Machine Learning for Search, Recommendations & Ads

Complete ML reference for Faire interview preparation

1. ML Problem Types in Search/Ads

A. Classification

Use Cases: Click prediction, fraud detection, query classification

Binary Classification:

- Output: $P(y = 1|x) \in [0, 1]$
- Loss: Binary cross-entropy (log loss)
- Models: Logistic Regression, XGBoost, Neural Nets

Multi-class:

- Intent classification (navigational, transactional, informational)
- Category prediction (200+ classes)
- Loss: Categorical cross-entropy

B. Ranking

Use Cases: Search results, recommendations, ad placement

Goal: Order items by relevance/value

- Input: Query + list of items
- Output: Ordered list
- Key: Relative order matters more than absolute scores

C. Regression

Use Cases: CTR prediction, price optimization, demand forecasting

Characteristics:

- Output: Continuous value
- Loss: MSE, MAE, Huber
- Challenge: Long-tail distribution

2. Evaluation Metrics by Task

A. Classification Metrics

Binary Classification:

1. AUC-ROC (Most common)

- Range: [0.5, 1.0] (0.5 = random)
- Measures: Ranking quality
- Good for: Imbalanced datasets
- Production: 0.7-0.8 (good), 0.8+ (excellent)

2. Log Loss (Cross-Entropy)

$$\text{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

- Lower is better
- Penalizes confident wrong predictions
- Use when: Calibrated probabilities matter

3. Precision, Recall, F1

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1} = 2 \cdot \frac{\text{Prec} \times \text{Rec}}{\text{Prec} + \text{Rec}}$$

- Use: When class balance matters
- Threshold-dependent (unlike AUC)

4. PR-AUC

- Better than ROC for highly imbalanced
- Example: CTR = 2% (98% negative)

B. Ranking Metrics

1. NDCG@K ✓ (Industry standard)

$$\text{DCG@K} = \sum_{i=1}^{K} \frac{2^{\text{rel}_i} - 1}{\log_2(i + 1)}$$

$$\text{NDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}}$$

- Range: [0, 1]
- Graded relevance (0, 1, 2, 3)
- Position-aware (top results matter more)
- Common: NDCG@10, NDCG@20

Production values:

- 0.60-0.70: Baseline
- 0.70-0.80: Good
- 0.80+: Excellent

2. MRR (Mean Reciprocal Rank)

$$\text{MRR} = \frac{1}{|Q|}\sum_{q=1}^{|Q|} \frac{1}{\text{rank}_q}$$

- Use: When first relevant result matters
- Example: Navigational queries (brand search)
- Range: [0, 1]

3. MAP (Mean Average Precision)

$$\text{AP} = \frac{1}{R}\sum_{k=1}^{N} P(k) \cdot \text{rel}(k)$$

- Use: When all relevant results matter
- Rare in production (NDCG preferred)

4. Precision@K, Recall@K

$$\text{Precision@K} = \frac{\text{\# relevant in top-K}}{K} \qquad \text{Recall@K} = \frac{\text{\# relevant in top-K}}{\text{total relevant}}$$

- Simple, interpretable
- Binary relevance only
- Use for quick evaluation

C. Online Metrics

Engagement Metrics:

- **CTR**: Clicks / Impressions
- **Conversion Rate**: Purchases / Clicks
- **Add-to-Cart Rate**: Carts / Clicks
- **Time on Page**: Engagement depth

Business Metrics:

- **GMV**: Gross Merchandise Value
- **Revenue per Search**: Total $ / queries
- **ARPU**: Average Revenue Per User

Quality Metrics:

- **Zero-Result Rate**: % queries with no results
- **Refinement Rate**: % users who refine query
- **Bounce Rate**: Single-page sessions

D. Regression Metrics

1. MSE / RMSE

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Heavily penalizes outliers
- Use: When large errors are critical

2. MAE

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- Robust to outliers
- More interpretable than MSE

3. MAPE

$$\text{MAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

- Percentage error
- Problem: Undefined when $y_i = 0$

3. Learning to Rank (LTR)

Core Problem: Given query q and documents $\{d_1, ..., d_n\}$, produce optimal ranking

A. Pointwise LTR

Approach: Predict relevance score for each doc independently

$$\text{score}_i = f(q, d_i)$$

Models:

- Linear Regression
- Neural Networks (single output)
- Treat as classification (relevant/not)

Pros:

- Simple, fast training
- Works with small data
- Easy to interpret

Cons:

- Ignores relative order
- Not optimized for ranking metrics
- Absolute scores may be misleading

When to use: Cold start, baseline, simple problems

B. Pairwise LTR ✓

Approach: Learn to compare pairs of documents

$$P(d_i \succ d_j | q) = \sigma(f(q, d_i) - f(q, d_j))$$

Key Algorithms:

1. RankNet

- Neural network
- Loss: Cross-entropy on pairs
- Gradient flows through pairs

2. LambdaRank

- Weights pair loss by ΔNDCG
- Focuses on top results

3. Pairwise SVM

- Maximize margin between relevant pairs
- Less common in production

Pros:

- Directly optimizes order
- Better than pointwise
- More data from pairs

Cons:

- Quadratic pairs ($O(n^2)$)
- Still not directly optimizing NDCG
- Slower training

When to use: Good data, need better ranking

C. Listwise LTR ✓✓

Approach: Optimize entire list directly

Key Algorithm: LambdaMART (Production standard)

- Gradient Boosted Decision Trees (GBDT)
- Directly optimizes NDCG
- Implementation: XGBoost, LightGBM

XGBoost Configuration:

```
params = {
    'objective': 'rank:ndcg',
    'eval_metric': 'ndcg@10',
    'eta': 0.1,
    'max_depth': 6,
    'subsample': 0.8
}
```

Other Listwise:

- **ListNet:** Permutation probability
- **ListMLE:** Maximum likelihood
- **AttentionRank:** Transformer-based

Pros:

- Best offline metrics
- Directly optimizes NDCG
- Industry standard

Cons:

- Complex training
- Needs more data
- Harder to debug

When to use: Production systems, large datasets

LTR Comparison Table

Type	Loss	Data Need	Use
Pointwise	Regression	Low	Baseline
Pairwise	RankNet	Medium	Good
Listwise	NDCG	High	**Prod**

4. Model Architectures

A. Traditional ML

1. Logistic Regression

- Fast, interpretable
- Linear decision boundaries
- Good baseline
- Use: Simple problems, small data

2. Gradient Boosted Trees ✓

- **XGBoost, LightGBM, CatBoost**
- Handles non-linear, interactions
- Feature importance built-in
- Robust to outliers
- Production: 80%+ of ranking systems

XGBoost Best Practices:

```
- Start: 100 trees, depth 6, lr 0.1
- Tune: Early stopping on validation
- Features: 50-200 is sweet spot
- Categorical: One-hot or target encode
```

3. Random Forest

- Parallel trees (vs sequential)
- Less overfitting than single tree
- Slower than XGBoost
- Use: Quick baseline, feature selection

4. Ensemble Methods ✓

Bagging (Bootstrap Aggregating):

- Train models on random subsets of data
- Average predictions (regression) or vote (classification)
- Reduces variance, prevents overfitting
- Example: Random Forest = Bagging + Decision Trees

Boosting:

- Sequential: each model corrects previous errors
- Reduces bias + variance
- Example: XGBoost, AdaBoost

Stacking:

- Train multiple models (level-0)
- Train meta-model on their predictions (level-1)
- Combines diverse models (XGB + NN + LR)
- Use: Kaggle competitions, when accuracy critical

B. Deep Learning

1. Deep Neural Networks (DNN)

- Multi-layer perceptron
- Good for: Complex patterns, embeddings
- Cons: Needs more data, harder to tune

Architecture:

```
Input → Dense(512) → ReLU → Dropout
      → Dense(256) → ReLU → Dropout
      → Dense(128) → ReLU
      → Output
```

Dropout: Regularization technique

- Randomly drop neurons (p=0.3-0.5) during training
- Forces network to learn robust features
- Reduces overfitting
- At inference: Use all neurons, scale by (1-p)

2. Wide & Deep ✓ (Google)

- **Wide:** Linear model (memorization)
- **Deep:** DNN (generalization)
- Best of both worlds
- Use: Recommendation, CTR prediction

3. DeepFM

- Factorization Machine + Deep
- Captures 2nd order interactions
- Use: CTR prediction, ads

4. Transformers / BERT

- State-of-art for text
- Latency: 50-200ms (too slow for ranking)
- Use: Query understanding, reranking top-K

4a. Attention Mechanism ✓

Core idea: Learn which parts of input to focus on

Self-Attention (Transformers):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Q = queries, K = keys, V = values (all from same input)
- d_k = dimension (prevents large dot products)
- Output: Weighted combination of values

Multi-Head Attention:

- Run 8-16 attention heads in parallel
- Each head learns different patterns
- Concatenate outputs, linear transform

Why powerful: Captures long-range dependencies, parallelizable (unlike RNNs)

C. Two-Tower Models ✓

Architecture:

```
Query → Query Tower → q_emb (128-dim)
Item  → Item Tower  → i_emb (128-dim)
Score = dot(q_emb, i_emb)
```

Benefits:

- Pre-compute item embeddings
- Fast retrieval via ANN (Faiss, ScaNN)
- Use: Candidate generation, 1M+ items

Companies using: YouTube, Pinterest, Facebook

D. Multi-Task Learning

Problem: Predict CTR, CVR, add-to-cart simultaneously

Architecture:

```
Shared layers (learn common patterns)
    ↓          ↓          ↓
Task head 1  Task 2   Task 3
(CTR)       (CVR)    (ATC)
```

Loss:

$$L = \alpha L_{\text{CTR}} + \beta L_{\text{CVR}} + \gamma L_{\text{ATC}}$$

Benefits:

- Share data across tasks
- Better generalization
- One model to serve

Challenge: Balancing task weights

5. Feature Engineering

A. Feature Categories

1. Query Features

- Query length (chars, words)
- Query type (brand, product, category)
- Historical CTR for query
- Query frequency (head vs tail)

2. Document Features

- Title, description (text)
- Category, brand (categorical)
- Price, rating, review count
- Age, popularity, inventory

3. Query-Doc Features ✓

- **BM25 score** (strongest signal)
- TF-IDF similarity
- Edit distance
- Exact match (title, brand)
- Embedding cosine similarity

4. User Features

- Demographics (age, location)
- Historical behavior (past clicks)
- Session context (device, time)
- User cohort (new, power user)

5. Context Features

- Time of day, day of week
- Season, holidays
- Device type (mobile, desktop)
- Location (country, city)

6. Text Matching Formulas ✓

BM25 (Best Match 25):

$$\text{BM25}(q,d) = \sum_{t \in q} IDF(t) \cdot \frac{TF(t,d) \cdot (k_1 + 1)}{TF(t,d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})}$$

- $TF(t,d)$ = term frequency in doc
- $IDF(t) = \log \frac{N - df(t) + 0.5}{df(t) + 0.5}$
- $k_1 = 1.5$ (term saturation), $b = 0.75$ (length norm)
- Most important text matching signal!

TF-IDF:

$$\text{TF-IDF}(t,d) = TF(t,d) \times \log \frac{N}{df(t)}$$

- TF = term frequency (count in doc)
- IDF = inverse document frequency
- N = total docs, $df(t)$ = docs containing t
- Simpler than BM25, but less effective

B. Feature Processing

Numerical Features:

- **Normalization:** $(x - \mu)/\sigma$
- **Log transform:** $\log(1 + x)$ for skewed
- **Binning:** Convert to categorical
- **Clipping:** Cap outliers at p99

Categorical Features:

- **One-hot:** For low cardinality ($<$50)
- **Target encoding:** For high cardinality
- **Embedding:** For deep learning
- **Frequency:** Count of occurrences

Text Features:

- TF-IDF vectors
- Word2Vec, GloVe embeddings
- BERT embeddings (expensive)
- N-grams (unigram, bigram)

Feature Crossing ✓ (Critical for CTR):

- **Explicit:** $f_1 \times f_2$ (price $\times$ category)
- **Polynomial:** $(user\_id, item\_id)$ pairs
- **Auto:** Deep learning learns interactions
- **Use:** Wide&Deep wide component, DeepFM

C. Feature Selection

Methods:

- **Correlation:** Remove redundant
- **Importance:** Use XGBoost feature_importance
- **Ablation:** Remove feature, measure impact
- **L1 Regularization:** Auto-select sparse

Rule of Thumb:

- Start: 50-100 features
- Production: 100-300 features
- More isn't always better (overfitting)

6. Training Pipeline

A. Data Collection

Labels for Ranking:

- **Explicit:** Human raters (expensive)
- **Implicit:** Clicks, purchases (biased)
- **Hybrid:** Combine both

Label Schema:

- 0: Irrelevant
- 1: Somewhat relevant
- 2: Relevant
- 3: Highly relevant

Click Bias: Position bias, presentation bias

- Solution: Inverse propensity weighting
- Interleaving experiments
- Randomization for exploration

B. Train/Val/Test Split

Time-based Split ✓:

```
Train: Days 1-60 (80%)
Val:   Days 61-75 (10%)
Test:  Days 76-90 (10%)
```

Why not random?

- Prevents data leakage
- Realistic evaluation
- Detects temporal drift

C. Training Frequency

Domain	Frequency	Why
Search	Weekly	Stable patterns
Ads	Daily	Fast changes
Recommendations	2-3 days	Medium drift

D. Negative Sampling ✓

Problem: Too many negatives (unchosen items)

Strategies:

- **Random:** Sample random items as negatives
- **Hard negatives:** High-scoring but not clicked
- **In-batch:** Use other positives as negatives

Sampling ratio:

```
1 positive : 5-10 negatives (typical)
1 positive : 100 negatives (extreme imbalance)
```

Hard Negative Mining:

- Select negatives model ranks highly (but wrong)
- Forces model to learn subtle differences
- Critical for two-tower models
- Update hard negatives every epoch

E. Model Evaluation

Offline:

- NDCG@10, MRR on test set
- Per-query analysis
- Error analysis (failure cases)

Online A/B Testing:

- **Metrics:** CTR, conversion, revenue
- **Duration:** 1-2 weeks (statistical power)
- **Sample:** 5-10% traffic
- **Statistical significance:** $p < 0.05$
- **Minimum detectable effect:** 2-5% improvement

A/B Test Calculations:

$$\text{Sample Size} = \frac{2(Z_{\alpha/2} + Z_{\beta})^2 \sigma^2}{\delta^2}$$

- δ = minimum detectable effect
- α = 0.05 (Type I error)
- β = 0.2 (Type II error, 80% power)

7. Serving & Deployment

A. Latency Budget

Total: 200ms p99

```
Retrieval:      50ms
Feature fetch:  30ms
Model inference: 80ms
Hydration:      30ms
Network:        10ms
```

Model Latency Optimization:

- **Quantization:** Float32 $\rightarrow$ Int8
- **Pruning:** Remove weak features/trees
- **Batch inference:** Process multiple queries
- **Two-stage:** Cheap first-pass, expensive rerank

B. Feature Store

Purpose: Centralized, low-latency feature access

Technologies:

- Redis (in-memory, <1ms)
- DynamoDB (AWS, 1-10ms)
- Feast (open-source)

Pattern:

```
Training: Read from data warehouse
Serving: Read from feature store
CDC: Keep both in sync
```

C. Online Learning

Problem: Model degrades over time (concept drift)

Solutions:

- **Batch:** Retrain weekly (most common)
- **Mini-batch:** Update daily
- **Online:** Update per-request (rare)

Monitoring:

- NDCG drop >2%: Alert
- CTR drop >5%: Rollback
- Latency p99 >500ms: Scale up

8. Common Challenges

A. Cold Start

Problem: New items, new users (no data)

Solutions:

- **Content-based:** Use item features (category, brand)
- **Popularity:** Show trending items
- **Exploration:** Random boosting (10%)
- **Transfer learning:** Use similar users/items

B. Position Bias

Problem: Top results get more clicks (not more relevant)

Solutions:

- **Inverse propensity weighting:** $\frac{\text{click}}{P(\text{click}|\text{position})}$
- **Randomization:** Shuffle top-K occasionally
- **Examination model:** Model position explicitly

C. Training/Serving Skew

Problem: Features different at train vs serve time

Causes:

- Inventory changes (in-stock $\rightarrow$ out-of-stock)
- Time lag (training on yesterday's data)
- Pipeline differences

Solutions:

- Use point-in-time features for training
- Bucketize fast-moving features (0, 1-10, 10+)
- Monitor feature distributions

D. Class Imbalance

Problem: CTR = 2% (98% negative)

Solutions:

- **Downsampling:** Sample negatives (keep all positives)
- **Class weights:** Higher weight for positives
- **Focal loss:** Focus on hard examples
- **SMOTE:** Synthetic oversampling (rarely used)

E. Handling Missing Values ✓

Problem: 20-30% of feature values often missing

Strategies by feature type:

- **Numerical:** Median/mean imputation, -999 flag
- **Categorical:** Add "missing" category
- **Boolean:** Impute as False + add is_missing flag

Advanced:

- **Model-based:** Use KNN, regression to predict
- **Multiple imputation:** Create multiple datasets
- **Indicator variables:** Add is_missing binary flag

Tree models (XGBoost): Handle missing natively!

F. Debugging: Offline ˜Online Mismatch

Symptom: Offline NDCG increases, but Online CTR/GMV drops.

Common Causes:

- **Data Leakage:** Using future signals
- **Training/Serving Skew:** Feature computation differs in prod
- **Objective Mismatch:** Optimizing clicks vs purchase/returns
- **Latency:** Model too slow, impacting UX
- **Simpson's Paradox:** Aggregate metrics hide slice degradation

Data Leakage Examples ✗:

- **Target leakage:** Using conversion rate from SAME session
- **Temporal leakage:** Using future clicks to predict past clicks
- **Train-test contamination:** Same user in both train/test
- **Label in features:** Purchase_count when predicting purchase
- **Proxy leakage:** Click_time when predicting click (circular!)

Investigation:

- Check feature distributions (Train vs Serve)
- Replay logged online requests through offline model
- Verify metric definitions (e.g., click definition)

9. Domain-Specific Models

A. Search Ranking

Best Model: LambdaMART (XGBoost)

- 100-300 features
- Train on (query, doc, label) tuples
- Optimize NDCG@10

Key Features:

- BM25, TF-IDF (textual relevance)
- Category/brand match
- Historical CTR (query-doc)
- Document quality (rating, sales)

B. Recommendations

Two-Stage:

1. **Candidate Generation:** Retrieve 1K items
   - Collaborative filtering (user-user, item-item)
   - Two-tower model (user/item embeddings)
   - ANN search (Faiss)
2. **Ranking:** Rank top-1K $\rightarrow$ top-50
   - XGBoost or DNN
   - Predict CTR, purchase probability

Models:

- **Matrix Factorization:** SVD, ALS
- **Neural CF:** User/item embeddings + MLP
- **Wide & Deep:** Memorization + generalization
- **DLRM:** Facebook's production model

C. Ads Ranking

Goal: Maximize revenue while maintaining user experience

Key Difference: Bid price matters

$$\text{Ad Score} = P(\text{click}) \times \text{Bid} \times \text{Quality}$$

Multi-Task:

- Predict CTR (P(click))
- Predict CVR (P(conversion — click))
- Expected value: CTR × CVR × Bid

Auction:

- **First-price:** Pay your bid
- **Second-price:** Pay next highest bid (VCG)

Models:

- DeepFM (most common)
- Wide & Deep
- DNN with multi-task heads

D. B2B Marketplace (Faire)

Key Difference: Retailer vs Consumer behavior

- **Repeat vs Discovery:** Retailers reorder bestsellers; Consumers seek novelty
- **Volume:** High AOV, bulk purchasing
- **Risk:** Net-60 terms (credit risk modeling)
- **Supply constraints:** Inventory limits matter more

Specific Features:

- **Retailer:** Store type (boutique vs online), location, credit score
- **Brand:** Margin, shipping time, minimum order value (MOV)
- **Graph:** Retailer-Brand history (past orders, returns)

Metrics:

- **GMV** (Gross Merchandise Value) is king
- **Sell-through rate:** Does the retailer actually sell the product?
- **Reorder Rate:** Critical for LTV (Lifetime Value)
- **First-order success:** Does first order lead to repeat?

Unique Challenges:

- **Match quality:** Right product for right retailer type
- **Payment risk:** Net-60 terms = need credit scoring
- **Seasonal patterns:** Retailers order 3-6 months ahead
- **Discovery paradox:** Need novelty but also reliability

Ranking Considerations:

- Balance: New brands (discovery) vs proven brands (safety)
- Personalize by: Store type, geography, price point
- Boost: Free return brands for new retailers (lower risk)
- Consider: Fulfillment speed (Faire Direct vs Brand ships)

10. Interview Tips

A. Problem Approach

1. Clarify the problem:

- What are we predicting? (CTR, relevance, etc.)
- What's the evaluation metric? (NDCG, AUC, etc.)
- What's the scale? (QPS, latency, data size)

2. Start simple:

- Baseline: Logistic Regression
- Stronger: XGBoost
- Advanced: Deep learning (if needed)

3. Feature engineering:

- Query features, doc features, query-doc
- Explain why each feature matters

4. Evaluation:

- Offline: NDCG, AUC
- Online: A/B test (CTR, conversion)

B. Common Questions

Ranking:

- Explain pointwise, pairwise, listwise LTR
- When to use each?
- What's NDCG and why use it?

Features:

- What features for search ranking?
- How to handle categorical features?
- Training/serving skew?

Models:

- XGBoost vs Neural Networks?
- How to handle cold start?
- Multi-task learning benefits?

Production:

- Latency optimization?
- How often to retrain?
- Monitoring and alerting?

C. Key Talking Points

- **Tradeoffs:** Always discuss (accuracy vs latency, complexity vs interpretability)
- **Numbers:** Give concrete examples (NDCG 0.75, latency 100ms, CTR 2%)
- **Experience:** "At [company], we used [model] and saw [X% improvement]"
- **Scale:** Understand QPS, data size, model size
- **Iterate:** Start simple, add complexity if needed

C2. Common Interview Mistakes to AVOID

Don't say:

- ✗"Deep learning is always better" (XGBoost wins 80% of time)
- ✗"We need more data" (without diagnosing bias vs variance)
- ✗"Accuracy is 95%" (wrong metric for imbalanced data)
- ✗"Just use BERT" (200ms latency, can't serve 10K QPS)
- ✗"Random split is fine" (causes data leakage in time-series)

Do say:

- ✓"It depends on..." (show you consider tradeoffs)
- ✓"Let me check train vs test error first" (bias-variance)
- ✓"For imbalanced data, I'd use AUC-ROC or PR-AUC"
- ✓"BERT for reranking top-100, not full corpus"
- ✓"Time-based split to prevent leakage"

C3. Numbers to Memorize

Metrics:

- NDCG: 0.6-0.7 (baseline), 0.7-0.8 (good), 0.8+ (excellent)
- AUC: 0.7-0.8 (good), 0.8-0.9 (excellent), 0.9+ (check for leakage)
- CTR: 1-5% (typical), 10%+ (promoted content)
- Conversion: 2-5% (e-commerce), 10-20% (SaaS)

Latency:

- Total budget: 200ms p99 (user-facing)
- Retrieval: 50ms (Elasticsearch)
- Feature fetch: 20-30ms (Redis)
- Model inference: 50-100ms (XGBoost/DNN)

Scale:

- Batch size: 32-256 (training)
- Features: 50-300 (typical)
- Trees: 100-500 (XGBoost)
- Learning rate: 0.01-0.1 (start), 0.001-0.01 (fine-tune)

D. Model Debugging & Error Analysis

When model underperforms:

1. Slice metrics by category

- Head vs tail queries (high vs low frequency)
- Per-category performance
- New vs returning users
- Mobile vs desktop

2. Error analysis

- Sample failed queries manually
- Identify patterns (missing features, wrong labels)
- Check if retrieval or ranking is failing

3. Feature importance

- Use XGBoost feature_importance
- Ablation study (remove one feature at a time)
- Check for feature leakage

3a. Model Interpretation (SHAP/LIME) ✓

- **SHAP (SHapley Additive exPlanations):** Game theory-based feature attribution
  - Shows each feature's contribution to prediction
  - Works for any model (XGBoost, NN, etc.)
  - Use: `shap.TreeExplainer(model)` for trees
- **LIME (Local Interpretable Model-agnostic):** Local linear approximation
  - Explains individual predictions
  - Perturbs input, trains simple model locally
  - Good for debugging edge cases

- **Partial Dependence Plots:** Show feature effect on prediction
- **When critical:** Regulated industries (finance, healthcare), debugging bias, explaining to stakeholders

4. Common root causes

- **Poor retrieval:** Not finding relevant docs
- **Label quality:** Noisy or biased labels
- **Feature issues:** Missing values, wrong encoding
- **Model complexity:** Under/overfitting

5. Calibration ✓

Problem: Model outputs aren't true probabilities

- Predicted 0.8 should mean 80% actually positive
- Important for: Ad auctions, risk assessment

Check calibration:

- Bin predictions: [0-0.1], [0.1-0.2], ..., [0.9-1.0]
- Calculate actual positive rate per bin
- Plot: Should align with diagonal

Fix calibration:

- **Platt scaling:** Train logistic regression on outputs
- **Isotonic regression:** Non-parametric calibration
- **Temperature scaling:** Divide logits by T

E. Production ML Best Practices

Deployment strategies:

- **Shadow mode:** New model runs in parallel, doesn't serve
- **Canary:** 1% traffic → 5% → 10% → 50% → 100%
- **Blue-green:** Two environments, instant switchover

Rollback triggers:

- Latency p99 $>$ 2x baseline
- Error rate $>$ 1%
- CTR drop $>$ 5%
- Zero-result rate $>$ 10%

Model staleness detection:

- Track prediction drift (distribution changes)
- Monitor feature drift
- Offline metric degradation
- Set retraining cadence (weekly/daily)

Remember: "Understand the problem, start simple, iterate with data, and always validate online."

Quick Reference Tables

Model Selection Guide

Use Case	Best Model	Key Metric	Latency
Search Ranking	LambdaMART (XGBoost)	NDCG@10	50-100ms
CTR Prediction (Ads)	DeepFM, Wide&Deep	AUC-ROC, Log Loss	10-50ms
Recommendation (Candidate)	Two-Tower, ALS	Recall@K	1-10ms
Recommendation (Ranking)	XGBoost, DNN	NDCG, CTR	20-100ms
Query Classification	BERT, Logistic Reg	Accuracy, F1	5-50ms
Semantic Search	BERT, Two-Tower	Recall@K, MRR	10-200ms

Feature Engineering Checklist

Category	Examples
Text Relevance	BM25, TF-IDF, Edit distance, Exact match, Embedding cosine similarity
Document Static	Category, Brand, Price, Rating, Review count, Age, Popularity
Document Dynamic	Inventory level, Recent CTR, Recent sales, Trending score
Query-Doc Interaction	Historical CTR (query-doc pair), Co-occurrence, Click-through pattern
User Context	Location, Device, Time of day, Session length, User cohort
User History	Past clicks, Past purchases, Browsing category, Avg order value

Metric Selection Guide

Task	Primary Metric	When to Use
Binary Classification	AUC-ROC Log Loss PR-AUC	Imbalanced data, ranking quality matters Calibrated probabilities needed Highly imbalanced (CTR $<$ 5%)
Ranking	NDCG@K MRR MAP	Graded relevance, position matters (default choice) First relevant result critical (navigational queries) All relevant results matter equally
Regression	RMSE MAE	Penalize large errors heavily Robust to outliers, interpretable
Multi-class	Accuracy F1 (macro/micro)	Balanced classes Imbalanced classes

Appendix: ML Fundamentals & Coding

A. ML Theory - Essential Q&A

1. Explain bias-variance tradeoff

Answer:

- **Bias:** Error from wrong assumptions (underfitting). High bias = model too simple
- **Variance:** Error from sensitivity to training data (overfitting). High variance = model too complex
- **Tradeoff:** Error $=$ Bias^2 $+$ Variance $+$ Irreducible Error
- Sweet spot: Balance both (e.g., ensemble methods)

2. L1 vs L2 regularization - when to use each?

Answer:

- **L1 (Lasso):** $\lambda \sum |w_i|$ - Sparse solutions, feature selection. Use when: Many irrelevant features
- **L2 (Ridge):** $\lambda \sum w_i^2$ - Smooth weights, no sparsity. Use when: All features matter, multicollinearity
- **Elastic Net:** $\alpha L1 + (1-\alpha)L2$ - Combines both benefits

3. How does gradient descent work? SGD vs Mini-batch vs Batch?

Answer:

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

- **Batch GD:** Use all data, slow but stable
- **SGD:** One sample, fast but noisy
- **Mini-batch:** 32-512 samples - best tradeoff (industry standard)

4. Explain Adam optimizer - why better than SGD?

Answer:

- Adaptive learning rates per parameter
- Combines momentum (first moment) + RMSprop (second moment)
- Benefits: Faster convergence, works well with sparse gradients, less tuning
- When to use: Deep learning (default). Use SGD with momentum for: Simple models, better generalization needed

Adam equations:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ typical)

4a. Gradient Clipping ✓

Answer:

- **Problem:** Exploding gradients in RNNs, deep networks
- **Clip by value:** $g =$ clip$(g, -\theta, \theta)$ (e.g., $\theta = 5$)
- **Clip by norm:** $g = g \cdot \min(1, \frac{\theta}{||g||})$
- **When critical:** RNNs, LSTMs, very deep networks (50+ layers)
- **Symptom:** Loss becomes NaN, weights explode

4b. Learning Rate Scheduling ✓

Answer:

- **Step decay:** Reduce LR by factor every N epochs ($\eta = \eta_0 \times 0.5^{\lfloor epoch/N \rfloor}$)
- **Exponential decay:** $\eta = \eta_0 \times e^{-kt}$ - Smooth continuous decay
- **Cosine annealing:** $\eta = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{T_{cur}}{T_{max}}\pi))$
- **Warmup:** Start with low LR, gradually increase (prevents early divergence)
- **ReduceOnPlateau:** Reduce when validation metric stops improving

When to use: Step decay (simple baseline), Cosine (SOTA), Warmup + Cosine (Transformers)

5. How to handle overfitting? (5+ techniques)

Answer:

1. More training data
2. Regularization (L1/L2, dropout)

3. Early stopping (monitor validation loss)
4. Data augmentation (images, text)
5. Simpler model (reduce capacity)
6. Ensemble methods (reduce variance)
7. Cross-validation

5a. Cross-Validation Variants ✓

Answer:

- **K-Fold CV:** Split data into K folds, train on K-1, test on 1 (typical K=5 or 10)
- **Stratified K-Fold:** Preserve class distribution in each fold (use for imbalanced data!)
- **Time-series CV:** Forward chaining - train on [1..t], test on [t+1..t+k] (prevents leakage)
- **Leave-One-Out CV:** K=N (expensive, use only for small datasets ¡1000)
- **Group K-Fold:** Keep same user/session in same fold (prevents leakage)

Interview tip: Always use time-based split for temporal data, stratified for imbalanced!

5b. Hyperparameter Tuning ✓

Answer:

- **Grid Search:** Try all combinations (exhaustive but slow)
  - Use: Few hyperparameters (¡4), discrete values
  - Example: learning_rate=[0.01, 0.1], depth=[3, 5, 7]
- **Random Search:** Sample randomly (often better than grid!)
  - More efficient than grid search (Bergstra & Bengio 2012)
  - Can match grid performance with fewer trials
  - Use: Many hyperparameters, continuous spaces
- **Bayesian Optimization:** Model which params work best
  - Tools: Optuna, Hyperopt, Ray Tune
  - Use: Expensive models (¿10min/trial)
- **Halving:** Start many, kill bad ones early (Hyperband)

5c. Early Stopping ✓

Answer:

- **Strategy:** Stop when validation loss stops improving
- **Patience:** Wait N epochs (typical N=3-10) before stopping
- **Implementation:** Track best val loss, stop if no improvement
- **Restore:** Load best weights (not final weights!)
- **Benefit:** Prevents overfitting, saves training time

6. Precision vs Recall - when to optimize for each?

Answer:

$$\text{Precision} = \frac{TP}{TP + FP} \quad \text{Recall} = \frac{TP}{TP + FN}$$

- **Optimize Precision:** When false positives are costly (spam detection - don't block real emails)
- **Optimize Recall:** When false negatives are costly (fraud detection - catch all fraud)
- **F1 Score:** Balance both

7. Why does XGBoost work so well? Technical details

Answer:

- **Boosting:** Sequential trees, each corrects previous errors
- **Regularization:** L1/L2 on leaf weights, max depth, min child weight
- **2nd order optimization:** Uses Hessian (not just gradient)
- **Handling sparsity:** Learns best direction for missing values
- **Parallel processing:** Fast training via column block structure
- **Tree pruning:** Max depth + gamma (complexity control)

8. Explain batch normalization - why does it help?

Answer:

- Normalizes layer inputs: $\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}$
- Benefits: (1) Faster convergence, (2) Higher learning rates, (3) Regularization effect, (4) Less sensitive to initialization

- When: Deep networks (6+ layers), CNNs
- Alternative: Layer normalization (for RNNs/Transformers)

9. How do embeddings work? Word2Vec vs BERT?

Answer:

- **Word2Vec:** Static embeddings, context-free. CBOW (predict word from context) or Skip-gram (predict context from word)
- **BERT:** Contextual embeddings, bidirectional. ”bank” has different embeddings in ”river bank” vs ”bank account”
- **Training:** Word2Vec (unsupervised), BERT (masked LM + next sentence prediction)
- **Use:** Word2Vec for simple tasks, BERT for complex NLP

10. Cross-validation - when NOT to use it?

Answer:

Don’t use when:

- **Time series:** Use time-based split instead (future can’t predict past)
- **Large datasets:** Computationally expensive, single split sufficient
- **Data leakage risk:** Related samples might end up in train/val
- **Production:** Need realistic temporal evaluation

11. Which loss function to use? ✓

Answer:

- **Binary Classification:**
  - Cross-entropy (log loss): Standard choice, probabilistic
  - Hinge loss: SVM, when you need margin
- **Multi-class:**
  - Categorical cross-entropy: Standard (softmax output)
  - Focal loss: For class imbalance (down-weights easy examples)
- **Regression:**
  - MSE: Penalizes large errors heavily, assumes Gaussian
  - MAE: Robust to outliers, more interpretable
  - Huber: Best of both (MSE for small, MAE for large errors)
- **Ranking:**
  - Pairwise (Hinge): LambdaRank, RankNet
  - Listwise: LambdaMART (optimizes NDCG directly)

Interview tip: Always justify your choice based on data distribution and outliers!

B. Implement from Scratch - Coding Checklist

Common ”implement from scratch” questions at FAANG:

1. Calculate AUC-ROC from scratch

```
def auc_roc(y_true, y_scores):
    # Sort by scores descending
    sorted_indices = sorted(range(len(y_scores)),
                          key=lambda i: y_scores[i],
                          reverse=True)
    sorted_labels = [y_true[i] for i in sorted_indices]

    # Count correctly ranked pairs
    num_pos = sum(y_true)
    num_neg = len(y_true) - num_pos

    if num_pos == 0 or num_neg == 0:
        return 0.5  # Undefined, return random

    # Iterate high score to low score
    # If we see a Negative, it is ranked lower than all
    # preceding Positives (which is good).
    correct_pairs = 0
```

```
    positives_so_far = 0

    for label in sorted_labels:
        if label == 1:  # Positive
            positives_so_far += 1
        else:  # Negative
            correct_pairs += positives_so_far

    auc = correct_pairs / (num_pos * num_neg)
    return auc
```

2. K-Means clustering

```
def kmeans(X, k, max_iters=100):
    # Random init centroids
    if k > len(X):
        raise ValueError(f"k={k} cannot exceed n_samples={len(X)}")
    centroids = X[np.random.choice(len(X), k, replace=False)]

    for _ in range(max_iters):
        # Assign points to nearest centroid
        distances = np.sqrt(((X[:, None] - centroids)**2).sum(2))
        labels = np.argmin(distances, axis=1)

        # Update centroids
        new_centroids = []
        for i in range(k):
            points = X[labels == i]
            if len(points) > 0:
                new_centroids.append(points.mean(0))
            else:
                # Empty cluster, reinitialize
                new_centroids.append(X[np.random.randint(len(X))])
        new_centroids = np.array(new_centroids)

        # Check convergence
        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids

    return labels, centroids
```

3. Sigmoid & Logistic Regression

```
def sigmoid(z):
    # Clip to prevent overflow
    return 1 / (1 + np.exp(-np.clip(z, -500, 500)))

def logistic_regression(X, y, lr=0.01, epochs=1000):
    m, n = X.shape
    weights = np.zeros(n)
    bias = 0

    for _ in range(epochs):
        # Forward pass
        z = np.dot(X, weights) + bias
        predictions = sigmoid(z)

        # Gradients
```

```
        dw = (1/m) * np.dot(X.T, (predictions - y))
        db = (1/m) * np.sum(predictions - y)

        # Update
        weights -= lr * dw
        bias -= lr * db

    return weights, bias
```

4. Calculate NDCG@K

```
def ndcg_at_k(y_true, y_scores, k):
    # Get top-k indices
    top_k_idx = np.argsort(y_scores)[::-1][:k]

    # DCG
    dcg = sum((2**y_true[i] - 1) / np.log2(pos + 2)
              for pos, i in enumerate(top_k_idx))

    # IDCG (ideal)
    ideal_idx = np.argsort(y_true)[::-1][:k]
    idcg = sum((2**y_true[i] - 1) / np.log2(pos + 2)
               for pos, i in enumerate(ideal_idx))

    return dcg / idcg if idcg > 0 else 0
```

4a. Precision@K, Recall@K, MRR

```
def precision_at_k(y_true, y_scores, k):
    if k == 0:
        return 0
    top_k_idx = np.argsort(y_scores)[::-1][:k]
    relevant_in_topk = sum(y_true[i] for i in top_k_idx)
    return relevant_in_topk / k

def recall_at_k(y_true, y_scores, k):
    top_k_idx = np.argsort(y_scores)[::-1][:k]
    relevant_in_topk = sum(y_true[i] for i in top_k_idx)
    total_relevant = sum(y_true)
    return relevant_in_topk / total_relevant if total_relevant > 0 else 0

def mrr(y_true, y_scores):
    # Mean Reciprocal Rank
    sorted_idx = np.argsort(y_scores)[::-1]
    for rank, i in enumerate(sorted_idx, 1):
        if y_true[i] == 1:
            return 1.0 / rank
    return 0  # No relevant item found
```

5. Cosine Similarity

```
def cosine_similarity(v1, v2):
    dot_product = np.dot(v1, v2)
    norm_v1 = np.linalg.norm(v1)
    norm_v2 = np.linalg.norm(v2)
    if norm_v1 == 0 or norm_v2 == 0:
        return 0  # Undefined, return 0
    return dot_product / (norm_v1 * norm_v2)
```

6. Confusion Matrix & Metrics

```
def confusion_matrix(y_true, y_pred):
    # For binary classification
    tp = sum((yt == 1 and yp == 1)
             for yt, yp in zip(y_true, y_pred))
    fp = sum((yt == 0 and yp == 1)
             for yt, yp in zip(y_true, y_pred))
    tn = sum((yt == 0 and yp == 0)
             for yt, yp in zip(y_true, y_pred))
    fn = sum((yt == 1 and yp == 0)
             for yt, yp in zip(y_true, y_pred))

    return {'TP': tp, 'FP': fp, 'TN': tn, 'FN': fn}

def metrics_from_cm(cm):
    precision = cm['TP'] / (cm['TP'] + cm['FP']) \
                if (cm['TP'] + cm['FP']) > 0 else 0
    recall = cm['TP'] / (cm['TP'] + cm['FN']) \
             if (cm['TP'] + cm['FN']) > 0 else 0
    accuracy = (cm['TP'] + cm['TN']) / sum(cm.values())
    f1 = 2 * (precision * recall) / (precision + recall) \
         if (precision + recall) > 0 else 0
    return {'precision': precision, 'recall': recall,
            'accuracy': accuracy, 'f1': f1}
```

Key Concepts to Know How to Implement:

- Linear/Logistic Regression with gradient descent
- K-Means, K-NN
- Decision Tree split (Gini/Entropy)
- Precision, Recall, F1, AUC-ROC
- NDCG, MRR, MAP
- Confusion matrix
- Train/test split, cross-validation
- L1/L2 regularization
- Batch normalization
- Sigmoid, ReLU, Softmax activations

C. ML System Design Framework (6 Steps)

Use this framework for any "Design X system" question

Step 1: Problem Formulation (5 min)

- Clarify the problem: "What exactly are we optimizing?"
- Define success: Business metric (GMV, engagement) + ML metric (NDCG, AUC)
- Scope: Latency requirements? Scale (QPS, users, items)?
- ML framing: Classification? Ranking? Regression?

Example: "Design YouTube video recommendations"

- Problem: Recommend videos users will watch & enjoy
- Business metric: Watch time, session length
- ML metric: NDCG@20, Recall@100
- Framing: Two-stage (retrieval + ranking)

Step 2: Data Strategy (5-7 min)

- What data do we have? (user history, video metadata, engagement)
- Labels: Implicit (clicks, watch time) or explicit (likes)?
- Data quality: Biases (position bias, popularity bias)
- Data volume: How much? (millions of users, billions of videos)
- Cold start: New users, new videos?

Key questions to address:

- How to collect training data?
- How to handle missing labels?
- How to ensure data freshness?

Step 3: Feature Engineering (7-10 min)

- **User features:** Demographics, history, preferences
- **Item features:** Metadata, popularity, quality
- **Context features:** Time, device, location
- **Interaction features:** User-item affinity, CTR

Example features for YouTube:

- User: Watch history (last 50 videos), subscriptions, demographics
- Video: Title/description embeddings, category, upload date, views, likes
- User-Video: Watched same channel before? Similar category?
- Context: Time of day, device type

Step 4: Model Selection (7-10 min)

- Start simple: "I'd begin with a baseline..."
- Propose 2-3 approaches with tradeoffs
- Justify choice based on scale, latency, complexity

Standard approach (2-stage):

```
Stage 1: Candidate Generation (retrieve 1000s)
  - Collaborative filtering
  - Two-tower model + ANN search
  - Multiple retrievers (content, CF, trending)

Stage 2: Ranking (rank top-1K → top-100)
  - LambdaMART (XGBoost) - listwise LTR
  - Or: Deep neural network
  - Rich features, optimize NDCG
```

Discuss tradeoffs:

- XGBoost: Fast, interpretable, strong baseline
- DNN: Captures complex patterns, needs more data
- Two-tower: Scalable retrieval, pre-compute embeddings

Step 5: Training & Evaluation (5-7 min)

Training:

- Train/val/test split: Time-based (last 7 days = test)
- Loss function: Listwise (NDCG), pairwise, or pointwise
- Frequency: Weekly (search), daily (ads)
- Challenges: Class imbalance, position bias

Offline Evaluation:

- Primary: NDCG@10, MRR
- Per-query analysis
- Breakdown: head vs tail queries, cold start

Online Evaluation:

- A/B test: 5-10% traffic, 1-2 weeks
- Metrics: CTR, conversion, watch time, GMV
- Guardrails: Latency, zero-result rate

Step 6: Deployment & Monitoring (5-7 min)

Serving Architecture:

```
Request → Candidate Generation (50ms)
        → Feature Fetch (30ms, Redis)
        → Model Inference (80ms)
        → Post-processing (20ms)
```

```
        → Return results
Total: 180ms
```

Optimization:

- Feature store: Redis for low-latency lookup
- Model: Quantization, pruning, distillation
- Caching: Cache popular queries (80/20 rule)
- Batch inference: Process multiple requests together

Monitoring:

- **Model metrics:** NDCG drop $>$2% $\rightarrow$ alert
- **Business metrics:** CTR, conversion, revenue
- **System metrics:** Latency p99, error rate, QPS
- **Data quality:** Feature drift, missing values

Failure modes:

- Model timeout $\rightarrow$ fallback to simple ranker
- Feature store down $\rightarrow$ use cached features
- No candidates $\rightarrow$ show popular/trending

D. Example Walkthrough: "Design Ad Click Prediction"

1. Problem: Predict P(click — user, ad, context) for ad ranking

2. Data: User events (impressions, clicks), ad metadata, context - Label: Binary (click=1, no click=0) - Challenge: Highly imbalanced (CTR ˜2%)

3. Features: - User: Demographics, browsing history, past ad interactions - Ad: Title, image, category, advertiser - User-Ad: Historical CTR, category match - Context: Time, device, page content

4. Model: DeepFM or Wide & Deep - Wide: Memorize user-ad pairs (high CTR combinations) - Deep: Generalize to unseen combinations - Loss: Binary cross-entropy with class weights

5. Evaluation: - Offline: AUC-ROC (0.75+), Log Loss - Online: CTR, revenue per impression, user experience

6. Serving: 20ms latency budget - Pre-compute ad embeddings - User features from Redis (5ms) - Model inference (10ms) - Rank ads by: score = P(click) × bid × quality

---

Framework Summary: "Formulate $\rightarrow$ Data $\rightarrow$ Features $\rightarrow$ Model $\rightarrow$ Evaluate $\rightarrow$ Deploy"