### **Machine Learning System Design Overview**

**1. What to Expect in a Machine Learning Interview:**

* **Interview Components:** Similar to traditional software engineering interviews, machine learning interviews focus on problem-solving (Leetcode style), system design, and knowledge of machine learning and machine learning system design.
* **Standard Development Cycle:** Includes data collection, problem formulation, model creation, implementation, and enhancement.
* **Objective:** Interviews aim to gauge the candidate’s competence across these areas. The course addresses the lack of common guidelines for end-to-end machine learning system design.

**2. Course Objectives:**

* **Top-Down View:** Teaches how to approach machine learning system design from a structural level, identifying challenges early on.

### **Steps to Approach Machine Learning System Design**

**1. Problem Statement:**

* **Clarification:** Understanding and defining the correct problem is crucial. Candidates should ask follow-up questions to clarify the problem statement and make explicit assumptions.
* **Example:** In designing a LinkedIn Feed Ranking system, questions about the output order and balance between feeds and ads are essential.

**2. Identify Metrics:**

* **Metrics Selection:** During development, use offline metrics to quickly test model performance. Common metrics include logloss and AUC for binary classification, or RMSE and MAPE for forecasting.

**3. Identify Requirements:**

* **Training Requirements:** Components include data collection, feature engineering, feature selection, and loss function. Address challenges such as class imbalance and model staleness.
* **Inference Requirements:** Ensure low latency (<100ms) and scalability for serving millions of users.

**4. Train and Evaluate Model:**

* **Components:** Feature engineering, feature selection, and models. Modern techniques are applied to each.
* **Examples:** Using ListingID as embedding features in Rental Search Ranking, handling latitude and longitude features in food delivery time estimation.

**5. Design High-Level System:**

* **System Components:** Identify minimal, viable components for a working system and explain their roles.
* **Example:** In Video Recommendation systems, separate components are needed for Video Candidate Generation and Ranking Model Services.

**6. Scale the Design:**

* **Address Bottlenecks:** Identify system bottlenecks and strategies to scale overloaded components, ensuring the system can serve millions of users.
* **Reliability:** Plan for component availability and system robustness.

### **Practical Applications**

The document also hints at practical applications of the principles discussed:

* **Ad Click Prediction**
* **Rental Search Ranking**
* **Estimate Food Delivery Time**

### **Additional Learning Resources**

Links to further resources and examples from real companies are provided for deeper understanding and practical insights into scaling machine learning systems.

### **Feature Selection and Feature Engineering Overview**

**1. One-Hot Encoding:**

* **Description:** Converts categorical variables into a one-hot numeric array, suitable for medium cardinality features.
* **Problems:** High computation and memory consumption, especially with high-dimensional feature vectors.
* **Best Practices:**
  + Group less important categories into an “Other” class.
  + Ensure the pipeline can handle unseen data in the test set.
  + Use tools like pandas.get\_dummies and sklearn.OneHotEncoder for consistency.
* **Applications:** Not suitable for large cardinality features; advanced techniques are used by companies like Instacart and DoorDash.

**2. Feature Hashing:**

* **Description:** Converts text data or categorical attributes with high cardinality into a feature vector of arbitrary dimensionality using a hash function.
* **Benefits:** Reduces dimensionality and memory usage by allowing multiple values to be encoded as the same value.
* **Example:** Converts text “The quick brown fox” into a feature vector with a specified dimensionality, illustrating the hashing trick.
* **Challenges:** Collisions can negatively affect model performance, especially if the hash size is too small.
* **Applications:** Widely used by companies like Booking, Facebook, Yahoo, Yandex, Avazu, and Criteo.

**3. Crossed Features:**

* **Description:** Combines two categorical variables into a single feature with higher cardinality, often used with hashing tricks to manage dimensions.
* **Example:** Using latitude and longitude in Uber pick-up data to predict demand more accurately.
* **Applications:** Used by LinkedIn for job recommendations and Airbnb for search ranking.

**4. Embedding:**

* **Description:** Transforms features into a new space to capture semantic meanings, representing similar features closely in the embedding vector space.
* **Benefits:** Unlike one-hot encoding and feature hashing, embeddings preserve semantic meanings.
* **Generation:** Defined through deep learning frameworks like TensorFlow, which can automatically learn embeddings.
* **Applications:**
  + Twitter for user IDs and recommendations.
  + DoorDash for personalizing store feeds (store2vec).
  + Instagram for content recommendations.
  + Embedding dimensionality is determined experimentally or by experience, and features are often pre-computed to reduce inference latency.

**5. Numeric Features:**

* **Normalization:**
  + **Description:** Adjusts numeric feature values to a range, often [-1, 1] or [0, 1].
  + **Formula:** vnorm=v−min(v)max(v)−min(v)v\_{norm} = \frac{v - min(v)}{max(v) - min(v)}vnorm​=max(v)−min(v)v−min(v)​
  + **Challenges:** Outliers can affect min and max values; clipping is used to address this.
* **Standardization:**
  + **Description:** Transforms features to have a mean of 0 and a standard deviation of 1.
  + **Formula:** vstd=v−μ(v)σ(v)v\_{std} = \frac{v - \mu(v)}{\sigma(v)}vstd​=σ(v)v−μ(v)​
  + **Log Transformation:** Used for power law distributions.
  + **Practical Considerations:** Normalization can cause issues due to outliers; solutions include using reasonable values for min and max.

### **Practical Applications**

The document also outlines practical use cases such as Ad Click Prediction, Rental Search Ranking, and Estimate Food Delivery Time, demonstrating the application of these techniques in real-world scenarios.

### **Additional Learning Resources**

Links are provided for further learning, including detailed techniques and practical examples from various tech companies.

### **Training Pipeline Overview**

**Objective:** A training pipeline should efficiently handle large volumes of data at low costs. Common solutions include storing data in column-oriented formats like Parquet or ORC, which enable high throughput for machine learning (ML) and analytics use cases. In TensorFlow ecosystems, tfrecord is also widely used.

### **Key Components and Practices**

**1. Data Partitioning:**

* **Efficiency:** Parquet and ORC files are often partitioned by time (e.g., by year and month) to avoid scanning entire datasets, significantly speeding up queries and reducing costs.
* **Example:** Partitioning training data in Parquet format can lead to queries being 30x faster, saving 99% of costs and reducing scanned data by 99%.

**2. Handling Imbalanced Class Distribution:** In scenarios like fraud detection, click prediction, or spam detection, class imbalance is common. Strategies to address this include:

* **Class Weights:** Adjusting the loss function to penalize more for the majority class (e.g., non-spam in a spam detection problem).
* **Naive Resampling:** Resampling the majority class at a certain rate to balance the training set, ensuring validation and test data remain intact.
* **Synthetic Resampling:** Using techniques like SMOTE (Synthetic Minority Oversampling Technique) to synthesize elements for the minority class by creating synthetic points between chosen points and their neighbors.

**3. Choosing the Right Loss Function:**

* **Binary Classification:** Cross-entropy is popular, with variations like Normalized Cross Entropy (logloss) used in click-through rate (CTR) prediction to reduce sensitivity to background conversion rates.
* **Forecasting:** Metrics like Mean Absolute Percentage Error (MAPE) and Symmetric Absolute Percentage Error (SMAPE) are common. Care is needed with skewed target values and asymmetric treatments of over- and under-forecasts.
* **Regression Problems:** Quantile Loss is used, for example, by DoorDash to forecast food delivery demand.

**4. Retraining Requirements:**

* **Non-Stationary Processes:** Data distributions often change over time, requiring models to be retrained to maintain performance.
* **AdTech and Recommendations:** Regular retraining is crucial to capture changes in user behavior and trending topics. Training pipelines need to be fast and scalable, balancing model complexity and training time.
* **Common Pattern:** Using schedulers to retrain models regularly, often multiple times per day, to keep models up-to-date.

### **Detailed Insights**

**Data Storage and Formats:**

* **Parquet and ORC:** Preferred for their efficiency in handling large datasets.
* **AWS Services:** RedShift and Athena commonly support these formats.

**Handling Imbalance:**

* **Example:** In a spam detection problem with 95% non-spam and 5% spam, class weights in the loss function adjust penalties to balance the impact of each class.

**Loss Functions:**

* **MAPE and SMAPE:** For forecasting, focus on accuracy while considering the distribution of target values.
* **Quantile Loss:** Helps in scenarios where predicting the quantiles of the target distribution is more meaningful than predicting the mean.

**Retraining and Scalability:**

* **Scheduler Use:** Automates regular retraining processes.
* **Balancing Complexity:** Ensures training pipelines are efficient while maintaining model accuracy.

### **Practical Applications**

**Ad Click Prediction, Rental Search Ranking, and Estimate Food Delivery Time:** These use cases illustrate the application of training pipelines in real-world scenarios, emphasizing the importance of handling large datasets, imbalanced classes, and the need for regular retraining.

### **Conclusion**

The document concludes by emphasizing the importance of designing robust and scalable training pipelines to handle large-scale machine learning tasks effectively. It provides practical strategies and insights for managing data, addressing class imbalances, selecting appropriate loss functions, and ensuring models remain relevant through regular retraining.

### **Inference Overview**

**Objective:** Inference involves using a trained machine learning model to make predictions on new data. The document outlines various strategies to scale inference processes to handle production-level workloads efficiently.

### **Key Techniques and Strategies**

**1. Imbalance Workload:**

* **Splitting Workloads:** A common pattern is to distribute inference workloads across multiple servers, similar to load balancers, sometimes referred to as an Aggregator Service.
* **Aggregator Service Workflow:**
  1. **Client Requests:** Upstream processes send requests to the Aggregator Service.
  2. **Workload Distribution:** If the workload is too high, the Aggregator Service splits it among workers in the Worker Pool. The workers can be selected based on:
     + Workload
     + Round Robin
     + Request parameters
  3. **Response Handling:** Workers process the requests and send responses back to the Aggregator Service, which then forwards the responses to the clients.

**2. Serving Logics and Multiple Models:**

* **Dynamic Logic Adjustment:** It is crucial to adjust serving logic dynamically in business-driven systems. For example, in Ad Prediction systems, different models might be used depending on the type of ad candidates.
* **Combining Models:** Multiple models can be used in combination during inference to improve prediction accuracy and handle different scenarios.

**3. Non-Stationary Problem:**

* **Data Distribution Shift:** In online settings, data distributions often change over time. It is essential to keep models updated to maintain performance.
* **Model Freshness:** Based on how quickly model performance degrades, determine the frequency of model updates or retraining.
* **Bayesian Logistic Regression:** This algorithm is commonly used to handle non-stationary problems by updating the model as new data arrives.

**4. Exploration vs. Exploitation: Thompson Sampling:**

* **Trade-off:** In scenarios like Ad Click prediction, a balance between exploring new ads and exploiting known high-performing ads is crucial. Too much exploration can reduce revenue due to fewer conversions.
* **Thompson Sampling:** This technique helps manage the exploration-exploitation trade-off by deciding actions based on the potential reward at each time step.

### **Practical Applications**

The document applies these inference techniques to several use cases:

* **Ad Click Prediction**
* **Rental Search Ranking**
* **Estimate Food Delivery Time**