AI/ML Model Explainability

Contents

Explainable AI

AI Explainability (XAI) refers to techniques that make **machine learning models more transparent and interpretable**. Instead of using black-box predictions, explainability helps us understand *why* a decision was made.

🔹 Why Explainability Matters

- 📜 **Regulation**: Banking and healthcare industries demand transparency.
- 🤝 **Trust**: Customers and regulators want interpretable decisions.
- ⚖️ **Fairness**: Helps detect and correct bias.
- 🛠️ **Debugging**: Easier to identify issues in training and features.

🔷 Key Libraries and Their History

1. **SHAP (SHapley Additive exPlanations)**

- 📅 **Introduced in 2017**.
- 👤 Developed by **Scott Lundberg** (University of Washington, Microsoft Research).
- 📖 Based on **game theory Shapley values (1953)**.
- 🎯 Goal: Provide **both global and local explanations**.
- 📌 Official Docs: https://shap.readthedocs.io/

---

2. **LIME (Local Interpretable Model-Agnostic Explanations)**

- 📅 **Published in 2016** (paper at KDD).
- 👤 Created by **Marco Tulio Ribeiro**, **Sameer Singh**, and **Carlos Guestrin** (University of Washington).
- 🎯 Goal: Explain **individual predictions** using local surrogate models.
- 📌 GitHub: marcotcr/lime

---

3. **Captum (for PyTorch)**

- 📅 **Released in 2019** by **Facebook AI Research (FAIR)**.
- 👤 Main contributors: PyTorch team at Facebook.
- 🎯 Goal: Interpret **deep learning models** (vision, NLP, etc.) with methods like **Integrated Gradients** and **DeepLIFT**.
- 📌 Official Website: https://captum.ai/

---

4. **AIX360 (AI Explainability 360)**

- 📅 **Released in 2019** by **IBM Research**.

- 👤 Developed as part of IBM's **Trusted AI initiative**.
- 🎯 Goal: Provide a comprehensive toolkit for **explainability, fairness, and transparency**.
- 📌 Website: [https://aix360.readthedocs.io/en/latest/](AI Explainability 360)

🔷 Comparative Overview

Tool	Year	Authors/Organization	Best For	Pros	Cons
SHAP	2017	Scott Lundberg (UW, MSR)	Global + Local explanations	Strong theory, trusted	Slower on big data
LIME	2016	Ribeiro, Singh, Guestrin	Local explanations	Easy, model-agnostic	Less stable
Captum	2019	Facebook AI Research	Deep learning (PyTorch)	Powerful for neural nets	PyTorch-only
AIX360	2019	IBM Research	Fairness & compliance	Rich ethical toolkit	Smaller community

---

🔷 Example in Banking: Loan Approval Model

- Train a **loan approval model** with features like Income, Credit History, Loan Amount.
- Use **SHAP** to see which features matter globally.
- Use **LIME** to explain a single applicant's approval or rejection.
- Present explanations in a business-friendly way for **regulators and customers**.

Preprocessing

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, f1_score
from sklearn.preprocessing import LabelEncoder

# Load dataset
df = pd.read_csv("datasets/loan_predication/loan_predication.csv")

df = df.drop(columns=['Loan_ID'])
# Check missing values
print("Missing values per column:\n")
print(df.isnull().sum())

# Encode target with LabelEncoder
le = LabelEncoder()
y = le.fit_transform(df['Loan_Status'])
print("Classes mapping:", le.classes_)
X = df.drop(columns=['Loan_Status'])

# Apply LabelEncoder on all non-numeric columns
for col in X.columns:
    if X[col].dtype == 'object':
        X[col] = LabelEncoder().fit_transform(X[col].astype(str))

# Handle missing values (median for numeric, mode for categorical)
X = X.fillna(X.median(numeric_only=True))
X = X.fillna(X.mode().iloc[0])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
X_train.shape, X_test.shape
```

```
Missing values per column:

Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
Classes mapping: ['N' 'Y']
```

```
((429, 11), (185, 11))
```

Shap

```
import numpy as np
import pandas as pd
import shap
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

▾ RandomForestClassifier

```
RandomForestClassifier(random_state=42)
```

```
# Build SHAP explainer and compute values
shap.initjs()
explainer = shap.TreeExplainer(rf_model)
shap_exp = explainer(X_test)

# Handle multiclass (select positive class = 1)
shap_vals = shap_exp.values
if shap_vals.ndim == 3:
    shap_vals = shap_vals[:, :, 1]

# Robust expected value for binary/multiclass
exp_value = explainer.expected_value
if isinstance(exp_value, (list, tuple, np.ndarray)):
    exp_value = exp_value[1]
```

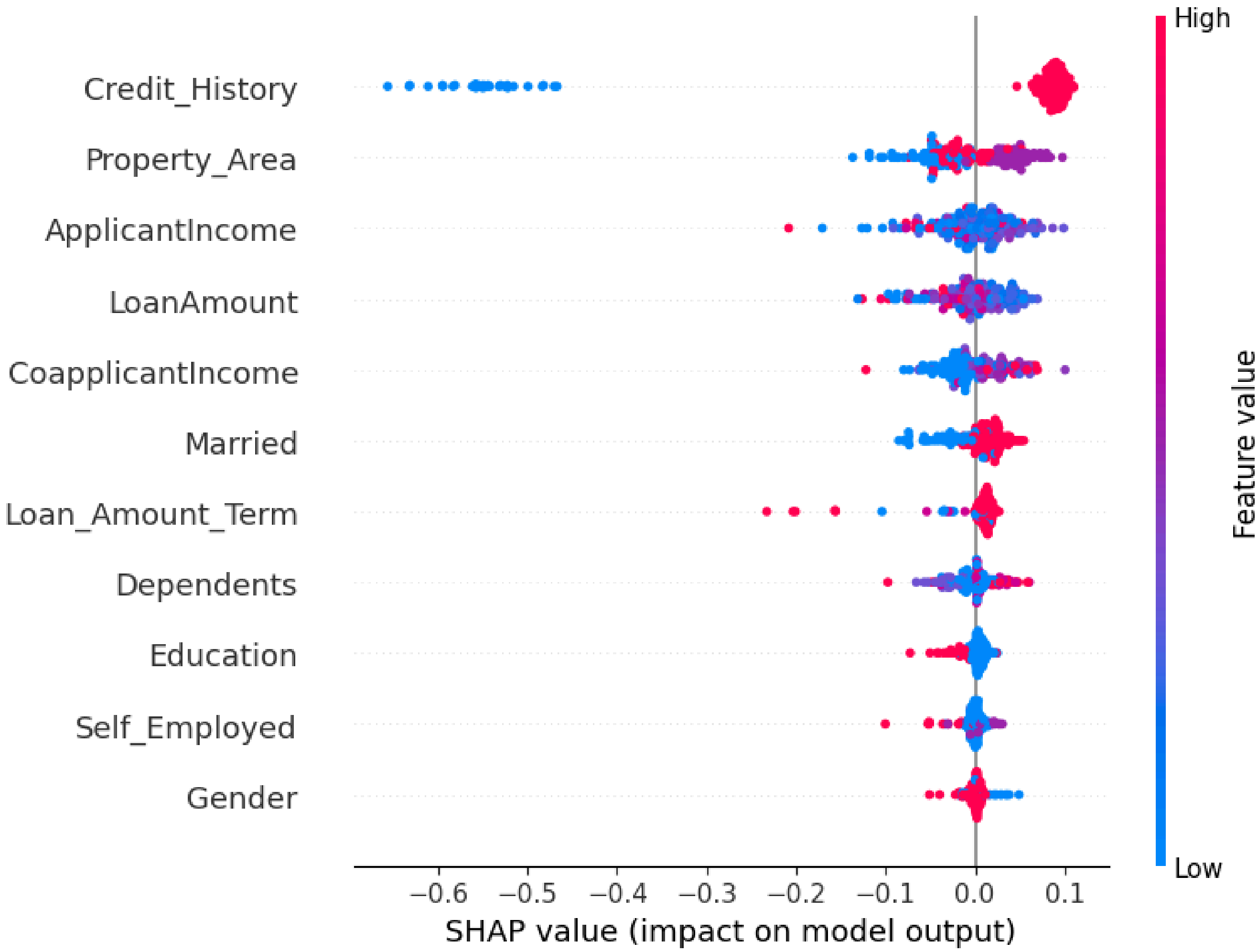
```
# Calculate global feature importance (mean absolute SHAP)
mean_abs_shap = np.abs(shap_vals).mean(axis=0)
feature_importance = pd.DataFrame({
    'Feature': X_test.columns,
    'Mean |SHAP|': mean_abs_shap
})
feature_importance['Percentage'] = 100 * feature_importance['Mean |SHAP|'] / feature_
feature_importance = feature_importance.sort_values(by='Mean |SHAP|', ascending=False
print('## Global Feature Importance (Numeric Values):')
display(feature_importance)
```

```
## Global Feature Importance (Numeric Values):
```

	Feature	Mean |SHAP|	Percentage
0	Credit_History	0.160446	43.730663
1	Property_Area	0.040504	11.039676
2	ApplicantIncome	0.031107	8.478320
3	LoanAmount	0.030088	8.200775
4	CoapplicantIncome	0.026706	7.278916
5	Married	0.021680	5.908994
6	Loan_Amount_Term	0.017710	4.826958
7	Dependents	0.016911	4.609241
8	Education	0.008569	2.335529
9	Self_Employed	0.007123	1.941492
10	Gender	0.006052	1.649437

```
# Global summary plot for all features
print('## Global Feature Importance (Summary Plot):')
shap.summary_plot(shap_vals, X_test)
```

```
## Global Feature Importance (Summary Plot):
```

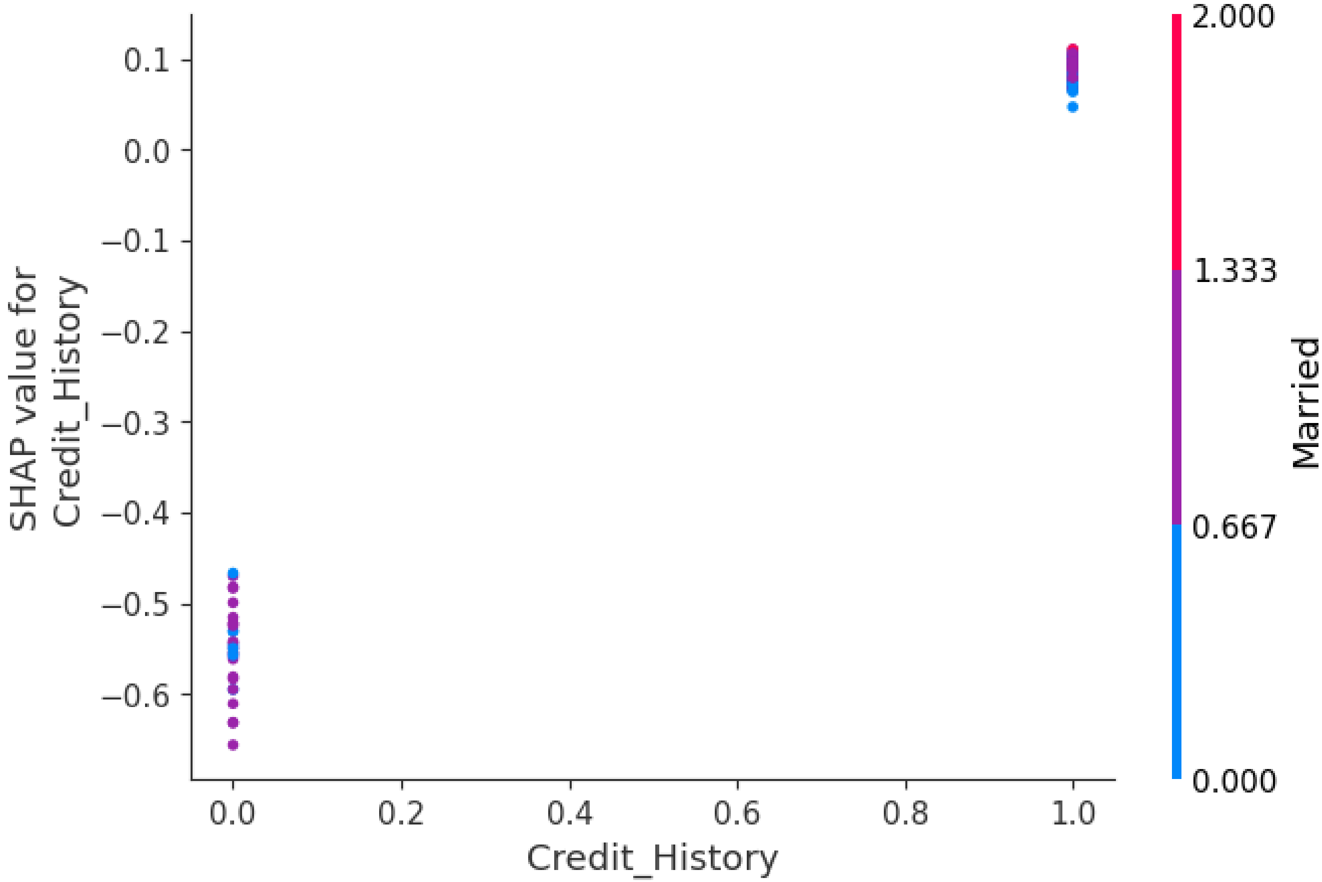


Explanation

The summary plot provides a global overview of feature importance. Positive SHAP values push the prediction toward loan approval, while negative values push it toward rejection. The color represents the actual feature value: blue means low, and red means high. For example, in the case of **Credit_History**, high values (red) strongly increase the chance of approval, whereas low values (blue) are associated with rejection.

```python
# Dependence plot for a selected feature
print('## Dependence Plot (Credit_History):')
shap.dependence_plot('Credit_History', shap_vals, X_test)
```

```
## Dependence Plot (Credit_History):
```



Explanation of Dependence Plot for `Credit_History`

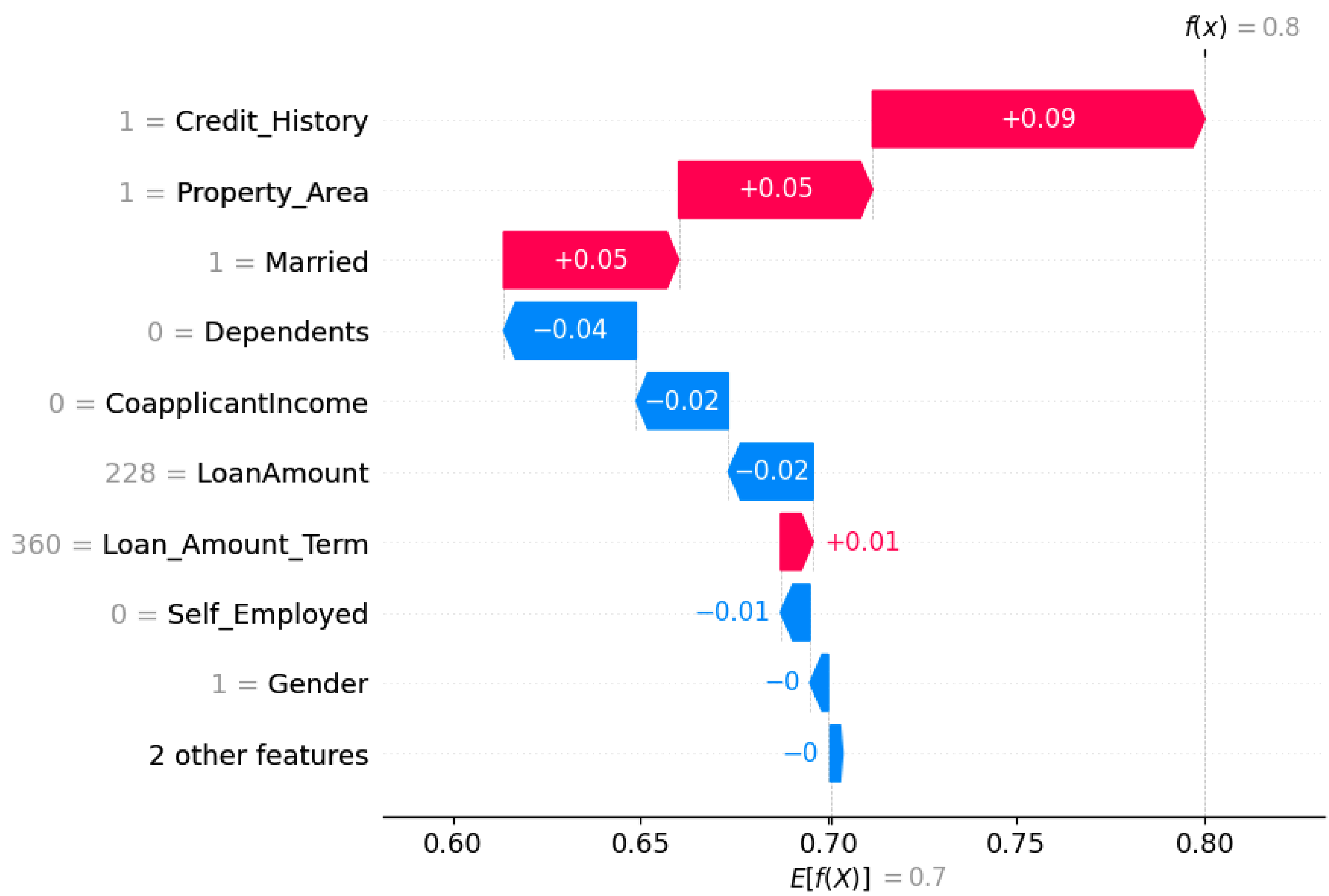
This dependence plot shows how the feature **Credit_History** influences the model predictions. Negative SHAP values reduce the likelihood of loan approval, while positive values increase it.

When `Credit_History` equals 0, all the SHAP values are strongly negative, meaning applicants without a good credit history are much more likely to be rejected. When `Credit_History` equals 1, the SHAP values are positive, showing that a good credit history strongly pushes the model toward approval.

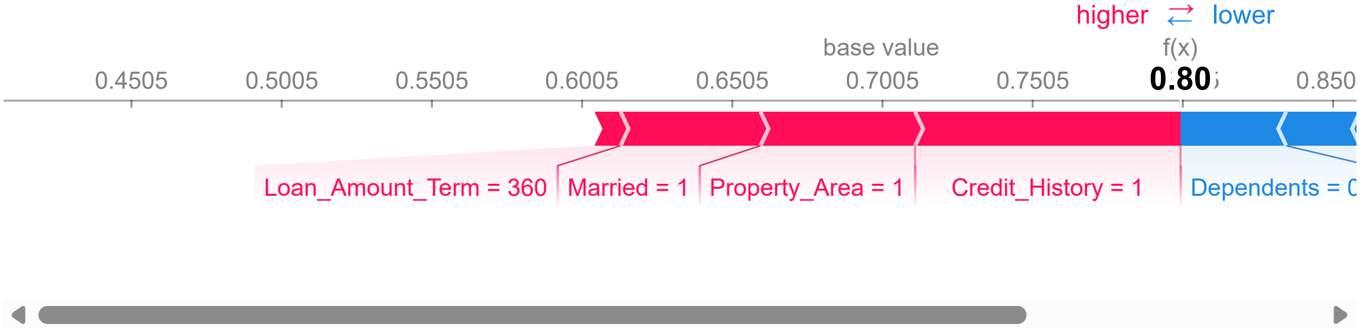
```
# Local explanation for one test sample with model decision
sample_idx = 0  # Change this index to analyze another applicant

# Get prediction from the trained model
sample = X_test.iloc[[sample_idx]]
pred_class = rf_model.predict(sample)[0]
pred_label = le.inverse_transform([pred_class])[0]  # Convert 0/1 back to N/Y
decision = 'APPROVED' if pred_class == 1 else 'REJECTED'

print(f'## Local Explanation for Sample {sample_idx}')
print(f'Model Prediction: {decision} (Original Label = {pred_label})')
```

```
## Local Explanation for Sample 0
Model Prediction: APPROVED (Original Label = Y)
```

```
# Waterfall plot for the same sample
shap.plots.waterfall(
    shap.Explanation(
        values=shap_vals[sample_idx,:],
        base_values=exp_value,
        data=X_test.iloc[sample_idx,:],
        feature_names=X_test.columns
    )
)
```

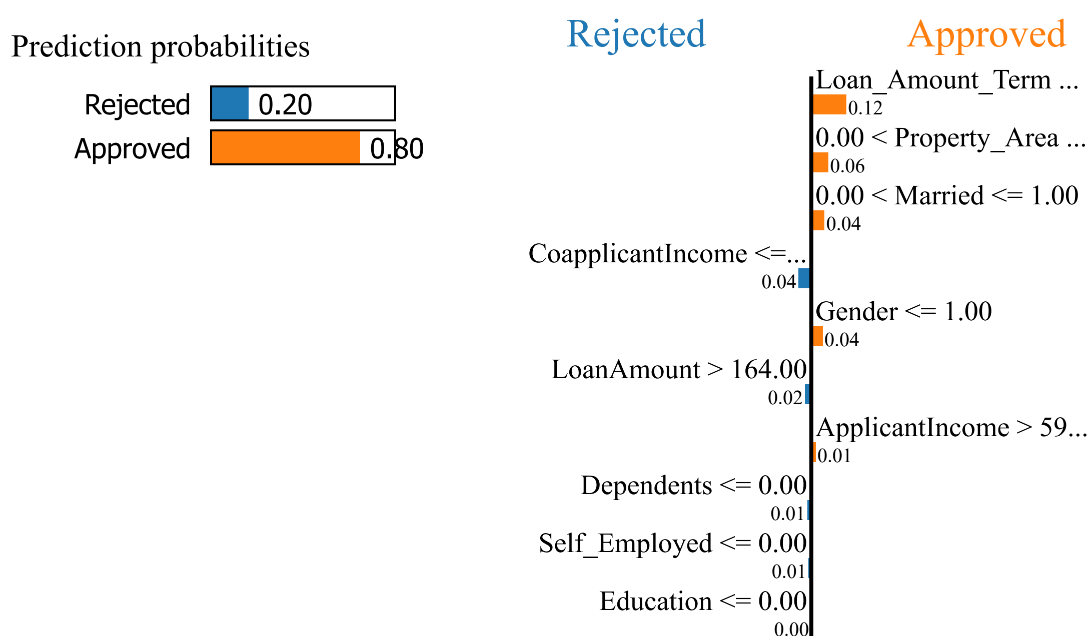



```
# Force plot for the same sample
shap.force_plot(
    exp_value,
    shap_vals[sample_idx,:],
    X_test.iloc[sample_idx,:]
)
```



Lime

```python
import lime
import lime.lime_tabular

# LIME Explainer
lime_explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=np.array(X_train),
    feature_names=X_train.columns,
    class_names=['Rejected','Approved'],
    mode='classification'
)

# Explain first test instance
exp = lime_explainer.explain_instance(
    data_row=X_test.iloc[0],
    predict_fn=rf_model.predict_proba
)
exp.show_in_notebook(show_table=True)
```



Feature	Value
Loan_Amount_Term	360.00
Property_Area	1.00
Married	1.00
CoapplicantIncome	0.00
Gender	1.00
LoanAmount	228.00
ApplicantIncome	9083.00