Test: Machine Learning Bancaire Avancé

Niveau: Avancé | Durée: 40 minutes | 20 Questions

Section A: Risque de Crédit (8 questions)

Question 1

Vous développez un modèle PD pour un nouveau produit de prêt. Vous n'avez que 500 défauts sur 50,000 prêts. Quelle approche adoptez-vous?

Voir la réponse

Défis: - Taux de défaut = 1% (déséquilibre) - 500 défauts = peu pour un modèle robuste

Approche:

1. **Données:**
   - Combiner avec des produits similaires si possible
   - Utiliser des données externes (bureau de crédit)
2. **Modélisation:**

```
# Stratified split pour préserver le ratio
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
)

# SMOTE sur train seulement
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Modèle avec class weights
model = LogisticRegression(class_weight='balanced')
```

3. **Validation:**
   - Cross-validation stratifiée
   - Bootstrap pour estimer l'incertitude
   - Out-of-time validation
4. **Prudence:**
   - Calibrer les PD de manière conservative
   - Documenter les limitations

Question 2

Expliquez la différence entre validation discriminante et calibration d'un modèle de scoring:

Voir la réponse

Discrimination: - Capacité à **ordonner** les clients par risque - Les mauvais clients ont-ils des scores plus bas? - Métriques: Gini, AUC, KS

Calibration: - Les probabilités prédites correspondent-elles à la réalité? - Si PD = 5%, observe-t-on ~5% de défauts? - Métriques: Brier score, courbe de calibration

Exemple:

```
Modèle A: Gini = 0.50, mais PD moyenne = 10% quand défaut réel = 5%
→ Bonne discrimination, mauvaise calibration

Modèle B: Gini = 0.40, PD moyenne = 5%, défaut réel = 5%
→ Discrimination moyenne, bonne calibration
```

Les deux sont nécessaires: - Discrimination pour le ranking - Calibration pour le pricing et les provisions

---

Question 3

Comment valider un modèle LGD (Loss Given Default)?

Voir la réponse

Données requises: - Uniquement les cas de défaut (population restreinte) - LGD réalisé = Perte / Exposition au défaut

Métriques:

```python
# Régression
mae = mean_absolute_error(lgd_reel, lgd_predit)
rmse = np.sqrt(mean_squared_error(lgd_reel, lgd_predit))
r2 = r2_score(lgd_reel, lgd_predit)
```

Validations spécifiques:

1. **Par type de garantie:**

```python
df.groupby('type_garantie').apply(
    lambda x: mean_absolute_error(x['lgd_reel'], x['lgd_predit'])
)
```

2. **Par tranche d'exposition:**

```python
df.groupby('tranche_ead').apply(
    lambda x: x['lgd_predit'].mean() - x['lgd_reel'].mean()
)
```

3. **Downturn LGD:**
   - LGD en période de stress économique
   - Requis par Basel pour le capital

Défis: - Peu de données (que les défauts) - LGD varie beaucoup selon les garanties - Processus de recouvrement long (données incomplètes)

Question 4

Qu'est-ce que le "development sample" vs "validation sample" vs "out-of-time sample"?

Voir la réponse

Development Sample: - Données utilisées pour construire le modèle - Split en train/test pendant le développement - Période: ex. 2018-2020

Validation Sample: - Données de la même période mais non utilisées - Pour validation indépendante - Période: ex. 2018-2020 (échantillon distinct)

Out-of-Time Sample: - Données d'une période postérieure - Test de stabilité temporelle - Période: ex. 2021-2022

```
Timeline:
2018 -------- 2020 -------- 2022
   Development      Out-of-Time

Development Sample:
[Train: 70%] [Test: 30%]
```

Importance: - Out-of-time est crucial car il teste si le modèle reste valide dans le temps - C'est le test le plus réaliste

---

Question 5

Un régulateur demande de démontrer que votre modèle n'est pas discriminatoire. Comment procédez-vous?

Voir la réponse

Fairness Testing:

1. **Variables sensibles à exclure:**
   - Genre, ethnie, religion, handicap
   - Variables proxy (code postal dans certains cas)
2. **Tests de disparité:**

```python
# Taux d'approbation par groupe
approval_by_gender = df.groupby('gender')['approved'].mean()
disparate_impact = approval_by_gender.min() / approval_by_gender.max()
# Règle des 80%: ratio > 0.8 acceptable
```

3. **Analyse des coefficients:**

```python
# Vérifier que les variables sensibles n'ont pas d'impact
# Ou que les proxies sont contrôlés
```

4. **Tests statistiques:**

```python
# Chi-carré sur approbation × variable sensible
from scipy.stats import chi2_contingency
table = pd.crosstab(df['gender'], df['approved'])
chi2, p_value, _, _ = chi2_contingency(table)
```

5. **Documentation:**
   - Processus de sélection des variables
   - Tests de fairness effectués
   - Justification des variables utilisées

Question 6

Calculez l'Expected Loss d'un portefeuille avec les données suivantes:

Segment	Exposition (M HTG)	PD	LGD
Prime	500	1%	30%
Standard	300	5%	45%
Subprime	100	15%	60%

Voir la réponse

Formule: EL = PD × LGD × EAD

Calcul par segment:

Segment	EAD	PD	LGD	EL
Prime	500M	1%	30%	500 × 0.01 × 0.30 = **1.5M**
Standard	300M	5%	45%	300 × 0.05 × 0.45 = **6.75M**
Subprime	100M	15%	60%	100 × 0.15 × 0.60 = **9.0M**

Total EL = 1.5 + 6.75 + 9.0 = 17.25M HTG

EL Ratio = 17.25 / 900 = 1.92%

Observations: - Subprime représente 11% de l'exposition mais 52% des pertes attendues - Concentration du risque sur les segments faibles

---

Question 7

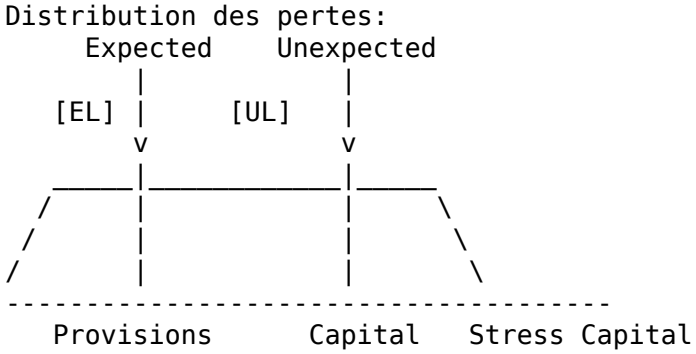
Différence entre Expected Loss et Unexpected Loss?

Voir la réponse

Expected Loss (EL): - Perte **moyenne** attendue - Couvert par les **provisions** et le pricing - Calculable: EL = PD × LGD × EAD

Unexpected Loss (UL): - Pertes au-delà de la moyenne - Volatilité autour de l'EL - Couvert par le **capital** - Calculé avec la VaR du crédit

Illustration:



Relation: - Capital réglementaire = UL × facteur de sécurité - Basel calcule le capital pour couvrir 99.9% des pertes

---

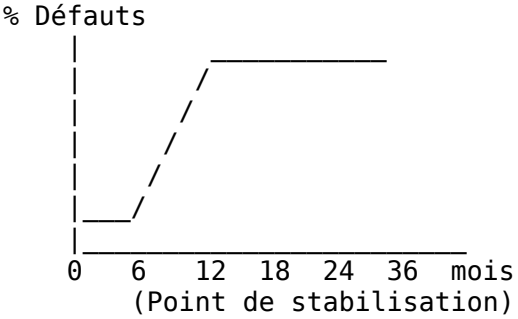
Question 8

Qu'est-ce que la maturation d'un portefeuille de prêts? Impact sur la modélisation?

Voir la réponse

Définition: La maturation est le temps nécessaire pour observer les défauts après l'octroi.

Courbe de maturation typique:



Impact sur la modélisation:

1. **Échantillon de développement:**
   - Prêts suffisamment maturés (12-24 mois)
   - Exclusion des prêts trop récents
2. **Définition du défaut:**
   - Horizon typique: 12 mois
   - Cohérent avec la maturation
3. **Biais potentiels:**
   - Prêts récents sous-estiment le défaut
   - Cohortes anciennes peuvent sur-estimer
4. **Performance window:**

```
# Exclure les prêts non maturés
df_model = df[df['mois_depuis_octroi'] >= 12]
```

Section B: Détection de Fraude et Anomalies (6 questions)

Question 9

Concevez le feature engineering pour un modèle de fraude carte bancaire:

Voir la réponse

Features Transactionnelles:

```
# Montant
df['montant_vs_moyenne'] = df['montant'] / df['montant_moyen_client']
df['montant_zscore'] = (df['montant'] - df['montant_mean']) / df['montant_std']

# Fréquence
df['nb_tx_1h'] = df.groupby(['card_id']).rolling('1H')['tx_id'].count()
df['nb_tx_24h'] = df.groupby(['card_id']).rolling('24H')['tx_id'].count()

# Vélocité
df['temps_depuis_derniere_tx'] = df.groupby('card_id')['timestamp'].diff().dt.seconds
```

Features Géographiques:

```
# Distance
df['distance_derniere_tx'] = haversine(df['lat'], df['lon'],
                                       df['lat_prev'], df['lon_prev'])
df['vitesse_impossible'] = df['distance'] / df['temps'] > 500  # km/h

# Pays
df['pays_risque'] = df['pays'].isin(PAYS_RISQUE)
df['changement_pays'] = df['pays'] != df['pays_prev']
```

Features Temporelles:

```
df['heure'] = df['timestamp'].dt.hour
df['est_nuit'] = df['heure'].between(0, 6)
df['est_weekend'] = df['timestamp'].dt.dayofweek >= 5
```

Features Comportementales:

```
df['merchant_nouveau'] = ~df['merchant_id'].isin(df['merchants_habituels'])
df['categorie_inhabituelle'] = df['mcc'] != df['mcc_principal']
```

Question 10

Expliquez le concept de “concept drift” et son impact sur un modèle de fraude:

Voir la réponse

Définition: Le concept drift se produit quand la relation entre les features et la cible change dans le temps.

Types de drift:

1. **Sudden drift:** Changement brusque (ex: nouveau type de fraude)
2. **Gradual drift:** Évolution lente (ex: adoption du paiement mobile)
3. **Recurring drift:** Patterns saisonniers

Impact sur la fraude: - Les fraudeurs s’adaptent constamment - Nouvelles techniques = nouveaux patterns - Le modèle devient obsolète rapidement

Détection:

```
# Monitoring du PSI
for feature in features:
    psi = calculate_psi(train_data[feature], production_data[feature])
    if psi > 0.25:
        print(f"Drift détecté: {feature}")

# Performance rolling
rolling_precision = df.groupby('semaine').apply(
    lambda x: precision_score(x['fraude_reelle'], x['fraude_predite'])
)
```

Solutions: 1. **Retraining régulier** (mensuel/trimestriel) 2. **Online learning** (mise à jour continue) 3. **Ensemble de modèles** par période 4. **Alertes de drift** automatiques

---

Question 11

Comment gérer les alertes en cascade dans un système de fraude (une fraude = plusieurs alertes)?

Voir la réponse

Problème: - Une carte compromise = plusieurs transactions frauduleuses - Chaque transaction génère une alerte - Surcharge des équipes d'investigation

Solutions:

1. **Agrégation par carte/session:**

```
# Grouper les alertes par carte sur 24h
alerts_grouped = alerts.groupby(['card_id', pd.Grouper(key='timestamp', freq='24H')])
alerts_dedup = alerts_grouped.first()
```

2. **Scoring de priorité:**

```
# Score = proba × montant × urgence
alerts['priority'] = (
    alerts['fraud_proba'] *
    np.log1p(alerts['montant']) *
    alerts['urgency_factor']
)
```

3. **Décision automatique:**

```
if fraud_proba > 0.9:
    block_card_automatically()
elif fraud_proba > 0.7:
    send_to_priority_queue()
else:
    send_to_standard_queue()
```

4. **Feedback loop:**

```
# Quand un cas est confirmé fraude,
# marquer toutes les transactions liées
df.loc[df['card_id'] == confirmed_card, 'confirmed_fraud'] = True
```

Question 12

Qu'est-ce que l'apprentissage semi-supervisé? Application en fraude?

Voir la réponse

Définition: Combine données labelées (peu nombreuses) et non-labelées (abondantes).

Pourquoi en fraude? - Labels coûteux (investigation manuelle) - Beaucoup de transactions non vérifiées - Nouveaux patterns non encore identifiés

Techniques:

1. **Self-training:**

```
# 1. Entraîner sur les données labelées
model.fit(X_labeled, y_labeled)

# 2. Prédire sur non-labelées
```

```
predictions = model.predict_proba(X_unlabeled)

# 3. Ajouter les prédictions confiantes aux labels
high_confidence = predictions.max(axis=1) > 0.95
X_labeled = np.vstack([X_labeled, X_unlabeled[high_confidence]])
y_labeled = np.hstack([y_labeled, predictions[high_confidence].argmax(axis=1)])

# 4. Ré-entraîner
```

2. **Label propagation:**

```
from sklearn.semi_supervised import LabelSpreading

# -1 pour les non-labelés
model = LabelSpreading()
model.fit(X, y_with_unlabeled)
```

3. **Active learning:**
   - Le modèle demande les labels des cas incertains
   - Maximise l'information par label acquis

Question 13

Comment évaluer un modèle de détection d'anomalies quand on n'a pas de labels?

Voir la réponse

Métriques indirectes:

1. **Pourcentage d'anomalies:**

```
anomaly_rate = (predictions == -1).mean()
# Devrait correspondre à l'attente business (~1-5%)
```

2. **Stabilité:**

```
# Cohérence entre runs
scores1 = model1.decision_function(X)
scores2 = model2.decision_function(X)
correlation = np.corrcoef(scores1, scores2)[0, 1]
```

3. **Cohérence multi-méthodes:**

```
# Consensus entre Isolation Forest et LOF
iso_anomalies = iso_forest.fit_predict(X) == -1
lof_anomalies = lof.fit_predict(X) == -1
consensus = iso_anomalies & lof_anomalies
```

Validation indirecte:

4. **Analyse manuelle:**

```
# Examiner les top anomalies
top_anomalies = df.nsmallest(100, 'anomaly_score')
# Revue manuelle par experts
```

5. **Validation retardée:**

```
# Après quelques mois, vérifier si les anomalies
# correspondaient à de vraies fraudes/problèmes
```

6. **Tests A/B:**

```
# Comparer les résultats d'investigation avec/sans modèle
```

Question 14

Un modèle de fraude a Precision=20% et Recall=95%. Est-ce acceptable?

Voir la réponse

Analyse: - **Recall 95%:** On détecte 95% des fraudes ✓ - **Precision 20%:** Seulement 20% des alertes sont des vraies fraudes

Calcul du coût:

```
# Supposons:
# - 1000 transactions
# - 10 vraies fraudes (1%)
# - Montant moyen fraude: 15,000 HTG
# - Coût vérification: 100 HTG

# Avec le modèle:
TP = 10 * 0.95  # 9.5 fraudes détectées
FN = 10 * 0.05  # 0.5 fraudes manquées
FP = 9.5 / 0.20 - 9.5  # 38 faux positifs (pour 9.5 TP avec precision 20%)

# Coûts
pertes_evitees = TP * 15000  # 142,500 HTG
pertes_subies = FN * 15000   # 7,500 HTG
cout_investigation = (TP + FP) * 100  # 4,750 HTG

benefice_net = pertes_evitees - pertes_subies - cout_investigation
# 142,500 - 7,500 - 4,750 = 130,250 HTG
```

Verdict: OUI, c'est acceptable!

En fraude, le ratio coût fraude / coût investigation est très élevé (~150x). Donc une precision de 20% reste rentable.

Amélioration possible: - Ajuster le seuil pour équilibrer precision/recall selon les ressources d'investigation

---

Section C: Déploiement et Production (6 questions)

Question 15

Quelles sont les étapes pour mettre un modèle ML en production?

Voir la réponse

Pipeline de déploiement:

1. **Préparation du modèle:**

```
# Sauvegarder le modèle
import joblib
```

```
joblib.dump(model, 'model_scoring_v1.pkl')
joblib.dump(scaler, 'scaler_v1.pkl')
```

2. **API de scoring:**

```
from flask import Flask, request, jsonify

app = Flask(__name__)
model = joblib.load('model_scoring_v1.pkl')
scaler = joblib.load('scaler_v1.pkl')

@app.route('/score', methods=['POST'])
def score():
    data = request.json
    X = preprocess(data)
    X_scaled = scaler.transform(X)
    proba = model.predict_proba(X_scaled)[0, 1]
    return jsonify({'PD': proba, 'score': proba_to_score(proba)})
```

3. **Tests:**
   - Tests unitaires
   - Tests d'intégration
   - Tests de charge
4. **Déploiement:**
   - Container Docker
   - Orchestration (Kubernetes)
   - CI/CD pipeline
5. **Monitoring:**
   - Latence des requêtes
   - Volume de scoring
   - Distribution des scores
   - Alertes d'erreur

Question 16

Comment gérer le versioning des modèles?

Voir la réponse

Best practices:

1. **Nomenclature:**

```
model_[type]_[version]_[date].pkl
model_scoring_v2.3_20240115.pkl
```

2. **Métadonnées:**

```
model_metadata = {
    'version': '2.3',
    'training_date': '2024-01-15',
    'features': ['age', 'revenu', ...],
    'performance': {'gini': 0.52, 'ks': 0.45},
    'population': 'demandes_credit',
    'author': 'equipe_risque'
}
```

3. **Registry:**

```
# MLflow
import mlflow
mlflow.log_model(model, "scoring_model")
mlflow.log_metrics({"gini": gini, "ks": ks})
mlflow.log_params({"max_depth": 5, ...})
```

4. **Rollback:**

```
# Pouvoir revenir à une version précédente
model = load_model('model_scoring_v2.2_20231215.pkl')
```

5. **A/B testing:**

```
# Comparer versions en production
if random.random() < 0.1:
    score = model_challenger.predict(X)
else:
    score = model_champion.predict(X)
```

Question 17

Comment assurer la reproductibilité d'un modèle ML?

Voir la réponse

Éléments à versionner:

1. **Code:**

```
# Git pour le code
git commit -m "Model training v2.3"
git tag model_v2.3
```

2. **Données:**

```
# Hash des données d'entraînement
import hashlib
data_hash = hashlib.md5(df.to_csv().encode()).hexdigest()
```

3. **Environnement:**

```
# requirements.txt
scikit-learn==1.3.0
pandas==2.0.0
numpy==1.24.0
```

4. **Random seeds:**

```
import random
import numpy as np

SEED = 42
random.seed(SEED)
np.random.seed(SEED)

# Dans les modèles
model = RandomForestClassifier(random_state=SEED)
train_test_split(..., random_state=SEED)
```

5. **Pipeline complet:**

```python
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LogisticRegression(random_state=42))
])

# Sauvegarder le pipeline entier
joblib.dump(pipeline, 'full_pipeline.pkl')
```

Question 18

Quels KPIs business suivre pour un modèle de scoring en production?

Voir la réponse

KPIs Opérationnels:

KPI	Fréquence	Seuil Alerte
Taux d'approbation	Quotidien	± 5% vs moyenne
Score moyen	Quotidien	± 10 points
Temps de réponse	Temps réel	> 200ms
Taux d'erreur API	Temps réel	> 0.1%

KPIs de Performance:

KPI	Fréquence	Seuil Alerte
Gini/KS	Trimestriel	Baisse > 5 pts
PSI du score	Mensuel	> 0.25
Défaut prévu vs réel	Trimestriel	Écart > 20%

KPIs Business:

KPI	Calcul
Pertes évitées	Défauts détectés × montant moyen
Coût du risque	Provisions / Encours
ROI du modèle	(Pertes évitées - Coût modèle) / Coût modèle

Dashboard exemple:

```python
dashboard = {
    'today': {
        'applications': 150,
        'approved': 85,
        'approval_rate': 56.7,
        'avg_score': 645
    },
```

```
    'alerts': {
        'score_drift': False,
        'volume_anomaly': False
    },
    'performance': {
        'current_gini': 0.48,
        'baseline_gini': 0.50
    }
}
```

Question 19

Comment expliquer à la direction que le modèle actuel doit être remplacé?

Voir la réponse

Arguments techniques:

1. **Performance dégradée:**

```
"Le Gini est passé de 52% à 42% en 18 mois,
indiquant une perte de pouvoir discriminant."
```

2. **Drift des données:**

```
"Le PSI du score est à 0.35, indiquant un changement
significatif dans la population."
```

3. **Calibration défaillante:**

```
"Nous prédisons 3% de défaut mais observons 5%,
soit un écart de 67%."
```

Arguments business:

4. **Impact financier:**

```
"Les pertes ont augmenté de 2M HTG vs prévisions,
directement lié à la dégradation du modèle."
```

5. **Coût d'opportunité:**

```
"Un nouveau modèle pourrait réduire les pertes
de 15%, soit 3M HTG par an."
```

6. **Risque réglementaire:**

```
"Le régulateur exige une validation annuelle.
Un modèle sous-performant peut entraîner des sanctions."
```

Format recommandé: - Executive summary (1 page) - Graphiques comparatifs - Recommandation claire avec budget - Timeline de remplacement

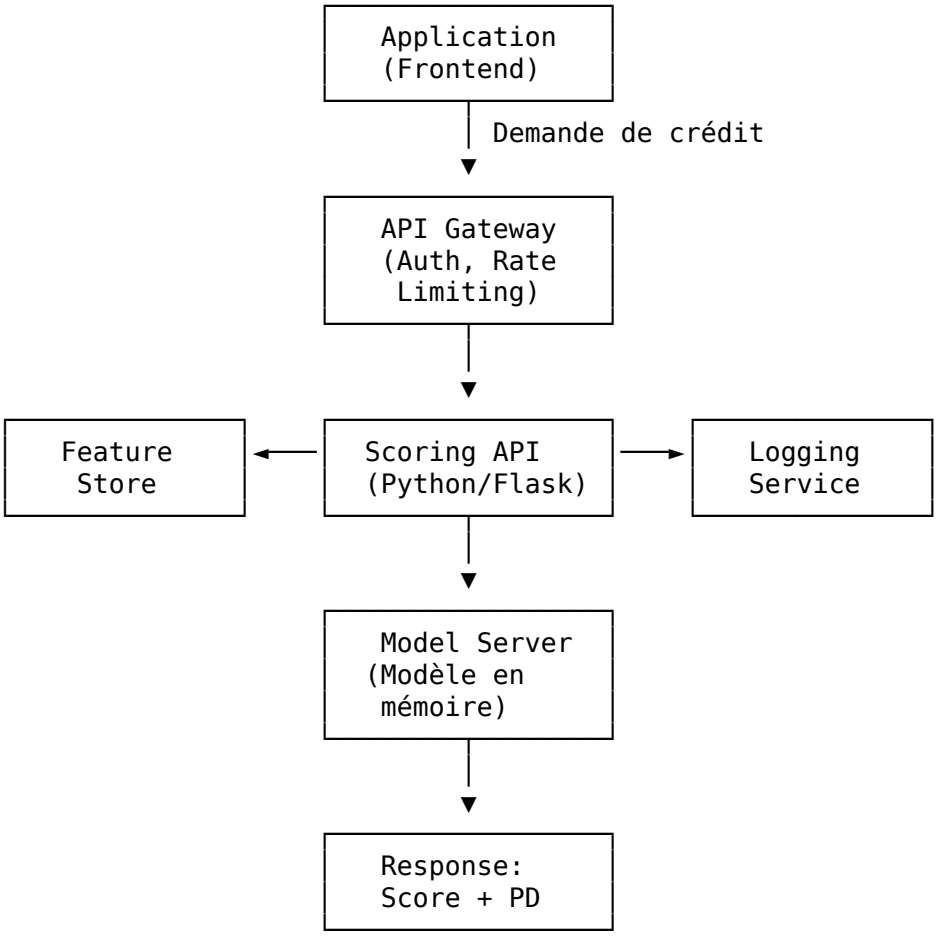
---

Question 20

Décrivez l'architecture d'un système de scoring en temps réel:

Voir la réponse

Architecture:



Composants clés:

1. **Feature Store:**
   - Features pré-calculées par client
   - Latence < 10ms
2. **Model Server:**
   - Modèle chargé en mémoire
   - Plusieurs instances pour la charge
3. **Caching:**
   - Cache des features fréquentes
   - Cache des scores récents
4. **Monitoring:**
   - Latence par endpoint
   - Distribution des scores
   - Alertes automatiques

Contraintes: - Latence totale < 100ms - Disponibilité 99.9% - Scalabilité horizontale

---

Barème

Section	Points
Section A (Q1-8)	35 points
Section B (Q9-14)	35 points
Section C (Q15-20)	30 points
Total	**100 points**

Formules et Mnémotechniques

EL = PLD (Perte Liée au Défaut)

```
Expected Loss = PD × LGD × EAD
```

Gini = GAIN

```
Gini = 2 × AUC - 1
Gini Allows INterpretation of discrimination
```

CRISP-DM pour tout projet ML: - **C**ompréhension business - **R**écupération des données - **I**nvestigation (EDA) - **S**tructuration (Feature engineering) - **P**révision (Modélisation) - **D**éploiement - **M**onitoring

“Train-Transform-Test”: - FIT sur Train - TRANSFORM sur Train ET Test - Jamais FIT sur Test