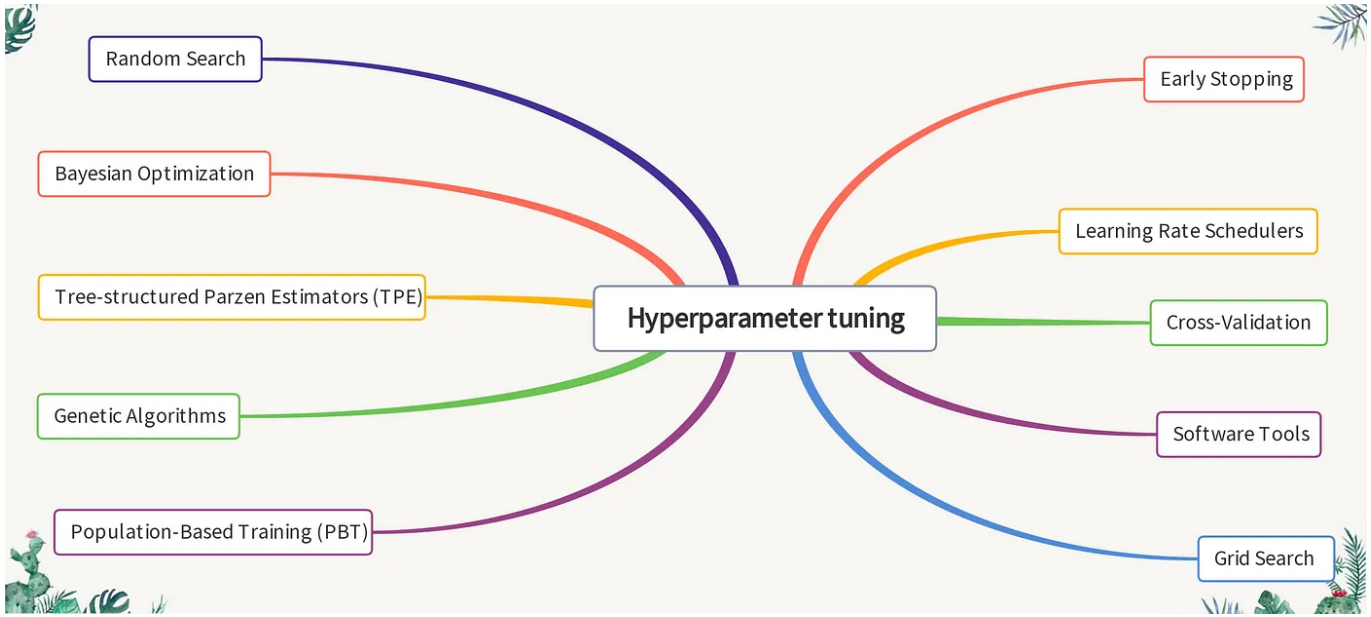


Image by the Author

1. **Grid Search:** Exhaustively search through a predefined set of hyperparameter values, evaluating each combination.
2. **Random Search:** Sample hyperparameter values randomly from specified distributions, covering a wider search space.
3. **Bayesian Optimization:** Model the objective function using a surrogate model (e.g., Gaussian Process) and iteratively select

hyperparameters based on acquisition function.

4. **Tree-structured Parzen Estimators (TPE):** Model the probability distribution of hyperparameters given the past performance, and sample the most promising regions.
5. **Genetic Algorithms:** Use evolutionary algorithms to evolve a population of hyperparameter combinations, applying mutation and crossover operations.
6. **Population-Based Training (PBT):** Train models with different hyperparameters simultaneously, periodically adjusting hyperparameters based on the performance of other models in the population.
7. **Early Stopping:** Terminate the training process when performance on the validation set stops improving, saving time and computational resources.
8. **Learning Rate Schedulers:** Adjust the learning rate during training using schedulers (e.g., step decay, cosine annealing) to find an optimal learning rate.
9. **Cross-Validation:** Use k-fold cross-validation or stratified k-fold cross-validation to estimate model performance across different hyperparameter combinations.

10. **Software Tools:** Employ tools like scikit-learn, Optuna, or Hyperopt for implementing and automating hyperparameter tuning strategies.

Image by the Author-Adobe Firefly

1. **Bagging:** Train multiple base models independently using bootstrapped samples, and combine their predictions through averaging or majority voting.
2. **Boosting:** Train models sequentially, adjusting instance weights to focus on misclassified samples, and combine predictions using a weighted majority vote.
3. **Stacking:** Train multiple base models, and use their predictions as input for a meta-model, which learns to combine them optimally.
4. **Blending:** Similar to stacking, but use a validation set to train the meta-model instead of cross-validated predictions from base models.
5. **Weighted Averaging:** Combine model predictions by assigning different weights to each model based on their performance or other

criteria.

6. **Rank Averaging:** Rank the predictions of individual models, average the ranks, and convert them back to predictions.
7. **Bayesian Model Averaging:** Combine model predictions using Bayesian probabilities, considering both the model's performance and uncertainty.
8. **Ensemble Selection:** Train a large number of candidate models and select a subset with the best performance or diversity to create an ensemble.
9. **Majority Voting:** Combine the predicted classes of multiple models by choosing the class with the highest vote count.
10. **Error-Correcting Output Codes (ECOC):** Train multiple binary classifiers on subsets of the output classes and use the ensemble to predict multiclass problems.

Image by the Author

- *LIME (Local Interpretable Model-Agnostic Explanations)*
- *SHAP (SHapley Additive exPlanations)*
- *Feature Importance*
- *Partial Dependence Plots (PDP)*
- *Individual Conditional Expectation (ICE) plots*
- *Permutation Importance*
- *Global Surrogate Models*
- *Attention Mechanisms*

Image by the Author-Adobe Firefly

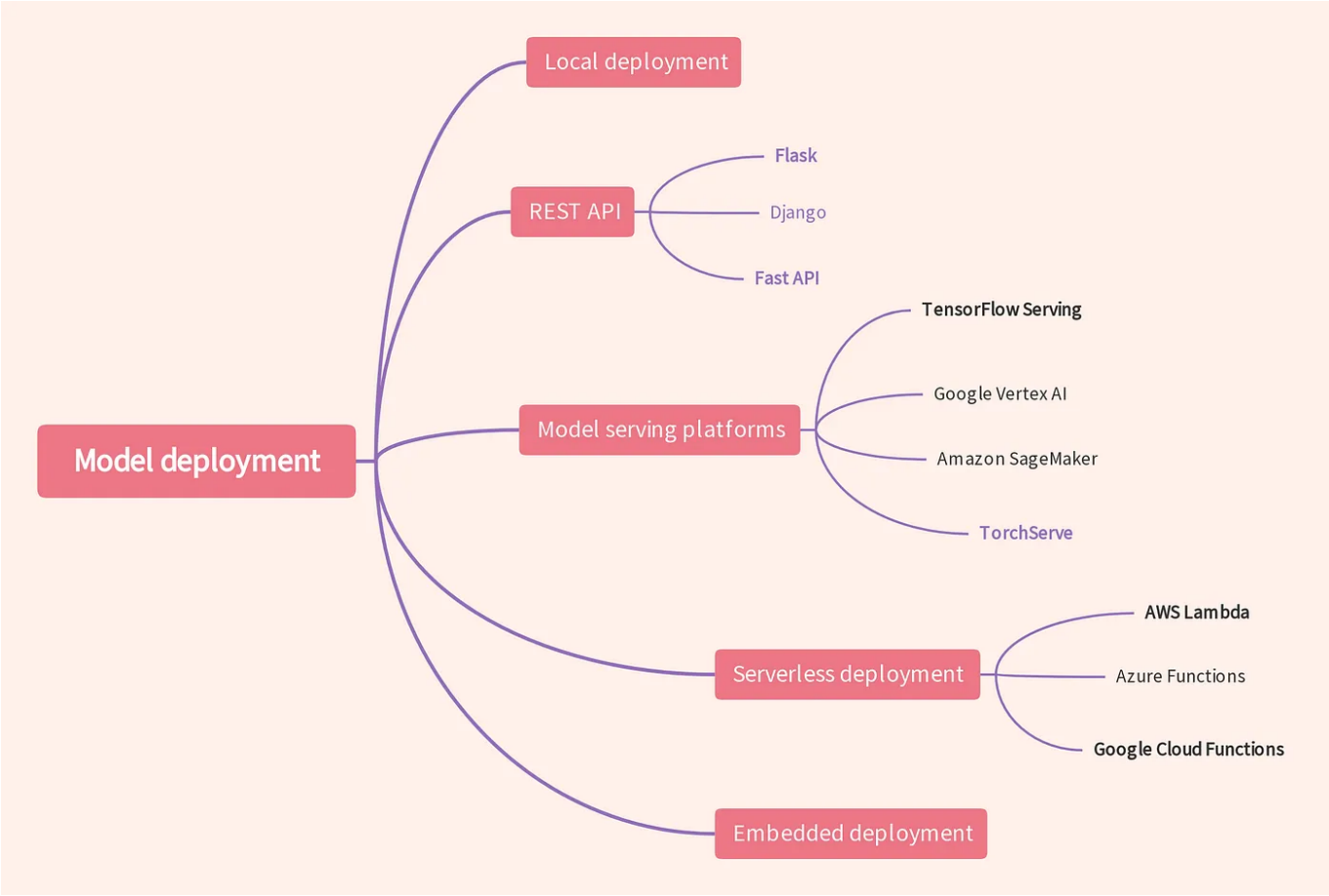


Image by the Author

Image by the Author

Local deployment:

- Deploy the model directly within the application or service, typically by including the model's files and using the appropriate libraries to load and use the model.
- Pros:
- Low latency, as the model runs on the same machine as the application.
- No network overhead or dependency on external services.
- Cons:
- Limited scalability, as the model shares resources with the application.
- Difficulty in updating the model or handling multiple versions.

REST API:

- Expose the model as a web service using a RESTful API, allowing clients to send data for prediction and receive the results over HTTP.

- **Pros:**
- Easy integration with various clients and platforms.
- Provides a centralized location for the model, making updates and versioning easier.
- Can be hosted on cloud providers for better scalability.
- **Cons:**
- Additional latency due to network overhead.
- Requires setting up and maintaining the API server infrastructure.

Model serving platforms:

- Use specialized platforms designed to host and serve machine learning models, such as TensorFlow Serving, TorchServe, or cloud-based solutions like Google AI Platform, Amazon SageMaker, or Microsoft Azure ML.
- **Pros:**
- Optimized for low latency and high throughput.
- Supports model versioning and easy updates.
- Scalability and resource management features.

- Cloud-based solutions offer managed infrastructure and additional tools for monitoring and maintenance.
- **Cons:**
- May require learning platform-specific tools and configuration.
- Potentially higher costs for cloud-based solutions.
- Additional latency due to network overhead.

Serverless deployment:

- Deploy the model as a serverless function using cloud platforms like AWS Lambda, Google Cloud Functions, or Azure Functions.
- **Pros:**
- Automatic scaling based on demand.
- Pay-per-use pricing model, which can be cost-effective.
- Simplified infrastructure management.
- **Cons:**
- Limited by the serverless platform's resource constraints (e.g., memory, execution time).

- Additional latency due to network overhead and potential cold starts.
- May require additional configuration and setup for optimal performance.

Embedded deployment:

- Deploy the model directly on edge devices, such as IoT devices or smartphones, by converting the model to a format suitable for the target device.
- **Pros:**
- Low latency, as the model runs directly on the device.
- Data privacy, as no data is sent to external servers.
- **Cons:**
- Limited resources on edge devices may require model optimization (e.g., pruning, quantization) to fit and run efficiently.
- Difficulty in updating the model or handling multiple versions.

Monitoring:

Monitor the model's performance in production and update it if needed.

Image by the Author-Adobe Firefly

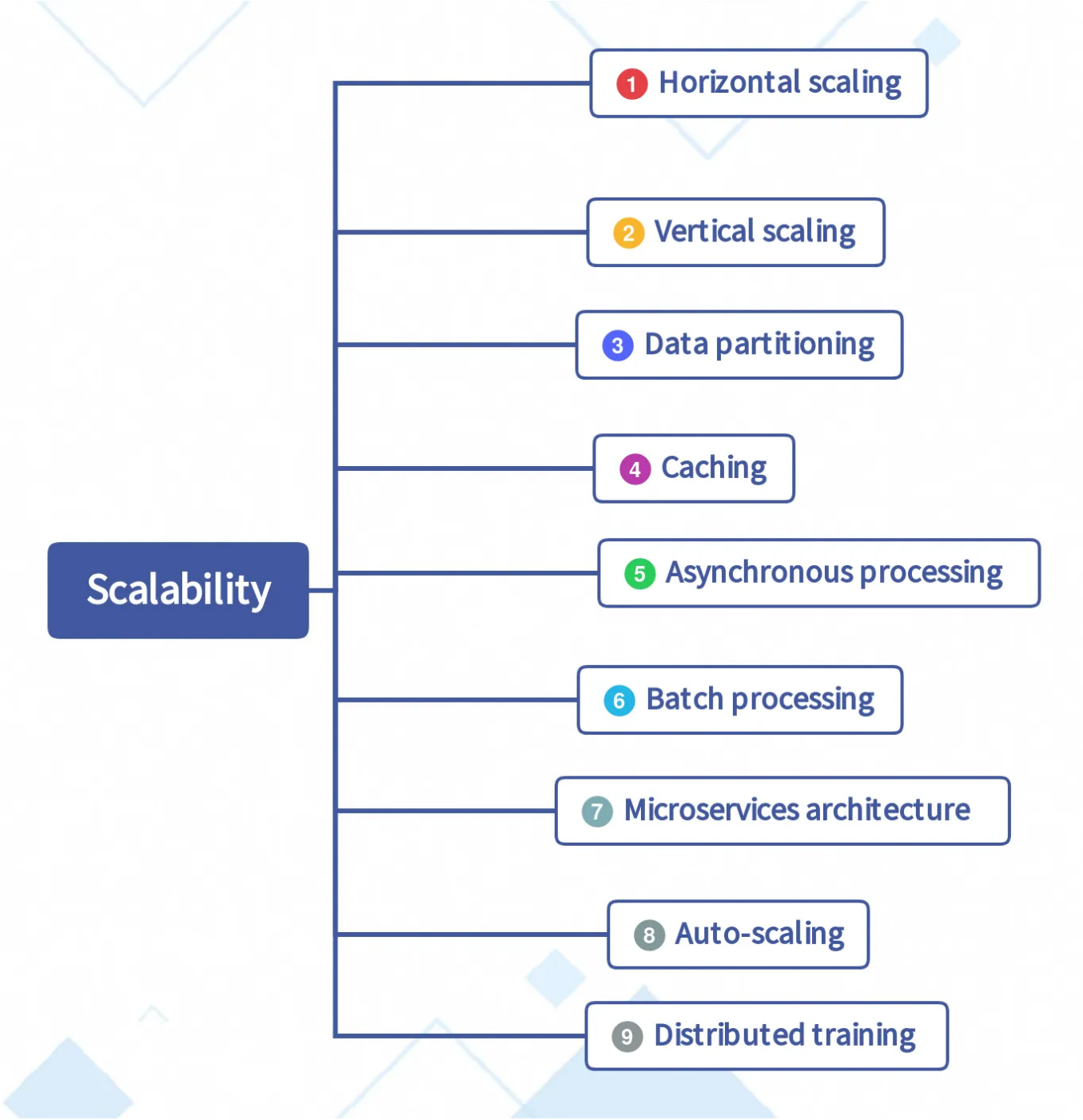


Image by the Author

Horizontal scaling:

- Add more machines or nodes to handle increased data and user loads. This is also called scale-out.
- Distribute data and tasks among multiple instances.
- Use load balancers to distribute requests evenly among instances.

Vertical scaling:

- Increase the resources (CPU, RAM, storage) of existing machines or nodes to handle increased loads. This is also called scale-up.
- Typically limited by the maximum capacity of a single machine.
- Easier to implement but may lead to higher costs and downtime during upgrades.

Data partitioning:

- Divide data into smaller chunks and distribute them across multiple machines or nodes to improve parallelism and load balancing.
- Techniques include sharding, consistent hashing, and range partitioning.

Caching:

- Store frequently accessed data or precomputed results in memory for faster retrieval and reduced load on the backend systems.
- Use caching strategies like least recently used (LRU) or time-to-live (TTL) to manage cache efficiently.

Asynchronous processing:

Perform time-consuming tasks asynchronously using message queues or task queues to avoid blocking and improve system responsiveness.

- Examples include RabbitMQ, Apache Kafka, and Amazon SQS.

Batch processing:

Process large volumes of data in batches to improve efficiency and resource utilization.

- Use frameworks like Apache Spark or Hadoop MapReduce for distributed batch processing.

Microservices architecture:

- Break the system into smaller, independent services that can be scaled and deployed independently.
- Facilitates better resource management and easier scaling.

Auto-scaling:

- Automatically adjust the number of instances or resources based on the current load, ensuring optimal performance and cost efficiency.
- Cloud-based solutions like AWS Auto Scaling, Azure Autoscale, and Google Cloud Autoscaler provide this capability.

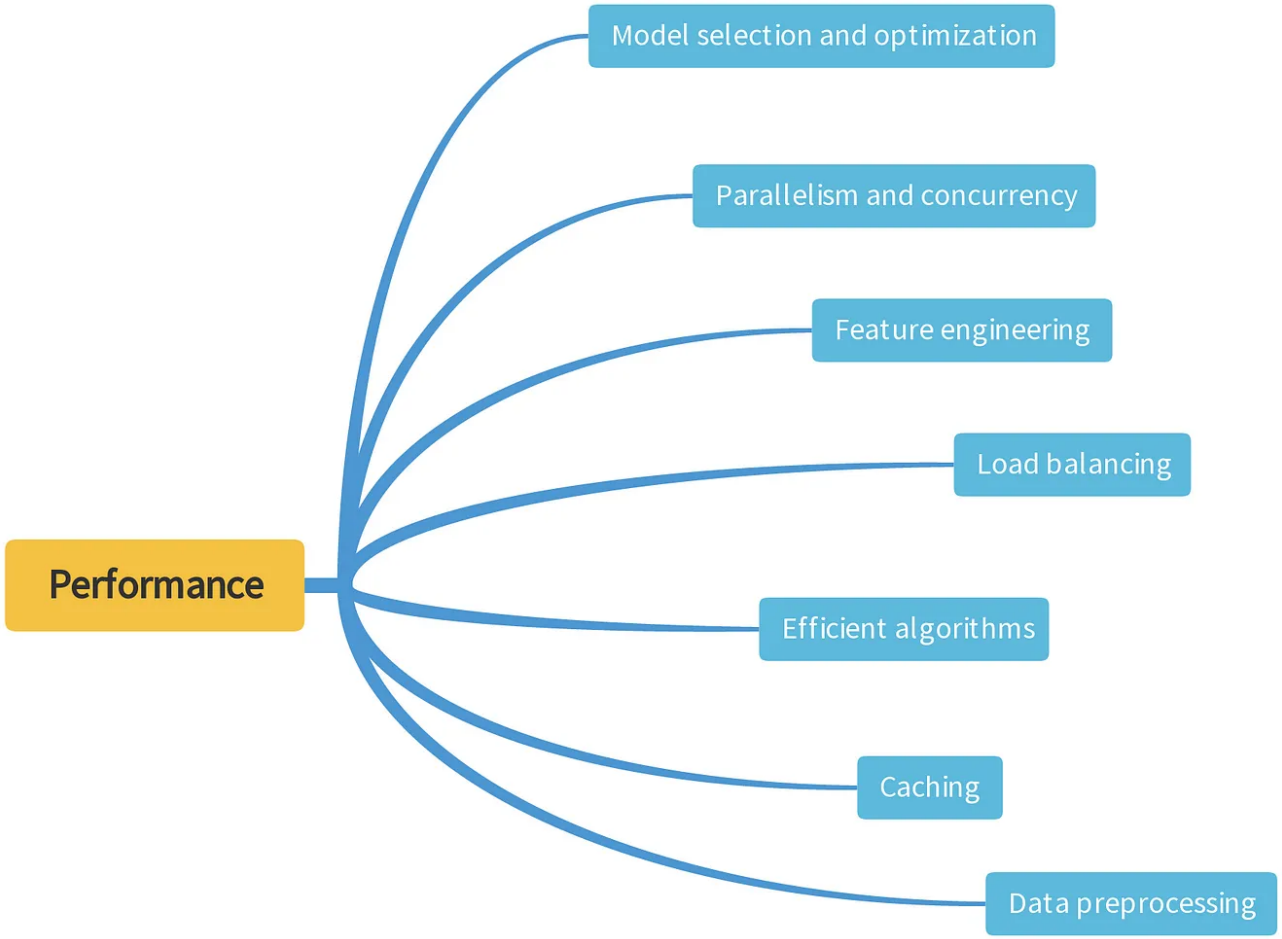
Distributed training:

- Leverage multiple machines or nodes for parallel model training to handle larger datasets and reduce training times.
- Use frameworks like Horovod, TensorFlow, and PyTorch for distributed training.

Monitoring and observability:

- Continuously monitor system performance, resource utilization, and user metrics to identify bottlenecks, plan capacity upgrades, and ensure optimal performance.
- Use tools like Prometheus, Grafana, and cloud-based monitoring solutions.

Image by the Author-Adobe Firefly



Model selection and optimization:

- Choose models with lower complexity to reduce training and inference times.

- Use model compression techniques like pruning, quantization, and knowledge distillation to reduce model size and improve inference speed.
- Utilize hardware acceleration (GPU, TPU) for faster training and inference.

Feature engineering:

- Reduce the number of features to minimize computation and storage overhead.
- Perform feature scaling and normalization to improve convergence speed during training.
- Use dimensionality reduction techniques like PCA or t-SNE to reduce feature space size without losing significant information.

Data preprocessing:

- Optimize data loading and preprocessing steps to minimize delays.
- Use parallel processing techniques to speed up preprocessing tasks.
- Cache preprocessed data or intermediate results to avoid redundant computations.

Efficient algorithms:

- Choose algorithms with lower time complexity for training and inference.
- Utilize approximate algorithms or online learning methods for real-time processing.

Parallelism and concurrency:

- Use multi-threading, multi-processing, or asynchronous processing to perform tasks concurrently.
- Implement distributed processing for handling large datasets or complex models.
- Use parallel algorithms, vectorization, or GPU acceleration to optimize computations.

Caching:

- Cache frequently accessed data, intermediate results, or model predictions in memory to reduce latency.

- Implement cache eviction policies like LRU or TTL to manage cache efficiently.

Load balancing:

- Distribute workload evenly across multiple instances or nodes to ensure optimal resource utilization and reduce processing time.
- Use techniques like round-robin, least connections, or consistent hashing for load balancing.

Security:

Protect the system against unauthorized access and data breaches.

Image by the Author

Privacy:

Ensure data privacy and compliance with regulations like GDPR.

Bias and fairness:

Assess and mitigate potential biases in the model.

Data exploration:

Perform exploratory data analysis (EDA) to understand data distributions.

Image by the Author-Adobe Firefly

Correlation analysis:

- Correlation measures the linear relationship between two variables. In the context of feature selection, it helps identify features that are strongly correlated with the target variable, as well as those that are strongly correlated with each other (multicollinearity).

- Pearson's correlation coefficient (for continuous variables) and Spearman's rank correlation coefficient (for ordinal variables) are commonly used correlation measures.
- Example: In a house price prediction dataset, you might find that the living area size and the number of bedrooms are both strongly correlated with the house price. In this case, you could choose to keep one of these features and drop the other to reduce redundancy.

Mutual information:

- Mutual information measures the amount of information that one variable provides about another. In the context of feature selection, it helps identify features that provide the most information about the target variable.
- It's particularly useful when dealing with nonlinear relationships between variables.
- Example: In a customer churn prediction dataset, you might find that the mutual information between the customer's total bill amount and the churn status is high. This indicates that the total bill amount is an important feature for predicting customer churn.

Recursive feature elimination (RFE):

- RFE is a feature selection technique that works by fitting the model multiple times and recursively removing the least important features.
- RFE starts by fitting the model using all features, ranking them based on their importance (e.g., using feature importances from a tree-based model or coefficients from a linear model), and eliminating the least important feature(s).
- The process is repeated, fitting the model with the remaining features and eliminating the least important ones until the desired number of features is reached.
- Example: In a credit risk assessment dataset, RFE could be used to rank features like credit score, income, and loan amount. After several iterations, RFE may determine that credit score and income are the most important features for predicting credit risk and eliminate other less important features.

Dimensionality reduction:

- **Principal Component Analysis (PCA):** A linear transformation technique that projects data onto a lower-dimensional space while maximizing the variance. It is useful for reducing noise, speeding up training, and visualizing high-dimensional data. Example: Reduce the

number of features in a facial recognition dataset while preserving the key characteristics that help distinguish different faces.

- **t-Distributed Stochastic Neighbor Embedding (t-SNE):** A non-linear dimensionality reduction technique that maps high-dimensional data to a lower-dimensional space while preserving local structure. It is particularly useful for visualizing complex data. Example: Visualize clusters of similar documents in a text dataset.
- **Uniform Manifold Approximation and Projection (UMAP):** A non-linear dimensionality reduction technique that balances preserving local and global structure. It is useful for visualization and general dimensionality reduction tasks. Example: Reduce the dimensionality of a gene expression dataset for visualization and clustering analysis.

Handling imbalanced data:

Image by the Author

- **Resampling:** Balance the class distribution by oversampling the minority class, undersampling the majority class, or both. Techniques include random oversampling, random undersampling, and SMOTE. Example: Balance a fraud detection dataset by oversampling fraudulent transactions.
- **Cost-sensitive learning:** Assign different misclassification costs to different classes during model training, making the model more sensitive to the minority class. Example: In a churn prediction model, assign a higher cost to misclassifying churned customers.
- **Ensemble methods:** Combine multiple models to improve prediction performance, such as using balanced random forests or EasyEnsemble. Example: Use balanced random forests for predicting rare diseases in a medical dataset.

Data augmentation:

- Data augmentation generates new training samples by applying transformations to existing data, increasing the size and diversity of the dataset.
- Example (image data): Apply rotations, flips, translations, and zooms to images to create new training samples.

- Example (text data): Use synonyms, paraphrasing, or back-translation to generate new text samples.

Model validation:

- **Cross-validation:** Partition the dataset into k subsets, train the model on k-1 subsets, and evaluate on the remaining subset. Repeat this process k times and average the performance metrics. Example: Use 5-fold cross-validation to estimate the performance of a classification model.
- **Holdout:** Split the dataset into a training set and a validation set. Train the model on the training set and evaluate its performance on the validation set. Example: Use a 70–30 train-validation split for a regression model.
- **Bootstrapping:** Sample with replacement from the dataset to create new datasets, train the model on these datasets, and evaluate its performance. Average the performance metrics over multiple iterations. Example: Use bootstrapping to estimate the performance of a time series forecasting model.

Loss functions:

Image by the Author

- Loss functions measure the difference between the model's predictions and the actual target values.
- **Regression:** Mean squared error (MSE), mean absolute error (MAE), and Huber loss are commonly used loss functions for regression tasks.
- **Classification:** Cross-entropy loss (log loss) for binary or multi-class classification, and hinge loss for support vector machines.
- **Ranking:** RankNet, LambdaMART, and RankBoost loss functions are used for ranking tasks in information retrieval and recommendation systems.

Image by the Author

- **L1 regularization (Lasso):** Adds the sum of the absolute values of the model parameters to the loss function. This can lead to sparse models

with some parameters becoming exactly zero, effectively performing feature selection. Example: Apply L1 regularization to a linear regression model to reduce the number of features and improve interpretability.

- **L2 regularization (Ridge):** Adds the sum of the squared values of the model parameters to the loss function. This helps to reduce the magnitude of the parameters, making the model less sensitive to small changes in the input features. Example: Apply L2 regularization to a logistic regression model to prevent overfitting when there are collinear features.
- **Dropout:** A regularization technique for neural networks that randomly sets a fraction of the neurons to zero during each training iteration. This helps to prevent overfitting and improve model generalization. Example: Apply dropout to a convolutional neural network (CNN) for image classification to reduce overfitting and improve generalization on new images.

Image by the Author

- **Batch Gradient Descent:** Update the parameters using the average gradient computed over the entire dataset. This can be computationally expensive for large datasets.
- **Stochastic Gradient Descent (SGD):** Update the parameters using the gradient computed from a single training example. This introduces noise, which can help escape local minima, but can also make convergence slower.
- **Mini-batch Gradient Descent:** Update the parameters using the average gradient computed from a small subset (batch) of the dataset. This balances the computational efficiency of batch gradient descent and the noise of SGD.
- **Linear Regression:** Gradient descent can be used to find the optimal weights (slope and intercept) for a linear regression model that minimizes the mean squared error (MSE) between the model's predictions and the actual target values.
- **Neural Networks:** Gradient descent, combined with backpropagation, is used to update the weights and biases of neural networks to minimize the loss function (e.g., cross-entropy loss for classification tasks).

Learning rate:

Choose and tune the learning rate for the optimization process.

Batch vs. online learning:

Understand the trade-offs between batch and online learning.

Transfer learning:

Some of the Transfer learning techniques:

1. **Pre-trained Models:** Use existing pre-trained models (e.g., VGG, ResNet, BERT) as a starting point for training on a specific task.
2. **Fine-tuning:** Adjust the weights of a pre-trained model by continuing training with a smaller learning rate on the target task dataset.
3. **Feature Extraction:** Use the pre-trained model to extract features from the target task data and train a separate classifier on top of those features.
4. **Layer Freezing:** Freeze the initial layers of a pre-trained model, allowing only the latter layers to be fine-tuned during training on the target task dataset.

Multi-task learning:

Train models to learn multiple tasks simultaneously.

Active learning:

Select informative samples to improve model performance with less data.

Reinforcement learning:

Understand the basics of RL, including states, actions, and rewards.

Supervised learning:

Understand classification, regression, and common algorithms.

Image by the Author

Unsupervised learning:

Understand clustering, dimensionality reduction, and common algorithms.

Image by the Author

Semi-supervised learning:

Utilize both labeled and unlabeled data.

Deep learning:

Understand neural networks, architectures, and common applications.

Please check this one.

Learning Machine Learning | Cloud AI | Google Cloud

Story, Design, and Layout by Lucy Bellwood, Dylan Meconis, Scott McCloud Line Art by Leila del Duca Color by Jenn...

cloud.google.com

Convolutional neural networks (CNNs):

Understand the role of convolutional layers and pooling layers.

Recurrent neural networks (RNNs):

Understand the role of LSTM and GRU cells.

Transformers:

Check this one.

Introduction - Hugging Face Course

This course will teach you about natural language processing (NLP) using libraries from the Hugging Face ecosystem - 🤗 ...

huggingface.co

What Are Large Language Models Used For and Why Are They Important? | NVIDIA Blog

AI applications are summarizing articles, writing stories and engaging in long conversations - and large language...

blogs.nvidia.com

Generative models:

Understand GANs, VAEs, and their applications.

Natural language processing (NLP):

Understand tokenization, embeddings, and common tasks.

Computer vision:

Understand image processing, object detection, and segmentation.

Time series analysis:

Understand techniques like ARIMA, state space models, and LSTM for handling time series data.

Anomaly detection:

Understand methods like clustering, autoencoders, and isolation forests.

Recommender systems:

Understand collaborative filtering, content-based filtering, and hybrid methods.

Image by the Author

Graph-based models:

Understand graph neural networks, node embeddings, and graph convolutions.

Feature extraction:

Utilize techniques like bag-of-words, TF-IDF, and Word2Vec for text data.

Model interpretability:

Use methods like LIME, SHAP, and permutation importance.

Image by the Author

Here's an overview of the A/B testing process, along with examples:

Define the objective: Clearly state the goal of the A/B test, such as improving click-through rates, increasing user engagement, or enhancing model

accuracy. This objective should be measurable and specific.

Example: An e-commerce website wants to improve its product recommendation algorithm. The objective is to increase the click-through rate of recommended products.

Develop variations: Create different versions of the model or feature to be tested. These variations can include different algorithms, hyperparameters, or feature sets.

Example: The data science team develops two models for product recommendation: Model A, based on collaborative filtering, and Model B, using a content-based approach.

Randomize and split the audience: Divide the target audience or dataset into random, non-overlapping groups. Each group is exposed to one of the variations.

Example: The e-commerce website randomly assigns half of its users to receive recommendations from Model A and the other half to receive recommendations from Model B.

Monitor and collect data: Track the performance metrics and gather data for each variation during the testing period.

Example: The website collects click-through data for the recommended products from both Model A and Model B over two weeks.

Analyze the results: Analyze the data to determine which variation performs better according to the defined objective. Use statistical tests, such as t-tests or chi-squared tests, to ensure that the results are statistically significant.

Example: After analyzing the data, the team finds that Model B has a significantly higher click-through rate than Model A.

Choose the winner and implement: If one variation outperforms the others, choose it as the winner and implement it in production. If there is no clear winner, consider running additional tests or refining the variations.

Example: Based on the test results, the e-commerce website decides to deploy Model B for product recommendations.

Check out this Youtube playlist.

Multi-armed bandits:

Understand the exploration-exploitation trade-off in decision-making.

Image by the Author

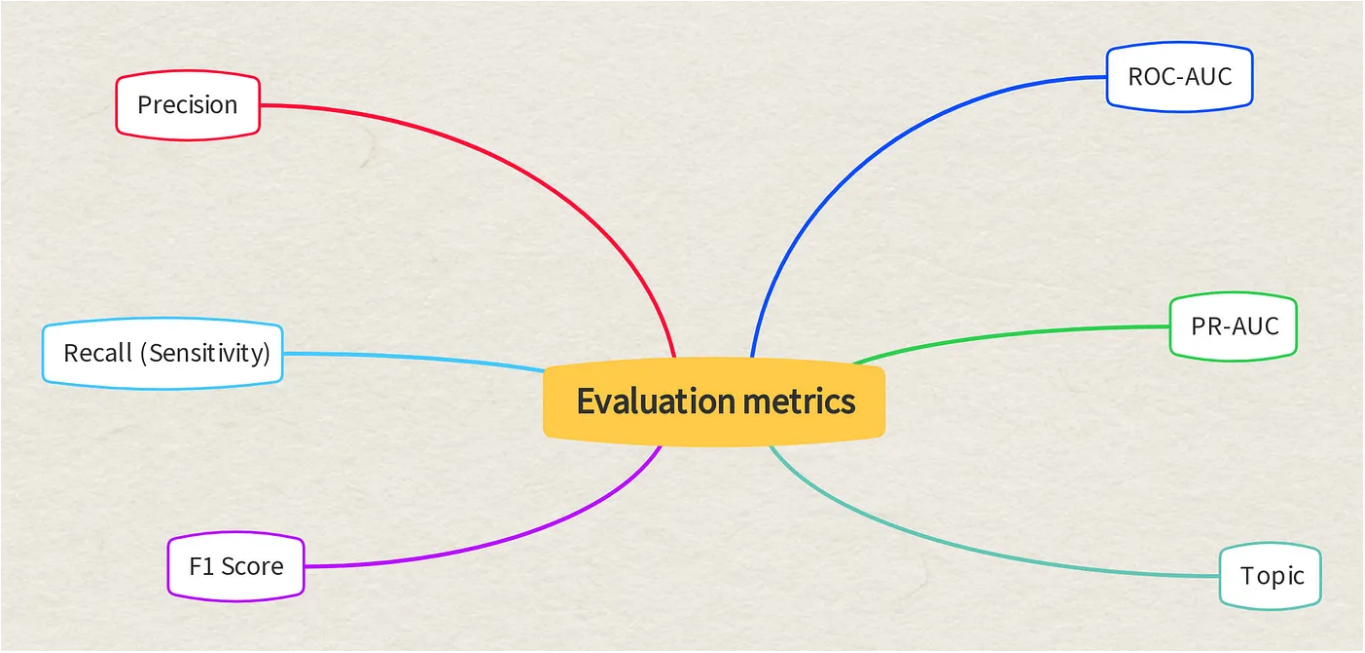


Image by the Author

Image by the Author

Precision: The proportion of true positive predictions among all positive predictions. It measures the model's ability to correctly identify positive instances.

Precision = True Positives / (True Positives + False Positives)

Recall (Sensitivity): The proportion of true positive predictions among all actual positive instances. It measures the model's ability to find all the positive instances.

Recall = True Positives / (True Positives + False Negatives)

F1 Score: The harmonic mean of precision and recall, providing a balanced measure of the model's performance, especially when dealing with imbalanced datasets.

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

ROC-AUC (Receiver Operating Characteristic — Area Under the Curve): A summary of the model's performance across various classification thresholds, measuring the trade-off between true positive rate (recall) and

false positive rate.AUC ranges from 0 to 1, where 1 indicates perfect classification, 0.5 indicates random guessing, and 0 means all predictions are wrong.

PR-AUC (Precision-Recall Area Under the Curve): A summary of the model's performance across various classification thresholds, focusing on the trade-off between precision and recall. It is particularly useful for imbalanced datasets, where the ROC-AUC may not be as informative. A high PR-AUC indicates both high precision and high recall, while a low PR-AUC indicates poor performance in either precision or recall or both.

Regression metrics:

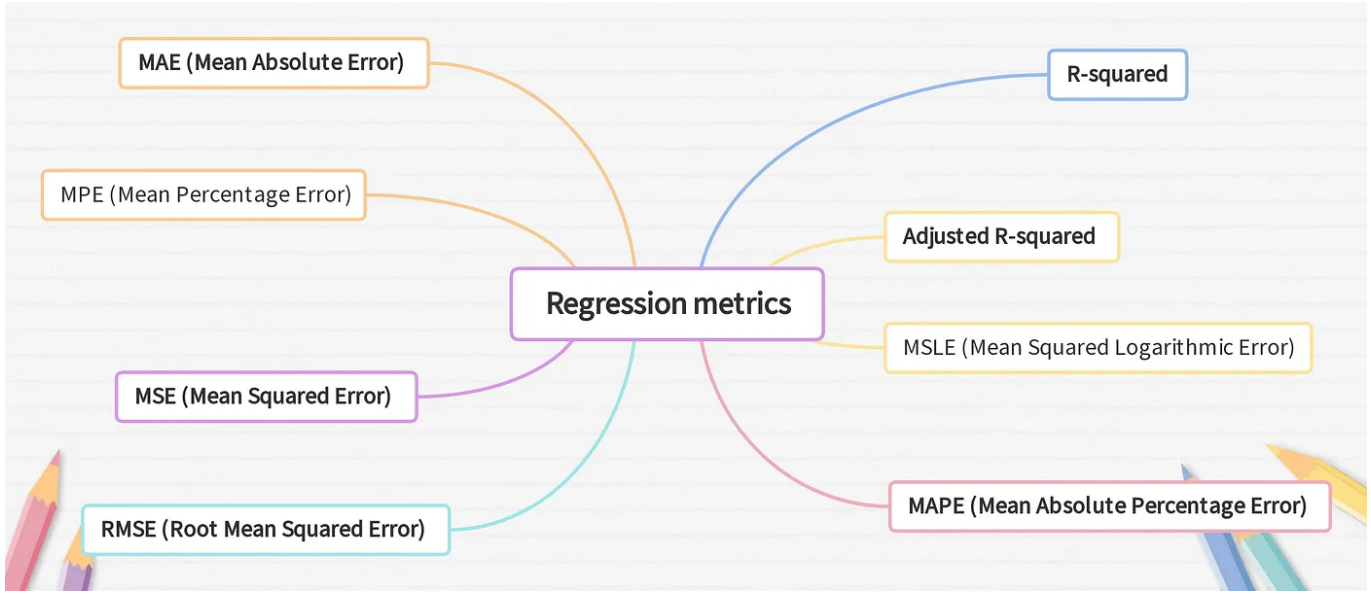


Image by the Author

1. **MAE (Mean Absolute Error):** Calculate the average of the absolute differences between predicted and actual values, indicating the magnitude of errors.
2. **MSE (Mean Squared Error):** Compute the average of squared differences between predicted and actual values, emphasizing larger errors.
3. **RMSE (Root Mean Squared Error):** Take the square root of MSE to bring the error metric back to the original value scale, useful for interpretation.
4. **R-squared:** Measure the proportion of variance in the dependent variable explained by the independent variables, indicating the model's goodness-of-fit.
5. **Adjusted R-squared:** Account for the number of features used in the model, providing a more balanced metric when comparing models with different numbers of features.
6. **MAPE (Mean Absolute Percentage Error)**: Calculate the average of absolute percentage differences between predicted and actual values, providing a relative error measure.

7. **MPE (Mean Percentage Error):** Compute the average of percentage differences between predicted and actual values, indicating the direction of the prediction bias.
8. **MSLE (Mean Squared Logarithmic Error):** Assess the average of squared logarithmic differences between predicted and actual values, emphasizing smaller errors.
9. **Huber Loss:** Combine the properties of MAE and MSE, providing a less sensitive metric to outliers than MSE.
10. **Quantile Loss:** Measure the difference between predicted and actual values for a specific quantile, useful for quantile regression tasks.

Clustering metrics:

Understand silhouette score, Davies-Bouldin index, and adjusted Rand index.

Hyperparameter search:

1. **Grid Search:** Perform an exhaustive search through a predefined set of hyperparameter values, evaluating each combination.
2. **Random Search:** Sample hyperparameter values randomly from specified distributions, covering a wider search space more efficiently.

3. **Bayesian Optimization:** Model the objective function using a surrogate model (e.g., Gaussian Process) and iteratively select hyperparameters based on an acquisition function.
4. **Sequential Model-Based Optimization:** Optimize hyperparameters using surrogate models and iterative selection, including Bayesian optimization and Tree-structured Parzen Estimators.

Image by the Author

Image by the Author

Min-max scaling (Normalization):

- Min-max scaling scales the features to a specific range, usually [0, 1], by subtracting the minimum value and dividing by the range (max-min) for each feature.
- Formula: X_scaled = (X — X_min) / (X_max — X_min)
- This method is sensitive to outliers, as they can cause the majority of the scaled values to lie within a small range.

Standardization (Z-score normalization):

- Standardization rescales the features such that they have a mean of 0 and a standard deviation of 1.
- Formula: X_scaled = (X — mean(X)) / std_dev(X)
- Unlike min-max scaling, standardization doesn't bound the features to a specific range, which can be a problem for some algorithms that expect input features to be within a certain range (e.g., neural networks).

- However, it is less sensitive to outliers compared to min-max scaling and often results in better performance for many machine learning algorithms.

Log transformation:

- Log transformation is a non-linear scaling technique used to reduce the impact of outliers and transform the feature distribution to be more Gaussian-like (i.e., reduce skewness).
- Formula: X_transformed = log(X + c), where c is a constant (e.g., 1) added to handle zero values in the dataset.
- It's particularly useful for features with a long-tailed distribution or when there's a multiplicative relationship between variables.
- Note that log transformation may not be appropriate for all datasets, and other transformations like square root or Box-Cox transformation can be considered based on the specific data distribution.

Categorical features:

One-hot encoding:

- One-hot encoding is a technique that creates binary features for each category in a categorical variable.
- For each unique category, a new binary feature is created, which takes the value of 1 if the original feature belongs to that category and 0 otherwise.
- This method results in a sparse representation, which can be memory-intensive for high cardinality categorical features (i.e., features with many unique categories).
- Example: Consider a “color” feature with three categories: red, green, and blue. One-hot encoding would create three binary features: is_red, is_green, and is_blue.

Ordinal encoding:

- Ordinal encoding is a technique that assigns an integer value to each unique category in a categorical feature based on its order.
- This method is appropriate for ordinal categorical features (i.e., features with a natural order, such as “small”, “medium”, “large”).

- It's not suitable for nominal categorical features (i.e., features without a natural order), as the assigned integer values may introduce an artificial order that can negatively affect the performance of the model.
- Example: Consider a "size" feature with three categories: small, medium, and large. Ordinal encoding would assign integer values such as 1, 2, and 3, respectively.

Target encoding (Mean encoding):

- Target encoding is a technique that replaces each category in a categorical feature with the mean of the target variable for that category.
- This method can be useful for high cardinality categorical features, as it results in a compact representation that captures information about the relationship between the feature and the target.
- However, target encoding can introduce leakage if not done correctly, as the encoding is based on the target variable. To avoid leakage, use K-fold cross-validation or split the data into training and validation sets before applying target encoding.

- Example: Consider a “city” feature in a dataset predicting house prices (target variable). For each city, calculate the average house price and replace the city name with the calculated average.

Missing data imputation:

Mean imputation:

- Mean imputation replaces missing values with the mean (average) of the available values in the same feature.
- This method is suitable for continuous variables and assumes that the missing data is missing at random.
- However, mean imputation can reduce the variance of the feature and may not be appropriate if the data is not missing at random or if the distribution is heavily skewed.
- Example: Consider a dataset with the “age” feature having some missing values. Calculate the mean age and replace the missing values with this mean.

Median imputation:

- Median imputation replaces missing values with the median (middle) value of the available values in the same feature.
- Like mean imputation, this method is suitable for continuous variables but is more robust to outliers and skewed distributions.
- Example: Consider a dataset with the “income” feature having some missing values. Calculate the median income and replace the missing values with this median.

Mode imputation:

- Mode imputation replaces missing values with the mode (most frequent) value of the available values in the same feature.
- This method is suitable for categorical variables and assumes that the missing data is missing at random.
- Example: Consider a dataset with the “color” feature having some missing values. Determine the most frequent color and replace the missing values with this mode.

Model-based imputations:

- Model-based imputation methods use a predictive model to estimate missing values based on the other available features in the dataset.
- Some common model-based imputation techniques include regression imputation (for continuous variables) and classification imputation (for categorical variables).
- More advanced techniques, like k-Nearest Neighbors (k-NN) imputation or using machine learning models like Random Forest or XGBoost, can also be employed for imputing missing values.
- Model-based imputations can provide better estimates of missing values, especially when there is a relationship between the feature with missing values and other features in the dataset.
- Example: Consider a dataset with the “height” feature having some missing values. Train a linear regression model using other features (e.g., age, gender) to predict the missing heights.

Image by the Author

Tukey fences:

- Tukey fences is a method that defines outliers based on the interquartile range (IQR) of the data.
- IQR is the range between the first quartile (25th percentile) and the third quartile (75th percentile) of the data.
- Tukey fences define outliers as data points that are below (Q1 — k * IQR) or above (Q3 + k * IQR), where k is a constant factor (typically 1.5 or 3).
- Example: Consider a dataset with the “price” feature. Calculate Q1, Q3, and IQR for this feature. If k = 1.5, any data point with a price below (Q1–1.5 * IQR) or above (Q3 + 1.5 * IQR) is considered an outlier.

Z-score:

- Z-score is a method that defines outliers based on the standard deviation of the data.
- The Z-score of a data point is the number of standard deviations it is away from the mean of the data.
- Outliers are defined as data points with Z-scores greater than a certain threshold (typically 2 or 3).

- Example: Consider a dataset with the “age” feature. Calculate the mean and standard deviation of this feature. If the threshold is 2, any data point with a Z-score greater than 2 or less than -2 is considered an outlier.

IQR method:

- The IQR method is similar to Tukey fences but uses a modified definition of the IQR to detect outliers.
- The IQR is calculated as the range between the first quartile (25th percentile) and the third quartile (75th percentile) of the data.
- Outliers are defined as data points that are below (Q1 — k * IQR) or above (Q3 + k * IQR), where k is a constant factor (typically 1.5 or 3).
- However, unlike Tukey fences, the IQR method uses a data-driven approach to determine the value of k, such as the median absolute deviation (MAD) or the robust Z-score.
- Example: Consider a dataset with the “income” feature. Calculate Q1, Q3, and IQR for this feature. Determine the value of k using the MAD or robust Z-score. If k = 2, any data point with an income below (Q1–2 * IQR) or above (Q3 + 2 * IQR) is considered an outlier.

Image by the Author

Image by the Author

1. **Apache Airflow:** An open-source platform used to programmatically author, schedule, and monitor workflows, allowing you to define and manage complex data pipelines.
2. **Apache NiFi:** A data integration and processing tool that provides a web-based interface for designing, controlling, and monitoring data flows.
3. **Luigi:** A Python-based workflow management system developed by Spotify, which helps in building complex pipelines of batch jobs.
4. **Kubeflow:** A Kubernetes-native platform for developing, orchestrating, deploying, and running scalable and portable ML workloads, particularly suitable for multi-cloud and hybrid cloud environments.
5. **TensorFlow Extended (TFX):** An end-to-end platform for deploying production ML pipelines, providing components for data validation, preprocessing, model training, and serving.
6. **MLflow:** An open-source platform for the complete machine learning lifecycle, including experimentation, reproducibility, deployment, and a central model registry.

7. **Apache Beam:** A unified programming model for both batch and streaming data processing, which allows you to build portable data pipelines using multiple languages and execution engines.
8. **Dask:** A parallel computing library for Python that allows you to build custom parallel data pipelines using familiar APIs like NumPy, pandas, and scikit-learn.

Dask Tutorial documentation

Edit description

tutorial.dask.org

TFX tutorials | TensorFlow

Google Cloud provides various products like BigQuery, Vertex AI to make your ML workflow cost-effective and scalable...

www.tensorflow.org

Tutorial

This tutorial showcases how you can use MLflow end-to-end to: Train a linear regression model Package the code that...

mlflow.org

Quick Start - Airflow Documentation

This quick start guide will help you bootstrap an Airflow standalone instance on your local machine. Note Successful...

airflow.apache.org

Samples and Tutorials

Samples and tutorials for Kubeflow Pipelines

www.kubeflow.org

Image by the Author

1. **Git:** The core distributed version control system, allows you to track changes, create branches, and collaborate with others.
2. **GitHub:** A web-based hosting service for Git repositories, providing a user-friendly interface, issue tracking, and collaboration features.

3. **GitLab:** A web-based DevOps platform that provides Git repository management, continuous integration/continuous deployment (CI/CD) pipelines, and project management features.
4. **Bitbucket:** A Git repository management solution by Atlassian, offering integration with other Atlassian tools like Jira, Bamboo, and Confluence.
5. **Azure DevOps:** A suite of DevOps services by Microsoft that includes Git repositories, CI/CD pipelines, and Agile planning tools, integrated with the Azure cloud platform.
6. **DVC (Data Version Control):** An open-source version control system specifically designed for managing data and machine learning projects, with Git integration for tracking code and data changes together.

Reproducibility:

Ensure experiment reproducibility through proper documentation and environment management.

Image by the Author

Image by the Author

Image by the Author

Use Docker for consistent and portable deployment environments.

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt ./

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Copy the rest of the application code into the container
COPY . .

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Build the Docker image: Run the `docker build` command to build a Docker image using the Dockerfile. This creates a snapshot of your application and its dependencies, which can be shared and deployed.

```
docker build -t your-image-name .
```

Run the Docker container: Start a Docker container from the built image using the `docker run` command. This launches an instance of your application in an isolated environment.

```
docker run -p 4000:80 your-image-name
```

Share the Docker image: Push the Docker image to a container registry like Docker Hub or Google Container Registry, allowing others to download and run your application.

```
docker tag your-image-name your-dockerhub-username/your-image-name
```

```
docker push your-dockerhub-username/your-image-name
```

Build your Python image

Now that you have a good overview of containers and the Docker platform, let's take a look at building your first...

docs.docker.com

MLOps-Dockers and Kubernetes Essentials for a Data Scientist

Introduction

medium.com

Check out my article about Docker and Kubernetes.

Continuous integration and deployment (CI/CD):

Here’s how to implement CI/CD for machine learning:

Version control:

- Use a version control system like Git to manage your code, data, and model artifacts.
- Organize your work using branches, tags, and pull requests to maintain a clean and efficient development process.

Automated testing:

- Write tests to validate the correctness and performance of your code and models. This may include unit tests, integration tests, and model evaluation tests.
- Use a testing framework like pytest for Python or Google Test for C++ to organize and execute your tests.

Continuous Integration (CI):

- Set up a CI server or service to automatically build and test your code whenever changes are pushed to the repository. Popular CI tools include Jenkins, CircleCI, Travis CI, and GitLab CI/CD.
- Configure the CI pipeline to run tests, validate code style, and check for potential issues using static analysis or linting tools.

- Automatically generate reports or notifications on test results and build status to keep the team informed.

Model training and validation:

- Automate the process of training and validating new models whenever new data or code changes are introduced.
- Use tools like MLflow or Kubeflow Pipelines to create automated training and validation workflows.
- Track and compare model performance across multiple experiments, and select the best-performing models for deployment.

Model packaging and containerization:

- Package your trained models and their dependencies into a container, using technologies like Docker, to create a consistent and portable deployment environment.
- Store and version your container images in a container registry, such as Docker Hub, Google Container Registry, or Amazon Elastic Container Registry.

Continuous Deployment (CD):

- Set up a CD pipeline to automatically deploy your models to production whenever a new version is available and has passed all tests.
- Use container orchestration platforms like Kubernetes or Docker Swarm, or cloud-based container services like Amazon ECS, Google Kubernetes Engine, or Azure Kubernetes Service to manage and scale your deployments.
- Implement rolling updates, blue-green deployments, or canary deployments to minimize downtime and reduce the risk of deploying faulty models.

Monitoring and observability:

- Monitor your deployed models for performance, resource utilization, and user metrics.
- Use tools like Prometheus, Grafana, or cloud-based monitoring solutions to collect and visualize metrics.
- Implement logging, tracing, and alerting to detect issues and facilitate debugging.

Feedback loop and model retraining:

- Collect feedback from the production environment, such as user interactions, model predictions, and ground truth labels, and use it to improve your models and training data.
- Automate the process of retraining and updating models based on new data and feedback to continuously improve model performance.

Check out my article on CI and CD article:

MLOps End-To-End Machine Learning Pipeline-CICD

The main objective of this project is to automate the whole machine learning app deployment process. To implement this...

medium.com

Image by the Author

Understand model serving options like TensorFlow Serving, TorchServe, and cloud-based solutions.

Image by the Author

TensorFlow Serving:

- Designed specifically for serving TensorFlow models.

- Supports gRPC and RESTful API for communication.
- Enables versioning and rolling updates of models.
- Highly scalable and optimized for high-performance serving.
- Can be deployed on-premises or in the cloud.

TorchServe:

- Developed for serving PyTorch models.
- Supports both HTTP and gRPC APIs for communication.
- Provides model versioning and management.
- Can be extended with custom handlers for specific use cases.
- Scalable and optimized for high-performance serving.
- Can be deployed on-premises or in the cloud.

Cloud-based solutions:

- **AWS SageMaker:** Supports TensorFlow, PyTorch, and many other frameworks. Provides auto-scaling, model versioning, and A/B testing.

- **Azure Machine Learning:** Supports various ML frameworks and offers auto-scaling, model versioning, and monitoring.
- **Google AI Platform:** Designed for serving TensorFlow models, but also supports other ML frameworks. Features auto-scaling, model versioning, and monitoring.

ONNX Runtime:

- Cross-platform runtime for serving models in the ONNX (Open Neural Network Exchange) format.
- Supports models from various ML frameworks, like TensorFlow, PyTorch, and Scikit-learn.
- Optimized for high-performance serving, including hardware acceleration.
- Can be deployed on-premises or in the cloud.

NVIDIA Triton Inference Server:

- Supports TensorFlow, PyTorch, ONNX, and other ML frameworks.
- Designed for GPU-accelerated serving, leveraging NVIDIA GPUs.

- Offers gRPC and HTTP/REST APIs for communication.
- Provides features like model versioning, multi-model serving, and dynamic batching.

Flask and FastAPI:

- Python web frameworks for building custom RESTful APIs.
- Lightweight and flexible, suitable for simple model serving.
- Easily integrates with various ML frameworks like TensorFlow and PyTorch.
- Can be containerized using Docker and deployed on-premises or in the cloud.

MLflow Model Server:

- Part of the MLflow platform, supporting various ML frameworks.
- Provides a simple and flexible way to serve models via REST API.
- Supports model versioning and deployment on-premises or in the cloud.

REST APIs:

Implementing a RESTful API for your model:

- **Choose a web framework**: Popular web frameworks for building RESTful APIs in Python include Flask and FastAPI. They help you define routes (endpoints) and handle HTTP requests easily.
- **Define endpoints**: For a machine learning model, you typically need at least one endpoint to make predictions. For example, you can create an endpoint like `/predict` that accepts POST requests with input data and returns predictions from your model.
- **Load your model**: Load your trained machine learning model into the server application so that it can be used for making predictions. Ensure that the model is loaded only once during the application's startup to avoid performance issues.
- **Process requests**: When a request is received at the `/predict` endpoint, extract the input data from the request, preprocess it as needed, and pass it to the model for prediction.
- **Return responses**: After getting the predictions, format them into a JSON object and return it as the HTTP response.

Consuming a RESTful API for model usage:

- **Make HTTP requests:** To use the model's RESTful API, you need to make HTTP requests to the defined endpoints. You can use libraries like `requests` in Python or built-in methods in other languages to make HTTP requests.
- **Format input data:** Before making a request to the `/predict` endpoint, format your input data as required by the API, usually as a JSON object.
- **Handle responses:** After making the request, parse the HTTP response to extract the JSON object containing the model's predictions. Use these predictions as needed in your application.

Tutorial - User Guide - Intro - FastAPI

This tutorial shows you how to use FastAPI with most of its features, step by step. Each section gradually builds on...

fastapi.tiangolo.com

Developing RESTful APIs with Python and Flask

TL;DR: Throughout this article, we will use Flask and Python to develop a RESTful API. We will create an endpoint that...

auth0.com

Tutorials - Streamlit Docs

Our tutorials include step-by-step examples of building different types of apps in Streamlit.

docs.streamlit.io

Image by the Author

Utilize tools like MLflow, TensorBoard, or cloud-based monitoring solutions.

Get started with TensorBoard | TensorFlow

In machine learning, to improve something you often need to be able to measure it. TensorBoard is a tool for providing...

www.tensorflow.org

MLflow Tracking

The MLflow Tracking component is an API and UI for logging parameters, code versions, metrics, and output files when...

mlflow.org

AutoML:

Understand the role of AutoML in automating the ML pipeline.

Image by the Author

Google Cloud AutoML - Train models without ML expertise

AutoML enables developers with limited machine learning expertise to train high-quality models specific to their...

cloud.google.com

AutoML - Automated Machine Learning - Amazon Web Services

Automatically create machine learning models with full visibility Automatically build, train, and tune the best ML...

aws.amazon.com

AutoML-train regression model (SDK v1) - Azure Machine Learning

APPLIES TO: Python SDK azureml v1 In this article, you learn how to train a regression model with the Azure Machine...

learn.microsoft.com

Model compression:

Use techniques like pruning, quantization, and knowledge distillation to reduce model size.

Edge computing:

Understand the challenges and benefits of deploying ML models on edge devices.

Image by the Author

Utilize tools like Horovod, TensorFlow, or PyTorch for parallel model training.

Data parallelism:

- Split the dataset into smaller chunks and distribute them across multiple devices or nodes.
- Each device trains a replica of the model on its subset of data.
- Model parameters are updated by aggregating gradients across all devices.

- Examples: TensorFlow's `tf.distribute.Strategy`, PyTorch's `torch.nn.parallel.DistributedDataParallel`.

Model parallelism:

- Distribute different parts of a model across multiple devices or nodes.
- Useful for large models that don't fit in the memory of a single device.
- Requires careful partitioning and communication between devices to ensure correct model execution.
- Examples: PyTorch's `torch.nn.parallel.DistributedDataParallel` with custom device assignment, Megatron-LM for large language models.

Hybrid parallelism:

- Combine data and model parallelism to leverage the advantages of both approaches.
- Distribute dataset chunks across multiple devices while partitioning the model to fit in memory.
- Examples: Mesh-TensorFlow for large-scale deep learning models.

Use distributed training frameworks and libraries:

- **Horovod:** A distributed training framework for TensorFlow, Keras, PyTorch, and Apache MXNet. It uses efficient communication methods like MPI or NCCL for gradient aggregation.
- **TensorFlow's `tf.distribute` module:** Provides various strategies like `MirroredStrategy` and `MultiWorkerMirroredStrategy` for distributed training.
- **PyTorch's `torch.distributed` module:** Offers distributed communication and data parallelism through classes like `DistributedDataParallel`.

GPU acceleration:

Understand the benefits of GPU acceleration for training and inference.

Faster computation: GPUs can perform parallel processing, which allows them to process large amounts of data simultaneously, significantly speeding up the training and inference of machine learning models, especially for deep learning.

High parallelism: GPUs have thousands of cores, enabling them to execute multiple threads concurrently. This feature is particularly beneficial for

tasks like matrix multiplication, which is common in deep learning algorithms.

Energy efficiency: GPUs often provide better performance per watt compared to CPUs, making them more energy-efficient for large-scale machine learning workloads.

Framework support: Popular machine learning frameworks like TensorFlow, PyTorch, and Keras have built-in support for GPU acceleration, making it easy to leverage the power of GPUs for training and inference.

Scalability: GPU-accelerated systems can be easily scaled by adding more GPUs to accommodate increasing data volumes and model complexity.

Example1: Training a deep learning model for image classification using a GPU can lead to a substantial reduction in training time compared to a CPU, allowing for quicker iterations and experimentation.

Example 2: In natural language processing tasks, such as training large transformer models like BERT or GPT, GPU acceleration is crucial to achieving reasonable training times, as these models require massive amounts of computation.

Check this out.

CUDA Python

"Anaconda is very supportive of NVIDIA's effort to provide a unified and comprehensive set of interfaces to the CUDA...

developer.nvidia.com

Model lifecycle management:

Manage the complete lifecycle of ML models from development to retirement.

Cost optimization:

Optimize resource usage and costs for cloud-based deployments.

Explainable AI (XAI):

Implement techniques and tools to provide insights into model predictions.

Check this google cloud documentation.

Why you need to explain machine learning models | Google Cloud Blog

Many companies today are actively using AI or have plans to incorporate it into their future strategies - 76% of...

cloud.google.com	

Human-in-the-loop:

Incorporate human feedback and expertise into the ML system.

Image by the Author

Stream processing:

- Stream processing is a technique for processing data in real time as it is generated or transmitted. It involves ingesting, processing, and analyzing data streams without waiting for the entire dataset to be available.
- Examples of stream processing frameworks include Apache Kafka, Apache Flink, and Apache Storm. These frameworks can handle high-throughput, low-latency data processing, making them suitable for real-time applications.

- Example: A social media platform wants to analyze user sentiment in real-time to identify and address negative experiences. The platform can use a stream processing framework like Apache Kafka to ingest user interactions, process them using natural language processing (NLP) techniques, and update sentiment scores in real time.

Real-time machine learning:

- Real-time machine learning involves training, updating, and using machine learning models on-the-fly as new data becomes available. This approach enables models to adapt quickly to changes in data patterns and maintain their performance.
- Online learning algorithms, such as stochastic gradient descent (SGD) and online learning with adaptive regularization, are suitable for real-time machine learning applications.
- Example: A financial institution wants to detect fraudulent transactions as they occur. The institution can use real-time machine learning to train and update a fraud detection model with the latest transaction data, ensuring that the model remains accurate and up-to-date.

Edge computing:

- Edge computing involves processing data near its source rather than sending it to a centralized data center or cloud. This approach can reduce latency, network congestion, and data transmission costs, making it suitable for real-time processing applications.
- Examples of edge computing platforms include AWS Greengrass, Azure IoT Edge, and Google Cloud IoT Edge. These platforms enable developers to deploy, manage, and run machine learning models and other processing tasks on edge devices like IoT sensors, cameras, and gateways.
- Example: A manufacturing company wants to monitor equipment performance and detect anomalies in real-time to prevent failures. The company can use edge computing to deploy machine learning models on IoT sensors and devices that monitor equipment parameters, enabling real-time anomaly detection and alerts.

Real-time databases:

- Real-time databases enable low-latency data storage, retrieval, and synchronization, making them suitable for applications that require real-time data processing and analysis.

- Examples of real-time databases include Firebase Realtime Database, Amazon DynamoDB, and Google Cloud Firestore. These databases can store and manage data in real time, enabling developers to build applications with real-time features like live updates, notifications, and collaboration.
- Example: A ride-hailing application wants to track and display the real-time location of drivers and riders. The application can use a real-time database like Firebase Realtime Database to store and synchronize location data, enabling real-time updates and notifications for all users.

Image by the Author

Image by the Author

Relational databases:

- Store structured data in a tabular format, with rows and columns.
- Use SQL (Structured Query Language) for data querying, retrieval, and manipulation.
- Provide strong consistency, ACID (Atomicity, Consistency, Isolation, Durability) properties, and transaction support.
- Examples: PostgreSQL, MySQL, Microsoft SQL Server, Oracle Database, MariaDB.

NoSQL databases:

- Designed to handle unstructured or semi-structured data, scaling horizontally, and providing high availability.
- Can be divided into several types, including key-value, document, column-family, and graph databases.
- Typically provide more flexible data models and faster write/read operations at the cost of weaker consistency guarantees.

- Examples: MongoDB (document), Cassandra (column-family), Redis (key-value), Neo4j (graph).

Time-series databases:

- Optimized for handling time-stamped data, such as sensor readings, log events, or financial data.
- Efficiently store, query, and analyze time-series data with built-in functions for aggregation, downsampling, and data retention.
- Examples: InfluxDB, TimescaleDB, OpenTSDB.

Search and analytics databases:

- Designed for indexing, searching, and analyzing large volumes of text or semi-structured data.
- Support full-text search, complex querying, and data aggregation.
- Examples: Elasticsearch, Apache Solr, Amazon CloudSearch.

Data warehouses:

- Designed for storing, analyzing, and reporting large volumes of structured data.
- Use a columnar storage format and advanced compression techniques for efficient storage and querying.
- Support SQL and provide powerful analytical capabilities, including OLAP (Online Analytical Processing) functions.
- Examples: Amazon Redshift, Google BigQuery, Snowflake, Apache Hive.

Data lakes:

- Store raw, unprocessed data in its native format, allowing for the storage of structured, semi-structured, and unstructured data.
- Provide a scalable and cost-effective storage solution, often based on object storage services.
- Enable data processing, transformation, and analysis using various tools and frameworks like Apache Spark, Apache Flink, or Dask.
- Examples: Amazon S3 (storage) with Amazon EMR (processing), Google Cloud Storage (storage) with Google Cloud Dataproc

(processing), Azure Data Lake Storage with Azure Databricks (processing).

Distributed file systems:

- Store large volumes of data across multiple machines, providing fault tolerance, scalability, and high throughput.
- Suitable for handling big data workloads, large-scale machine learning, and distributed data processing.
- Examples: Hadoop Distributed File System (HDFS), GlusterFS, Ceph.

Image by the Author

Image by the Author

Apache Kafka:

- Apache Kafka is an open-source distributed streaming platform that is designed for high-throughput, fault-tolerant, and scalable data streaming.
- It uses a publish-subscribe model, where data producers send messages to Kafka topics, and data consumers read messages from these topics.
- Kafka is designed to handle millions of events per second and is commonly used for log aggregation, stream processing, and event-driven architectures.
- Example: An e-commerce website wants to analyze customer behavior in real-time to provide personalized recommendations. The website can use Kafka to collect user events (e.g., clicks, views, and

purchases) and process them in real-time to generate recommendations for each user. Producers send user events to Kafka topics, and consumers process these events to update recommendation models and serve personalized content.

Amazon Kinesis:

- Amazon Kinesis is a managed service offered by AWS that makes it easy to collect, process, and analyze real-time streaming data.
- It has multiple components, such as Kinesis Data Streams, Kinesis Data Firehose, and Kinesis Data Analytics.
- Kinesis Data Streams enables real-time data streaming with low latency and high throughput.
- Kinesis Data Firehose simplifies the process of loading streaming data into data lakes, data stores, and analytics services.
- Kinesis Data Analytics allows you to process and analyze streaming data using SQL or Apache Flink.
- Example: A financial institution wants to detect fraudulent transactions in real-time. It can use Amazon Kinesis to stream transaction data and analyze it for signs of fraud. Kinesis Data

Streams receives transaction data from various sources, Kinesis Data Analytics processes and analyzes the data using a fraud detection model, and Kinesis Data Firehose stores the results in a data lake or database for further analysis or reporting.

Image by the Author

Introduction to streaming for data scientists

As machine learning moves towards real-time, streaming is becoming increasingly important for data scientists. We'll...

huyenchip.com

Data warehousing:

Understand concepts like data marts, star schema, and snowflake schema.

Image by the Author

Data lakes:

Understand the role of data lakes in handling large-scale, raw data.

GitHub - databricks-academy/ml-in-production-english: Machine Learning in Production

This repository contains the resources students need to follow along with the instructor teaching this course, in...

github.com

ETL processes:

Design and implement Extract, Transform, Load (ETL) processes for data integration and preparation.

Data governance:

Establish policies and processes for data quality, security, and compliance.

Metadata management:

Maintain metadata for data lineage, cataloging, and discoverability.

Image by the Author

Here are some key aspects of data lineage:

Data sources: Data lineage starts with identifying the origin of the data, which can include databases, external data sources, APIs, or files. Knowing where the data comes from helps organizations understand its quality, trustworthiness, and potential biases.

Data transformations: As data moves through a system, it undergoes various transformations, such as cleaning, normalization, aggregation, or enrichment. Data lineage tracks these transformations, providing insights into how the data has been modified and helping identify any issues or errors introduced during the process.

Data integration: Data from multiple sources may be combined, joined, or merged to create a unified dataset. Data lineage helps track the relationships between different datasets and ensures that the resulting integrated data is consistent and accurate.

Data usage: Understanding how and where data is used within an organization is crucial for data governance and compliance. Data lineage tracks data consumption across different applications, systems, and teams, providing insights into how data is utilized and enabling organizations to manage data access and security more effectively.

Data provenance: Data lineage helps establish the provenance of the data, providing a historical record of the data's origins, transformations, and usage. This information can be critical for auditing purposes, regulatory compliance, and ensuring the overall integrity of the data.

Data governance: Data lineage plays a vital role in data governance by enabling organizations to manage data quality, ensure data consistency, and support data privacy and security regulations. By understanding the flow of data and its transformations, organizations can implement better data governance policies and practices.

To implement data lineage, organizations can use various tools and techniques:

Metadata management: Metadata, or data about data, can be used to store and manage information about data lineage, such as data sources, transformations, and dependencies.

Data catalog: A data catalog is a centralized repository that stores metadata and information about data lineage, enabling users to discover, understand, and manage data assets.

Data lineage tools: Several tools and platforms, such as Apache Atlas, Collibra, and Informatica, are available to help organizations automate and visualize data lineage. These tools can extract and manage lineage information from various data sources, transformation processes, and applications.

Model drift:

Monitor and handle changes in model performance due to changing data distributions.

Concept drift:

Detect and adapt to changes in the underlying relationships in the data.

Check out my article on Data Drift.

Data visualization:

Utilize tools like Matplotlib, Seaborn, and Plotly for data visualization and analysis.

Check out my article on the Data Visualization tool.

MLOps-How to do EDA without writing a single line of python code.

Low-Code, No-Code, and Zero-Code EDA Tools

medium.com

Dashboarding:

Create interactive dashboards using tools like Tableau or Power BI.

Image by the Author

Here's a detailed explanation of MLOps:

Data versioning: Track and manage different versions of your data, allowing you to reproduce experiments and compare model performance across various datasets. Tools like DVC (Data Version Control) can help you version your data.

Model versioning: Similar to data versioning, model versioning tracks different versions of trained models, enabling easy comparison, rollback, and deployment. You can use tools like MLflow or ModelDB for model versioning.

Experiment tracking: Monitor and manage your experiments, including hyperparameters, model architectures, and performance metrics. This helps you keep track of progress, reproduce experiments, and identify the best performing models. Tools like TensorBoard, MLflow, and Weights & Biases can assist with experiment tracking.

Automated machine learning (AutoML): Automate the process of model selection, hyperparameter tuning, and model evaluation to save time and improve model performance. AutoML tools like H2O, AutoGluon, or Google Cloud AutoML can help automate these tasks.

Continuous integration (CI): Automate the process of integrating code changes into a shared repository, ensuring that the codebase is always up-to-

date and functioning correctly. Tools like Jenkins, GitLab CI/CD, and CircleCI can help with continuous integration.

Continuous delivery (CD): Automate the process of deploying models to a staging or production environment, ensuring that they are always up-to-date and ready for use. Continuous delivery can be implemented using tools like Jenkins, GitLab CI/CD, and Spinnaker.

Model deployment: Automate the deployment of trained models to various environments (e.g., cloud, on-premises, or edge devices) and manage their lifecycle. Tools like TensorFlow Serving, TorchServe, and Seldon Core can help with model deployment.

Model monitoring: Continuously monitor the performance of deployed models, detecting drifts, anomalies, or other issues that might impact their performance. Model monitoring can be implemented using tools like Grafana, Kibana, or custom-built monitoring solutions.

Model retraining: Automate the process of retraining models with new data, ensuring that they remain up-to-date and perform optimally. Retraining can be scheduled periodically or triggered by specific events (e.g., performance degradation or data drift).

Collaboration and communication: Facilitate communication and collaboration among team members, including data scientists, ML engineers, and other stakeholders. This can be achieved through the use of shared repositories, project management tools, and documentation.

Please check for more information.

Image by the Author

Some Open Source MLOps can be considered :

Experiment Tracking: MLflow is an open-source platform for managing the end-to-end machine learning lifecycle, including experiment tracking, model training, and deployment. It provides tools for tracking experiments, including parameters, code versions, and performance metrics.

Model Versioning: DVC (Data Version Control) is an open-source tool for versioning data and models. It allows users to track changes to data and models, collaborate with team members, and reproduce experiments.

Deployment Pipeline: Kubeflow is an open-source platform for deploying and managing machine learning workflows on Kubernetes. It includes tools for building and deploying machine learning models, as well as for monitoring and scaling them in production environments.

Model Monitoring: Prometheus is an open-source monitoring system that provides real-time metrics and alerts for machine learning models. It allows users to monitor model performance, detect anomalies, and alert users to potential issues.

Model Explainability: Lime is an open-source library for explaining the behavior of machine learning models. It allows users to generate human-

interpretable explanations for model predictions, as well as to evaluate the fairness and bias of models.

Machine Learning Tools Landscape v2 (+84 new tools)

Last June, I published the post What I learned from looking at 200 machine learning tools. The post got some attention...

huyenchip.com

Collaboration:

Work effectively with cross-functional teams, including data scientists, engineers, and domain experts.

Agile methodologies:

Agile methodologies are iterative, incremental approaches to software development and project management. They prioritize flexibility, collaboration, and rapid delivery of working software. Scrum and Kanban are two popular agile methodologies.

Domain knowledge:

Understand the specific domain and its impact on the ML system design.

Performance trade-offs:

Make informed decisions about the trade-offs between model complexity, accuracy, and latency.

Ethics:

Consider the ethical implications of the ML system and its potential impact on stakeholders.

Continuous learning:

Stay up-to-date with the latest advancements in machine learning, tools, and best practices.

No Free Lunch Theorem:

Understand that no single algorithm performs best for every problem, and selecting the right algorithm depends on the specific problem and data.

Machine Learning System Design Interview Books:

1. Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications 1st Edition by Chip Huyen (Author)
2. Machine Learning System Design Interview by Ali Aminian (Author), Alex Xu (Author)

3. Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps 1st Edition by Valliappa

Lakshmanan (Author), Sara Robinson (Author), Michael Munn (Author)

Some additional resources:

- Archives Page 1 | NVIDIA Blog

Artificial Intelligence Computing Leadership from NVIDIA

blogs.nvidia.com

Data / ML Blog | Uber Blog

Data / Machine Learning

www.uber.com

Medium

Edit description

netflixtechblog.medium.com

Machine Learning - tech-at-instacart

Read writing about Machine Learning in tech-at-instacart. Instacart Engineering.

tech.instacart.com

Machine Learning - DoorDash Engineering Blog

Data Machine Learning Five Common Data Quality Gotchas in Machine Learning and How to Detect Them Quickly Data...

doordash.engineering

AI & Machine Learning | Google Cloud Blog

Find all the latest news about Google Cloud and Machine Learning & AI with customer stories, product announcements...

cloud.google.com

ML Applications Archives

We use artificial intelligence and machine learning across Facebook to improve our products and services.

engineering.fb.com

Tags

The Stanford AI Lab (SAIL) Blog is a place for SAIL students, faculty, and researchers to share our work with the...

ai.stanford.edu

AWS Machine Learning Blog

This is a guest post by Carter Huffman, CTO and Co-founder at Modulate. Modulate is a Boston-based startup on a mission...

aws.amazon.com

Blog

Edit description

openai.com

AI - Machine Learning Blog

AI AML Artificial Intelligence automation Azure Databricks Azure Machine Learning Bot Framework Classical Machine...

techcommunity.microsoft.com

Conclusion:

I hope that this article proves beneficial for those preparing for machine learning system design interviews.

Image by the Author

Machine Learning Libraries:

- Scikit-learn: https://scikit-learn.org/stable/index.html
- TensorFlow: https://www.tensorflow.org/
- Keras: https://keras.io/
- PyTorch: https://pytorch.org/

Model Interpretability Libraries:

- LIME: https://github.com/marcotcr/lime
- SHAP: https://github.com/slundberg/shap
- ELI5: https://github.com/TeamHG-Memex/eli5

Hyperparameter Optimization Libraries:

- Scikit-Optimize: https://github.com/scikit-optimize/scikit-optimize
- Optuna: https://github.com/optuna/optuna
- Hyperopt: https://github.com/hyperopt/hyperopt
- Spearmint: https://github.com/JasperSnoek/spearmint

Pre-trained Models:

- TensorFlow Model Garden: https://github.com/tensorflow/models
- Hugging Face Transformers: https://github.com/huggingface/transformers
- PyTorch Image Models: https://github.com/rwightman/pytorch-image-models

Model Ensembling and Transfer Learning Libraries:

- XGBoost: https://github.com/dmlc/xgboost
- LightGBM: https://github.com/microsoft/LightGBM
- CatBoost: https://github.com/catboost/catboost
- Fast.ai: https://github.com/fastai/fastai

AutoML Libraries:

- Auto-sklearn: https://github.com/automl/auto-sklearn
- TPOT: https://github.com/EpistasisLab/tpot
- H2O AutoML: https://github.com/h2oai/h2o-3/tree/master/h2o-automl

Machine Learning Data Science Programming Python Deep Learning

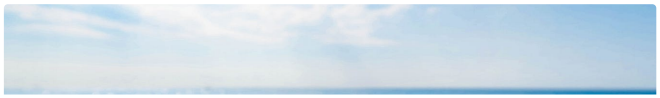
Written by Senthil E

Follow

1K Followers · Writer for Analytics Vidhya

ML/DS - Certified GCP Professional Machine Learning Engineer, Certified AWS Professional Machine learning Speciality,Certified GCP Professional Data Engineer .

More from Senthil E and Analytics Vidhya



Senthil E in Analytics Vidhya

Machine Learning System Design Interview Cheat Sheet-Part 2

Questions & Answers

Sumedh Datar in Analytics Vidhya

Application of Foreground and Background separation with Dee...

Foreground and background separation had always been a huge problem before the onse...

✦ · 22 min read · Apr 24

19

3 min read · Apr 20, 2021

10.1K

Kia Eisinga in Analytics Vidhya

How to create a Python library

Ever wanted to create a Python library, albeit for your team at work or for some open...

7 min read · Jan 27, 2020

1.8K 23

Senthil E in Analytics Vidhya

Machine Learning System Design Interview Cheat Sheet-LLM's Part 4

Large Language Models

✦ · 23 min read · Apr 24

12

See all from Senthil E

See all from Analytics Vidhya

Recommended from Medium

Dr. Roi Yehoshua in Towards Data Science

Mastering Logistic Regression

From theory to implementation in Python

✨ · 17 min read · 6 days ago

146

Gonzalo Recio

AI Trends of May 2023 You Need to Know

Summary of latest AI trends for May 2023

8 min read · 4 days ago

62 2

Lists

Senthil E in Analytics Vidhya

Machine Learning System Design Interview Cheat Sheet-LLM's Part 4

Large Language Models

✦ · 23 min read · Apr 24

12

Gabe Araujo, M.Sc. in Level Up Coding

🐼Introducing PandasAI: The Generative AI Python Library🐼

Pandas AI is an additional Python library that enhances Pandas, the widely-used data...

✦ · 9 min read · May 17

519 6