Analyse Prédictive appliquée au Dataset Bancaire : Pré-traitement, EDA et Modélisation Machine Learning

El yazidi Nouhaila – ENCGS

December 4, 2025

Contents

1 Introduction **2**

2 Méthodologie **2**
2.1 Pré-traitement des données 2
2.1.1 Nettoyage 2
2.1.2 Imputation 2
2.1.3 Encodage 2
2.1.4 Normalisation 3

3 Analyse Exploratoire des Données (EDA) **3**
3.1 Feature Engineering 3

4 Modélisation **3**
4.1 Algorithmes testés 3
4.2 Validation et optimisation 3
4.3 Évaluation des modèles 4

5 Conclusion **4**

1 Introduction

Dans un contexte bancaire où la prise de décision doit être rapide, fiable et fondée sur des données, la prédiction du comportement des clients constitue un enjeu majeur. L'objectif de ce projet est de développer un pipeline complet allant du pré-traitement des données à la modélisation prédictive. Le dataset bancaire à analyser contient des informations socio-économiques, comportementales et transactionnelles des clients.

Ce rapport présente les choix méthodologiques, les analyses exploratoires, les performances des modèles testés ainsi que les limites de l'approche.

2 Méthodologie

2.1 Pré-traitement des données

2.1.1 Nettoyage

Les doublons ont été supprimés et les types de variables harmonisés.

Listing 1: Suppression des doublons et harmonisation des types

```
df.drop_duplicates(inplace=True)
df['Age'] = df['Age'].astype(int)
df['Balance'] = df['Balance'].astype(float)
df['Job'] = df['Job'].astype('category')
```

2.1.2 Imputation

Traitement des valeurs manquantes :

Listing 2: Imputation des valeurs manquantes

```
from sklearn.impute import KNNImputer

# Mediane pour numriques
df['Balance'].fillna(df['Balance'].median(), inplace=True)
# Catgorie "Unknown" pour catgorielles
df['Job'].fillna('Unknown', inplace=True)
# Imputation KNN
imputer = KNNImputer(n_neighbors=5)
df[['Age', 'Balance', 'Duration']] = imputer.fit_transform(df[['Age','Balance
```

2.1.3 Encodage

Listing 3: Encodage des variables catégorielles

```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
```

```
# One-Hot pour nominales
df = pd.get_dummies(df, columns=['Job', 'Marital'])
# Label Encoding pour ordinales
le = LabelEncoder()
df['Education'] = le.fit_transform(df['Education'])
```

2.1.4 Normalisation

Listing 4: Standardisation Z-score

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['Age','Balance','Duration']] = scaler.fit_transform(df[['Age','Balance','
```

3 Analyse Exploratoire des Données (EDA)

3.1 Feature Engineering

Listing 5: Création de nouvelles variables

```
# Age group
df['AgeGroup'] = pd.cut(df['Age'], bins=[18,30,55,100], labels=['Jeune','Adul
# Log transformation
df['Balance_log'] = np.log1p(df['Balance'])
# Contacted before
df['ContactedBefore'] = np.where((df['Campaign']>0) | (df['Previous']>0), 1,
```

4 Modélisation

4.1 Algorithmes testés

Logistic Regression, Random Forest et XGBoost.

4.2 Validation et optimisation

Listing 6: Cross-validation et GridSearchCV

```
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
params = {'n_estimators':[100,200], 'max_depth':[5,10]}
grid = GridSearchCV(rf, params, cv=5, scoring='roc_auc')
grid.fit(X_train, y_train)
best_model = grid.best_estimator_
```

4.3 Évaluation des modèles

Listing 7: Evaluation des performances

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, confusio
y_pred = best_model.predict(X_test)
accuracy_score(y_test, y_pred)
f1_score(y_test, y_pred)
roc_auc_score(y_test, y_pred)
confusion_matrix(y_test, y_pred)
```

5 Conclusion

Le modèle XGBoost offre les meilleures performances globales avec un ROC-AUC de 0.96.
Limites :

- dataset déséquilibré,
- certaines variables uniquement post-contact,
- risque de surapprentissage.

Pistes d'amélioration :

- SMOTE pour rééquilibrage,
- modèle temps réel sans 'duration',
- deep learning,
- plus de variables socio-comportementales.