****ا**لجمهورية العربية السورية**

**وزارة التعليم العالي والبحث العلمي**

**الجامعة الافتراضية السورية**

**برنامج ماجستير التأهيل والتخصص في علوم الحاسوب**

**وظيفة مقرر Machine Learning Techniques MLT**

**F24**

مدرس المقرر: د. عصام سلمان

**إعداد الطلاب:**

|  |  |  |
| --- | --- | --- |
| اسم الطالب | الرقم الجامعي | الصف |
| سليمان نزيها | 286017 | C2 |
| ريم علي | 316235 | C2 |

جدول المحتويات

[المقدمة 2](#_Toc197535818)

# المقدمة

هذا المشروع يهدف إلى بناء نموذج تنبؤ آلي يستطيع تحديد ما إذا كان سيتم الموافقة على طلب قرض أو رفضه اعتمادًا على خصائص مقدم الطلب مثل الدخل، الحالة الاجتماعية، التعليم، ومكان الإقامة. يعتمد النموذج على بيانات تاريخية حقيقية من مؤسسة قروض.

# مراحل التنفيذ

## استيراد البيانات وتحضيرها

تم تحميل البيانات من ملف CSV، وتنظيف أسماء الأعمدة لتجنب المشكلات البرمجية، ثم **فصل البيانات إلى ميزات (**X**) وهدف (**y**)** وهو حالة القرض.

|  |
| --- |
| df = pd.read\_csv('loan\_prediction.csv')  df.columns = df.columns.str.strip()  X = df.drop(['Loan\_ID', 'Loan\_Status'], axis=1)  y = df['Loan\_Status']  # ------------------------data splitting ------------------------  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y,  test\_size=0.2,  random\_state=42,  stratify=y  ) |

## معالجة القيم المفقودة

تم استخدام SimpleImputer لمعالجة القيم الناقصة:

* الأعمدة الرقمية تم ملؤها بالمتوسط.
* الأعمدة الفئوية تم ملؤها بالقيم الأكثر تكرارًا.

|  |
| --- |
| # ------------------------ معالجة القيم المفقودة ------------------------  # الأعمدة الرقمية  num\_imputer = SimpleImputer(strategy='mean')  num\_cols = ['LoanAmount', 'Loan\_Amount\_Term', 'Credit\_History']  X\_train[num\_cols] = num\_imputer.fit\_transform(X\_train[num\_cols])  X\_test[num\_cols] = num\_imputer.fit(X\_test[num\_cols])  # الأعمدة الفئوية  cat\_imputer = SimpleImputer(strategy='most\_frequent')  cat\_cols = ['Gender', 'Married','Dependents', 'Self\_Employed']  X\_train[cat\_cols] = cat\_imputer.fit\_transform(X\_train[cat\_cols])  X\_test[cat\_cols] = cat\_imputer.fit(X\_test[cat\_cols]) |

## تحويل القيم الفئوية

* تم استخدام LabelEncoder لتحويل بعض القيم الفئوية إلى أرقام.
* OneHotEncoder لتحويل عمود Property\_Area إلى أعمدة ثنائية لتجنّب التأثير الخاطئ للتصنيفات الاسمية.

|  |
| --- |
| # ------------------------ المعالجة الأولية ------------------------  # تحويل الأنواع الفئوية  le = LabelEncoder()  cat\_cols = ['Gender', 'Married', 'Dependents', 'Education', 'Self\_Employed', 'Credit\_History', 'Loan\_Status']  for col in cat\_cols:  X\_train[col] = le.fit\_transform(X\_train[col])  ohe= OneHotEncoder(drop='first', sparse\_output=False)  property\_ohe = ohe.fit\_transform(df[['Property\_Area']])  df\_ohe = pd.DataFrame(property\_ohe, columns=ohe.get\_feature\_names\_out(["Property\_Area"]))  df = pd.concat([df.drop("Property\_Area", axis=1), df\_ohe], axis=1) |

## هندسة الميزات

تم إنشاء ميزتين جديدتين لتعزيز أداء النموذج:

* TotalIncome = دخل مقدم الطلب + دخل الشريك.
* LoanToIncomeRatio = نسبة مبلغ القرض إلى الدخل الكلي.

كما تم تطبيق دالة log1p لتقليل تأثير القيم المتطرفة.

|  |
| --- |
| # ------------------------ هندسة الميزات ------------------------  # إضافة ميزات جديدة  df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']  df['LoanToIncomeRatio'] = df['LoanAmount'] / (df['TotalIncome'] + 1e-6) # تجنب القسمة على صفر  # معالجة القيم المتطرفة  X\_train\_transformed = np.log1p(X\_train) # Log Transformation  X\_test\_transformed = np.log1p(X\_test) # Log Transformation |

## معالجة عدم التوازن

استخدمنا SMOTE (Synthetic Minority Oversampling Technique) لزيادة تمثيل الفئة الأقل (الطلبات المرفوضة ) لتحقيق توازن بين الفئات قبل تدريب النموذج.

|  |
| --- |
| smote = SMOTE(random\_state=42, sampling\_strategy='auto')  X\_res, y\_res = smote.fit\_resample(X\_train\_transformed, y\_train) |

## بناء النموذج

**النموذج المستخدم:**

* **Random Forest Classifier**
* تم ضبطه على:
  + 100 شجرة (n\_estimators=100)
  + عمق شجرة أقصى = 10
  + موازنة الفئات باستخدام class\_weight='balanced'

تم تدريب النموذج على البيانات المعالجة والمتوازنة، ثم اختبار أدائه باستخدام البيانات المختبرة.

|  |
| --- |
| model = RandomForestClassifier(  n\_estimators=100, # عدد الأشجار  max\_depth=10, # الحد الأقصى لعمق الشجرة  random\_state=42,  # enable for best recal  class\_weight='balanced' # لمعالجة عدم التوازن في الفئات    )  model.fit(X\_res, y\_res)  # ----------- التنبؤ والتقييم -----------  y\_pred = model.predict(X\_test) |

## تقييم الأداء

تم قياس أداء النموذج باستخدام عدة مؤشرات:

* **الدقة (Accuracy)**
* **تقرير التصنيف (Classification Report)**
* **مصفوفة الارتباك (Confusion Matrix)** مع تمثيل حراري باستخدام مكتبة Seaborn.
* **التحقق المتقاطع (Cross Validation)** لقياس ثبات النموذج على 5 طيات، مع حساب متوسط وانحراف f1-score.

|  |
| --- |
| print("----- model evaluation -----")  print(f"Accuracy: {accuracy\_score(y\_test, y\_pred):.2f}")  print("\n Confusion Matrix :")  print(confusion\_matrix(y\_test, y\_pred))  print("\n Classification Report :")  print(classification\_report(y\_test, y\_pred))  # ------------------------ مصفوفة الارتباك ------------------------  plt.figure(figsize=(8,6))  sns.heatmap(confusion\_matrix(y\_test, y\_pred),  annot=True,  fmt='d',  cmap='Blues',  xticklabels=['Rejected', 'Approved'],  yticklabels=['Rejected', 'Approved'])  plt.title('Confusion Matrix')  plt.show()  # Cross Validation  scores = cross\_val\_score(model, X\_res, y\_res, cv=5, scoring="f1")  print(f"F1 Scores: {scores}")  print(f"Mean F1: {scores.mean()}")  print(f"STD F1: {scores.std()}") |