Simulacro Entrevista Lead/Senior ML Engineer - Parte 2

Continuación de preguntas 41-115

Preguntas 41-45: BankChurn (continuación)

Pregunta 41: Risk Level Classification

¿Cómo categorizas el riesgo de churn?

Respuesta:

```
        # prediction.py
        def _assign_risk_level(self, probability: float) -> str:
            if probability < 0.3:
                return "low"
            elif probability < 0.7:
                return "medium"
            else:
                return "high"
```

Uso en negocio: - **High**: Contacto proactivo inmediato - **Medium**: Ofertas de retención - **Low**: Monitoreo estándar

---

Pregunta 42: Feature Contribution Explanation

¿Cómo explicas las predicciones?

Respuesta:

```
        # prediction.py
        def _generate_explanation(self, X_row) -> Dict[str, float]:
            # Enfoque simplificado: distancia a media por feature
            contributions = {}
            for col in self.feature_names:
                mean_val = self.feature_means.get(col, 0)
                diff = (X_row[col] - mean_val) / (self.feature_stds.get(col, 1) + 1e-8)
                contributions[col] = float(diff)
            return contributions
```

Limitación reconocida: No es SHAP values real, pero da intuición inicial sin dependencia adicional.

Pregunta 43: Model Metadata

¿Qué metadata guardas con el modelo?

Respuesta:

```
{
  "version": "1.0.0",
  "trained_at": "2024-11-20T10:30:00Z",
  "config_hash": "abc123def456",
  "test_metrics": {
    "f1_score": 0.82,
    "auc_roc": 0.853
  },
  "feature_names": ["CreditScore", "Age", ...],
  "training_samples": 8000,
  "git_commit": "abc123"
}
```

Uso: Trazabilidad, comparación entre versiones, debugging.

Pregunta 44: Lifespan vs On-Event Loading

¿Por qué usas lifespan en lugar de startup event?

Respuesta:

```
# FastAPI >= 0.95 depreca @app.on_event
@asynccontextmanager
async def lifespan(app: FastAPI):
    # Startup
    load_model()
    yield
    # Shutdown (cleanup)

app = FastAPI(lifespan=lifespan)
```

Beneficios: - Patrón moderno recomendado - Context manager claro para setup/teardown - Mejor manejo de recurso async

---

Pregunta 45: Test Coverage Strategy

¿Cómo alcanzaste 77% coverage?

Respuesta:

```
# pytest.ini
[pytest]
addopts = --cov=src/bankchurn --cov-report=term-missing

# Estrategia:
# 1. Unit tests por módulo
# 2. Integration tests para workflows completos
# 3. Edge cases: datos vacíos, missing columns, tipos inválidos
# 4. Error paths: excepciones esperadas
```

Cobertura por módulo: - training.py: 85% - evaluation.py: 80% - config.py: 90% - cli.py: 60% (I/O difícil de testear)

Preguntas 51-60: CarVision (continuación)

Pregunta 51: Feature Type Inference

¿Cómo detectas tipos de features automáticamente?

Respuesta:

```python
# data.py
def infer_feature_types(df, cfg):
    num_cfg = cfg.get("numeric_features", [])
    cat_cfg = cfg.get("categorical_features", [])

    if not num_cfg:
        num_cfg = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
    if not cat_cfg:
        cat_cfg = df.select_dtypes(include=["object", "category"]).columns.tolist()

    return num_cfg, cat_cfg
```

Beneficio: Config minimalista (`[]`) usa inferencia automática.

Pregunta 52: Segment Error Analysis

¿Cómo analizas errores por segmento?

Respuesta:

```python
def _analyse_errors_by_segment(df, y_true, y_pred, segment_cols):
    results = []
    for col in segment_cols:  # ["condition", "type", "model_year"]
        for val, group in df.groupby(col):
            if len(group) < 30:
                continue
            results.append({
                "segment": f"{col}={val}",
                "rmse": rmse(group[target], y_pred[group.index]),
                "n_samples": len(group)
            })
    return pd.DataFrame(results)
```

Hallazgo típico: Mayor error en autos muy nuevos (pocos datos) o muy viejos (alta variabilidad).

Pregunta 53: Dual Application (API + Dashboard)

¿Cómo manejas API y Streamlit juntos?

Respuesta:

```
app/
├── fastapi_app.py     # REST API (port 8002)
└── streamlit_app.py   # Dashboard (port 8501)
```

Ambos usan:

```python
from src.carvision.features import FeatureEngineer
from src.carvision.data import clean_data

# Mismo pipeline, diferentes interfaces
model = joblib.load(MODEL_PATH)
```

Docker Compose:

```yaml
services:
  carvision-api:
    command: uvicorn app.fastapi_app:app --port 8002
  carvision-dashboard:
    command: streamlit run app/streamlit_app.py --server.port 8501
```

Pregunta 54: Price Category Leakage

¿Por qué price_category causa leakage?

Respuesta:

```
# features.py - ANTES (bug)
X["price_category"] = pd.cut(X["price"], bins=[0, 5000, 15000, 50000, np.inf])
```

Problema: `price_category` depende de `price` (el target). - En training: tiene el valor correcto - En inference: `price` no existe → error o leakage si se imputa

Solución: `drop_columns: ["price_per_mile", "price_category"]` en config.

Pregunta 55: Split Indices Persistence

¿Por qué guardas los índices del split?

Respuesta:

```
def save_split_indices(indices, path):
    with open(path, "w") as f:
        json.dump({k: v.tolist() for k, v in indices.items()}, f)
```

Uso:

```
# Reproducir exactamente el mismo split
indices = load_split_indices("split_indices.json")
X_train = X.iloc[indices["train"]]
X_test = X.iloc[indices["test"]]
```

Beneficio: Comparar modelos con EXACTAMENTE los mismos datos de test.

Pregunta 56: Dummy Baseline

¿Por qué comparas con DummyRegressor?

Respuesta:

```
dummy = DummyRegressor(strategy="median")
baseline_rmse = rmse(y_test, dummy.fit(X_train, y_train).predict(X_test))
```

Estrategias disponibles: - `mean`: Predice promedio - `median`: Predice mediana (robusta a outliers) - `constant`: Valor fijo

Interpretación: Si tu modelo no supera dummy, no agrega valor.

Pregunta 57: Random Forest para Regresión

¿Por qué RF y no XGBoost para CarVision?

Respuesta:

Criterio	RandomForest	XGBoost
Simplicidad	Menos hiperparámetros	Muchos HP críticos
Robustez	Menos sensible a HP	Requiere tuning fino
Interpretabilidad	Feature importance directa	Más complejo
Performance	Competitivo en tabular	Marginalmente mejor

Decisión: RF es "good enough" con menor riesgo de overfitting y configuración más simple.

Pregunta 58: MAPE Calculation

¿Por qué sumas epsilon en MAPE?

Respuesta:

```
mape = np.mean(np.abs((y_true - y_pred) / (y_true + 1e-8))) * 100
```

Problema: Si `y_true = 0`, división por cero. **Solución**: `+ 1e-8` evita división por cero.

Alternativa mejor para precios:

```
# Symmetric MAPE
smape = np.mean(2 * np.abs(y_true - y_pred) / (np.abs(y_true) + np.abs(y_pred)))
```

Pregunta 59: Streamlit Sections

¿Cómo organizas el dashboard?

Respuesta:

```
# streamlit_app.py
tab1, tab2, tab3, tab4 = st.tabs([
    " Overview",
    " Market Analysis",
    " Model Metrics",
    " Price Predictor"
])

with tab1:
    display_overview_kpis()
with tab4:
    # Formulario de predicción
    brand = st.selectbox("Brand", brands)
    year = st.slider("Year", 1990, 2024)
    if st.button("Predict"):
        pred = model.predict(features)
        st.success(f"Estimated Price: ${pred:,.0f}")
```

Pregunta 60: API vs Dashboard Trade-offs

¿Cuándo recomiendas API vs Dashboard?

Respuesta:

Caso de Uso	Recomendación
Integración con otros sistemas	API
Usuarios no técnicos	Dashboard
Alto volumen	API
Exploración ad-hoc	Dashboard
Automatización	API
Demos/POC	Dashboard

CarVision ofrece ambos: API para integración con sistemas de dealers, Dashboard para analistas de mercado.

Preguntas 63-70: TelecomAI (continuación)

Pregunta 63: Unified Pipeline Benefit

¿Por qué un solo artefacto pipeline?

Respuesta:

```
# ANTES (2 archivos)
preprocessor = joblib.load("preprocessor.pkl")
model = joblib.load("model.pkl")
X_proc = preprocessor.transform(X)
pred = model.predict(X_proc)

# AHORA (1 archivo)
pipeline = joblib.load("model.joblib")
pred = pipeline.predict(X)
```

Beneficios: 1. Deployment más simple 2. Imposible desincronizar preprocessor/model 3. Una sola versión para auditar

---

Pregunta 64: Config YAML Structure

¿Cómo estructuras el config de TelecomAI?

Respuesta:

```yaml
project_name: TelecomAI-Customer-Intelligence
random_seed: 42

paths:
  data_csv: users_behavior.csv
  model_path: artifacts/model.joblib

features: [calls, minutes, messages, mb_used]
target: is_ultra

split:
  test_size: 0.2
  stratify: true

model:
  name: gradient_boosting
  params:
    n_estimators: 200
    max_depth: 2
```

Principio: Toda configuración externalizada, ningún valor hardcodeado.

Pregunta 65: Stratify con Clasificación Binaria

¿Cuándo es crítica la estratificación?

Respuesta:

Crítica cuando: - Clases desbalanceadas (< 70/30) - Dataset pequeño (< 5K samples) - Métrica sensible a distribució (precision, recall)

```
# Sin estratificación en 2K samples con 30% positivos:
# Test set podría tener 20% o 40% positivos → métricas no comparables
```

Pregunta 66: Simple Model Debugging

¿Cómo debuggeas un modelo simple?

Respuesta:

```
# 1. Check feature distributions
print(X_train.describe())
print(X_test.describe())  # Debería ser similar

# 2. Check target distribution
print(y_train.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))

# 3. Learning curve
from sklearn.model_selection import learning_curve
train_sizes, train_scores, val_scores = learning_curve(
    model, X, y, cv=5, scoring="roc_auc"
)
```

Pregunta 67: Gradient Boosting Learning Rate

¿Qué pasa si lr es muy alto o muy bajo?

Respuesta:

lr	Efecto
0.5+ (alto)	Convergencia rápida, riesgo de overfitting
0.01-0.1 (medio)	Balance típico
< 0.01 (bajo)	Necesita muchos estimators, más robusto

```
# TelecomAI usa lr=0.05, n_estimators=200
# Conservador pero estable
```

Regla práctica: `lr * n_estimators ≈ 10-20` para convergencia.

Pregunta 68: Evaluation Metrics Save

¿Cómo persistes métricas?

Respuesta:

```python
# evaluation.py
def evaluate_model(pipeline, X_test, y_test, cfg):
    metrics = compute_classification_metrics(y_test, y_pred, y_proba)

    # Save to YAML
    with open(cfg.paths["metrics_path"], "w") as f:
        yaml.safe_dump(metrics, f)

    return metrics
```

Formato YAML para legibilidad humana:

```yaml
accuracy: 0.812
precision: 0.785
recall: 0.743
f1: 0.763
roc_auc: 0.840
```

Pregunta 69: FastAPI App TelecomAI

¿Cómo estructuras la API?

Respuesta:

```python
# app/fastapi_app.py
class UserBehavior(BaseModel):
    calls: int
    minutes: float
    messages: int
    mb_used: float

@app.post("/predict")
async def predict_plan(user: UserBehavior):
    features = pd.DataFrame([user.dict()])
    pred = pipeline.predict(features)[0]
    proba = pipeline.predict_proba(features)[0][1]
    return {
        "recommended_plan": "Ultra" if pred == 1 else "Basic",
        "confidence": float(proba)
    }
```

Pregunta 70: TelecomAI Business Context

¿Cómo se usa el modelo en el negocio?

Respuesta:

Contexto: Recomendar plan de datos (Basic vs Ultra) basado en patrones de uso.

Flujo: 1. Cliente usa servicio por periodo de prueba 2. Sistema recolecta métricas: calls, minutes, messages, mb_used 3 Modelo predice plan óptimo 4. Ventas contacta con oferta personalizada

Valor: - Reduce churn por plan inadecuado - Aumenta ARPU en usuarios infraservidos - Mejora satisfacción del cliente

---

Preguntas 73-80: Arquitectura (continuación)

Pregunta 73: Error Handling Philosophy

¿Cuál es tu filosofía de manejo de errores?

Respuesta:

1. **Fail fast**: Validar inputs al inicio
2. **Errores específicos**: `ValueError` vs `FileNotFoundError` vs genérico
3. **Logs contextuales**: Incluir datos relevantes en el error
4. **Graceful degradation**: Servicio funciona con capacidad reducida

```
# Malo
try:
    do_something()
except Exception:
    pass

# Bueno
try:
    do_something(input_data)
except ValueError as e:
    logger.error(f"Invalid input {input_data}: {e}")
    raise HTTPException(400, f"Invalid input: {e}")
except FileNotFoundError as e:
    logger.error(f"Model file missing: {e}")
    raise HTTPException(503, "Model not available")
```

Pregunta 74: Dependency Management

¿Cómo manejas dependencias entre proyectos?

Respuesta:

```
Projects Tripe Ten/
├── common_utils/         # Shared code
│   ├── __init__.py
│   ├── logger.py
│   └── seed.py
├── BankChurn-Predictor/
│   └── requirements.txt  # Project-specific deps
├── CarVision-Market-Intelligence/
│   └── requirements.txt
└── TelecomAI-Customer-Intelligence/
    └── requirements.txt
```

common_utils en cada project:

```
import sys
sys.path.insert(0, str(Path(__file__).parent.parent.parent))
from common_utils.seed import set_seed
```

Pregunta 75: API Versioning

¿Cómo versionarías la API?

Respuesta:

```
# Opción 1: Path versioning
@app.post("/v1/predict")
@app.post("/v2/predict")

# Opción 2: Header versioning
@app.post("/predict")
async def predict(request: Request):
    version = request.headers.get("API-Version", "1")

# Opción 3: Query param
@app.post("/predict")
async def predict(version: str = Query("1")):
```

Recomendación: Path versioning es más explícito y cacheable.

Pregunta 76: Configuration Hierarchy

¿Cómo manejas diferentes environments?

Respuesta:

```
# configs/config.yaml (base)
mlflow:
  tracking_uri: "file:./mlruns"  # Default: local

# Override via environment variables
# MLFLOW_TRACKING_URI=http://mlflow.prod:5000

# O archivos separados
# configs/config.dev.yaml
# configs/config.prod.yaml
```

Carga con override:

```
config = BankChurnConfig.from_yaml("configs/config.yaml")
config.mlflow.tracking_uri = os.getenv("MLFLOW_URI", config.mlflow.tracking_uri)
```

Pregunta 77: Async vs Sync en FastAPI

¿Cuándo usas async?

Respuesta:

```
        # Sync - operaciones CPU-bound (ML inference)
        @app.post("/predict")
        def predict_sync(data: CustomerData):
            return model.predict(data)  # CPU bound

        # Async - operaciones I/O bound
        @app.get("/health")
        async def health_async():
            await check_external_service()  # Network call
```

ML inference es CPU-bound: `def` es preferible para evitar bloquear event loop.

Pregunta 78: Testing Pyramid

¿Cómo estructuras tus tests?

Respuesta:

```
Tests/
├── Unit (70%)
│   ├── test_config.py
│   ├── test_models.py
│   └── test_evaluation.py
├── Integration (20%)
│   └── test_training.py (E2E workflow)
└── E2E (10%)
    └── test_api.py (requests reales)
```

Pirámide: Muchos unit tests (rápidos), pocos E2E (lentos pero valiosos).

Pregunta 79: Code Review Checklist

¿Qué revisas en un PR de ML?

Respuesta:

1. **Data leakage**: ¿Split antes de fit?
2. **Reproducibilidad**: ¿Seeds configurados?
3. **Tests**: ¿Cubren happy path y edge cases?
4. **Config**: ¿Valores hardcodeados?
5. **Métricas**: ¿Apropiadas para el problema?
6. **Logging**: ¿Suficiente para debugging?
7. **Documentation**: ¿Model card actualizado?

Pregunta 80: Technical Debt Management

¿Cómo manejas deuda técnica?

Respuesta:

Categorización: - **Critical**: Bugs de seguridad, data leakage → Sprint actual - **High**: Tests faltantes, logging pobre - Próximo sprint - **Medium**: Refactoring, documentación → Backlog priorizado - **Low**: Nice-to-have → Tech debt day mensual

Tracking: Issues en GitHub con label `tech-debt` y estimación de impacto.

Preguntas 83-90: CI/CD (continuación)

Pregunta 83: Matrix Testing Strategy

¿Por qué matrix Python 3.11/3.12?

Respuesta:

```yaml
strategy:
  matrix:
    python-version: ['3.11', '3.12']
    project: [BankChurn, CarVision, TelecomAI]
```

Razones: - **3.11**: Versión estable ampliamente usada - **3.12**: Versión más reciente, validar compatibilidad - **3 proyectos** Detectar regresiones cross-project

Total jobs: 2 × 3 = 6 combinaciones paralelas.

Pregunta 84: Fail-Fast Strategy

¿Cuándo usar fail-fast: false?

Respuesta:

```yaml
strategy:
  fail-fast: false  # Continúa aunque un job falle
```

Usar `false` cuando: - Quieres ver TODOS los errores, no solo el primero - Jobs son independientes (matriz de versiones) Debugging de problemas de compatibilidad

Usar `true` cuando: - Jobs dependientes - Quieres feedback rápido - Recursos de CI limitados

---

Pregunta 85: Docker Build Caching

¿Cómo optimizas builds de Docker?

Respuesta:

```dockerfile
# Orden óptimo de COPY
COPY requirements.txt .
RUN pip install -r requirements.txt  # Cached si requirements no cambia
COPY src/ ./src/                     # Solo copia código
```

CI caching:

```
      - uses: docker/build-push-action@v5
        with:
          cache-from: type=gha
          cache-to: type=gha,mode=max
```

Pregunta 86: Security Scanning Results

¿Qué haces con findings de seguridad?

Respuesta:

Proceso: 1. **Critical/High**: Bloquea PR, fix inmediato 2. **Medium**: Documenta, fix en siguiente sprint 3. **Low**: Backlog evaluar riesgo/esfuerzo 4. **False positives**: Añadir a `.gitleaksignore` con justificación

```
      # .gitleaksignore
      # False positive: example API key in documentation
      docs/examples/api_usage.md:15
```

Pregunta 87: Coverage Thresholds

¿Por qué 70% coverage mínimo?

Respuesta:

```
      - name: Check coverage threshold
        run: |
          coverage report --fail-under=70
```

Razón del 70%: - **< 60%**: Riesgo de bugs no detectados - **70-80%**: Balance costo/beneficio típico - **> 90%**: Diminishing returns, tests frágiles

No todo necesita tests: I/O, logging, error messages triviales.

Pregunta 88: Integration Test Strategy

¿Qué cubren tus integration tests?

Respuesta:

```
      integration-test:
        steps:
          - run: docker-compose -f docker-compose.demo.yml up -d
          - run: |
              # Wait for services
              sleep 30
              # Test each API
              curl http://localhost:8001/health
              curl http://localhost:8002/health
              curl -X POST http://localhost:8001/predict -d '...'
```

Cobertura: Health checks, predicciones básicas, formato de respuesta.

Pregunta 89: Documentation Validation

¿Cómo validas documentación?

Respuesta:

```yaml
doc-validation:
  steps:
    - name: Check markdown links
      run: |
        npm install -g markdown-link-check
        find . -name "*.md" -exec markdown-link-check {} \;

    - name: Build mkdocs
      run: |
        pip install mkdocs
        mkdocs build --strict
```

Valida: Links rotos, sintaxis markdown, build de docs.

Pregunta 90: Deployment Strategy

¿Cómo desplegarías a producción?

Respuesta:

```yaml
deploy-prod:
  needs: [tests, security, docker-build]
  if: github.ref == 'refs/heads/main'
  steps:
    - name: Push to registry
      run: docker push ghcr.io/${{ github.repository }}:${{ github.sha }}

    - name: Update K8s deployment
      run: |
        kubectl set image deployment/bankchurn \
          bankchurn=ghcr.io/${{ github.repository }}:${{ github.sha }}
```

Estrategia: Rolling update con readiness probes.

Preguntas 93-100: Infraestructura (continuación)

Pregunta 93: Resource Requests vs Limits

Explica requests vs limits en K8s.

Respuesta:

```yaml
resources:
  requests:
    memory: "512Mi"  # Garantizado
    cpu: "250m"      # Scheduler usa esto para placement
  limits:
    memory: "1Gi"    # Máximo (OOMKill si excede)
    cpu: "1000m"     # Throttling si excede
```

Best practice: - `requests`: Uso típico (P50) - `limits`: Uso pico aceptable (P99)

Pregunta 94: Prometheus Scraping

¿Cómo configuras Prometheus para scraping?

Respuesta:

```yaml
# prometheus-config.yaml
scrape_configs:
  - job_name: 'bankchurn-predictor'
    kubernetes_sd_configs:
    - role: pod
      namespaces:
        names: [ml-portfolio]
    relabel_configs:
    - source_labels: [__meta_kubernetes_pod_label_app]
      action: keep
      regex: bankchurn-predictor
```

Annotations en Deployment:

```yaml
annotations:
  prometheus.io/scrape: "true"
  prometheus.io/port: "8000"
  prometheus.io/path: "/metrics"
```

Pregunta 95: ConfigMap Usage

¿Cuándo usas ConfigMap vs Secret?

Respuesta:

Dato	Usar
MODEL_VERSION	ConfigMap
LOG_LEVEL	ConfigMap
API_KEY	Secret
DB_PASSWORD	Secret

```yaml
# ConfigMap
apiVersion: v1
kind: ConfigMap
data:
  MODEL_VERSION: "v2.0.0"
  LOG_LEVEL: "INFO"
---
# Montaje en pod
envFrom:
  - configMapRef:
      name: bankchurn-config
```

Pregunta 96: Rolling Update Strategy

Explica tu estrategia de actualización.

Respuesta:

```yaml
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1  # Máximo 1 pod down
      maxSurge: 1        # Máximo 1 pod extra temporal
```

Flujo con 3 replicas: 1. Crear 1 nuevo pod (4 total) 2. Cuando nuevo está Ready, terminar 1 viejo (3 total) 3. Repetir hast todos actualizados

Zero downtime con readiness probes correctos.

Pregunta 97: Volume Mounts para Modelos

¿Cómo montas modelos en K8s?

Respuesta:

```yaml
spec:
  containers:
  - name: bankchurn-api
    volumeMounts:
    - name: models
      mountPath: /app/models
      readOnly: true
  volumes:
  - name: models
    persistentVolumeClaim:
      claimName: ml-models-pvc
```

Alternativas: - **PVC**: Modelos compartidos entre pods - **S3/GCS**: Descarga al inicio - **ConfigMap**: Solo para config pequeños

---

Pregunta 98: Ingress Configuration

¿Cómo expones servicios externamente?

Respuesta:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ml-portfolio-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: bankchurn.example.com
    http:
      paths:
      - path: /
        backend:
          service:
            name: bankchurn-service
            port:
              number: 8000
```

Pregunta 99: Terraform Overview

¿Qué infraestructura defines con Terraform?

Respuesta:

```
infra/terraform/
├── aws/
│   ├── main.tf      # EKS cluster, S3, RDS
│   ├── variables.tf
│   └── outputs.tf
└── gcp/
    ├── main.tf      # GKE, GCS, CloudSQL
    └── ...
```

Recursos típicos: - Container registry (ECR/GCR) - Kubernetes cluster (EKS/GKE) - Object storage (S3/GCS) para modelo - Database (RDS/CloudSQL) para MLflow

---

Pregunta 100: Disaster Recovery

¿Cómo manejas DR para ML systems?

Respuesta:

1. Model artifacts: - S3 versioning + cross-region replication - Git LFS / DVC como backup secundario

2. MLflow database: - RDS multi-AZ + daily snapshots - Point-in-time recovery

3. Kubernetes: - Declarative configs en Git (GitOps) - Multi-region deployment para HA

RTO/RPO objetivos: - RTO (Recovery Time): < 1 hora - RPO (Recovery Point): < 1 día de experimentos

---

Preguntas 102-105: Ética (continuación)

Pregunta 102: Bias Detection

¿Cómo detectas bias en producción?

Respuesta:

```
        # Monitoreo continuo
        for segment in ["Geography", "Gender", "Age_bucket"]:
            pred_rate = predictions.groupby(segment)["churn_pred"].mean()
            log_metric(f"pred_rate_{segment}", pred_rate.to_dict())

            # Alerta si disparate impact < 0.8
            di = pred_rate.min() / pred_rate.max()
            if di < 0.8:
                alert(f"Potential bias in {segment}: DI={di}")
```

Pregunta 103: Model Card Maintenance

¿Con qué frecuencia actualizas Model Cards?

Respuesta:

Actualizar cuando: - Nueva versión del modelo - Cambio en datos de entrenamiento - Descubrimiento de limitación/bias Cambio en uso previsto

Versionado: Model Card versión = Model versión (`v1.0.0`).

Pregunta 104: Human-in-the-Loop

¿Cómo integras supervisión humana?

Respuesta:

BankChurn: - Modelo sugiere clientes en riesgo - Equipo de retención revisa lista - Decisión final es humana (llamar o no)

No automatizar: Ofertas, descuentos, cierre de cuentas.

Pregunta 105: GDPR Compliance

¿Cómo manejas datos personales?

Respuesta:

1. **Minimización**: Solo features necesarias (no nombre, email)
2. **Pseudonimización**: CustomerID sin mapeo a identidad real
3. **Right to erasure**: Pipeline para eliminar datos de un cliente
4. **Audit trail**: Logs de predicciones (sin PII) para auditoría

---

Preguntas 107-115: Liderazgo (continuación)

Pregunta 107: Team Communication

¿Cómo comunicas decisiones técnicas al equipo?

Respuesta:

1. **ADRs (Architecture Decision Records)**: Documentar why, not just what
2. **Tech talks**: Sesiones de 30 min sobre decisiones importantes
3. **PR descriptions**: Contexto suficiente para reviewers
4. **Diagrams**: Mermaid/Lucidchart para arquitectura

Pregunta 108: Mentoring Junior Engineers

¿Cómo mentoras a juniors en ML?

Respuesta:

1. **Pair programming**: En primeros PRs de ML
2. **Code review detallado**: Explicar el “por qué”
3. **Recursos curados**: Pointing to best practices

4. **Proyectos graduales**: Simple → Complejo

Errores comunes a prevenir: - Data leakage (el más crítico) - Overfitting sin validación - Métricas incorrectas para e problema

---

Pregunta 109: Stakeholder Management

¿Cómo manejas expectativas de stakeholders?

Respuesta:

1. **Baseline comparison**: "El modelo mejora X% sobre regla actual"
2. **Confidence intervals**: "Precisión entre 75-85%"
3. **Limitations explícitas**: "No funciona bien para casos X"
4. **Iterative delivery**: MVP → Mejoras incrementales

Evitar: Prometer 99% accuracy, plazos imposibles, omitir limitaciones.

Pregunta 110: Technical Debt Negotiation

¿Cómo negocias tiempo para tech debt?

Respuesta:

Argumentos efectivos: 1. **Riesgo cuantificado**: "Sin tests, bugs llegan a prod" 2. **Velocity impact**: "Refactor ahor ahorra X horas/semana" 3. **Costo de delay**: "Cada mes aumenta esfuerzo 20%"

Estrategia: 20% del sprint para tech debt (negociado upfront).

Pregunta 111: Production Incident Response

¿Cómo manejas un incidente en producción?

Respuesta:

Playbook: 1. **Detect**: Alertas de Prometheus/PagerDuty 2. **Triage**: Severity assessment (P1-P4) 3. **Communicate**: Statu page update 4. **Mitigate**: Rollback if needed 5. **Fix**: Root cause resolution 6. **Postmortem**: Blameless análisis

Para ML específico: Rollback = deploy versión anterior del modelo.

Pregunta 112: Cross-functional Collaboration

¿Cómo trabajas con Data Scientists vs ML Engineers?

Respuesta:

Rol	Responsabilidad
Data Scientist	Exploración, feature engineering, model selection
ML Engineer	Productionization, CI/CD, monitoring
Overlap	Evaluación, experiments

Handoff: DS entrega notebook + requirements, MLE convierte a pipeline.

Pregunta 113: Prioritization Framework

¿Cómo priorizas features de ML?

Respuesta:

RICE Score: - **R**each: ¿Cuántos usuarios afecta? - **I**mpact: ¿Cuánto mejora métricas? - **C**onfidence: ¿Qué tan seguro estamos? - **E**ffort: ¿Cuánto trabajo requiere?

Score = (R × I × C) / E

Pregunta 114: Remote Team Leadership

¿Cómo lideras equipos remotos?

Respuesta:

1. **Async by default**: Documentación > meetings
2. **Overlap hours**: 2-3 horas para sync
3. **Clear ownership**: Cada task tiene responsable
4. **Over-communication**: Status updates frecuentes
5. **Trust + accountability**: Medir outcomes, no horas

Pregunta 115: Career Growth Path

¿Cómo defines el growth path para ML Engineers?

Respuesta:

```
Junior ML Engineer
  ↓ (1-2 años)
ML Engineer
  ↓ (2-3 años)
Senior ML Engineer
  ↓ (2-4 años)
  ├─→ Staff ML Engineer (IC track)
  └─→ ML Engineering Manager (Management track)
```

Skills por nivel: - **Junior**: Implementar pipelines existentes - **Mid**: Diseñar nuevos pipelines - **Senior**: Arquitectura end-to-end, mentoring - **Staff**: Cross-team influence, technical vision

Resumen y Recursos

Skills Demostrados en Este Portafolio

Área	Evidencia
ML Fundamentals	3 proyectos: clasificación, regresión, ensemble
MLOps	MLflow, DVC, CI/CD, monitoring
Software Engineering	Pydantic, tests, modular design
DevOps	Docker, K8s, Terraform
Leadership	Documentation, decisions, trade-offs

Preparación Adicional Recomendada

1. **System Design**: Diseñar ML system de principio a fin
2. **Coding Interview**: LeetCode medium (estructuras de datos)
3. **Behavioral**: STAR method para experiencias pasadas
4. **Deep Dive**: Estar listo para explicar CUALQUIER línea de código

---

Fin del Simulacro de Entrevista

Generado basado en análisis exhaustivo del portafolio ML-MLOps