Bayesian Credit Risk Model — Documentation

This project implements a Bayesian logistic regression model for credit risk prediction. The primary objective is to estimate the probability of loan default based on financial and demographic features of applicants. Unlike traditional machine learning models that provide single-point estimates, Bayesian methods quantify uncertainty in predictions by generating full posterior distributions of parameters.

The workflow consists of several stages. First, the dataset is preprocessed: new features are engineered, categorical variables are encoded, and numerical features are standardized. The model is then defined in a probabilistic framework using PyMC3, where regression coefficients are given hierarchical priors to reflect uncertainty and potential variation. Model fitting is carried out with Markov Chain Monte Carlo (MCMC) sampling, generating posterior distributions of parameters.

Evaluation is performed on a held-out test set using ROC AUC, PR AUC, and a confusion matrix, with visualizations of ROC and Precision–Recall curves. To enhance interpretability, SHAP values are applied to analyze feature contributions, providing insights into how each predictor influences the probability of default. This combination of Bayesian inference and explainability ensures that the model is both accurate and transparent, which is essential for risk-sensitive applications like credit scoring.

Full Explanation of the Code

Importing Libraries

```
import numpy as np

import pandas as pd

import pymc3 as pm

import arviz as az

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.metrics import (roc_auc_score, average_precision_score,

                             confusion_matrix, classification_report)

import shap
```

- **numpy, pandas** → for numerical operations and handling datasets.
- **pymc3** → Bayesian statistical modeling and MCMC sampling.
- **arviz** → analyzing and visualizing Bayesian inference results.

- **matplotlib, seaborn** → for plotting curves and evaluation metrics.
- **scikit-learn tools** → scaling features, splitting data, computing ROC/PR metrics, confusion matrix.
- **shap** → interpretability (explains how features affect predictions).

Reading the Dataset

data = pd.read_csv("credit_data.csv")

Loads a dataset with customer credit information (e.g., age, salary, credit score, loan details).

Model Class Definition

class BayesianCreditRiskModel:

This class wraps everything: preprocessing, building the Bayesian model, training, evaluating, and analyzing features.

Class Initialization

```
def __init__(self, n_chains=4, n_samples=2000, n_tune=1000):
    self.n_chains = n_chains
    self.n_samples = n_samples
    self.n_tune = n_tune
    self.model = None
    self.trace = None
    self.feature_names = None
    self.scaler = StandardScaler()
```

- Sets hyperparameters for sampling (number of chains, samples, tuning steps).
- Stores the model and trace (posterior samples).
- Creates a StandardScaler to normalize features.

Preprocessing

```
def preprocess_data(self, df):
    df['loan_percent_income'] = df['loan_amount'] / (df['salary'] + 1e-6)
```

```
    df = pd.get_dummies(df, columns=['home_ownership'], drop_first=True)
    features = ['age', 'salary', 'credit_score',
                'loan_percent_income', 'years_employed',
                'home_ownership_MORTGAGE', 'home_ownership_OWN']
    X = df[features]
    y = df['loan_status'].values
    self.feature_names = features
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y)
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.transform(X_test)
    return X_train, X_test, y_train, y_test
```

- **Creates new feature**: loan amount as a percentage of income.
- **Encodes categorical variable** home_ownership into dummy variables.
- Selects the main predictors.
- Splits into training/testing (80/20).
- Scales the features (standardization).
- Returns train/test sets.

Building the Model

```
def build_model(self, X, y):
    with pm.Model() as model:
        mu_beta = pm.Normal('mu_beta', mu=0, sigma=1)
        sigma_beta = pm.HalfNormal('sigma_beta', sigma=1)

        beta = pm.Normal('beta', mu=mu_beta, sigma=sigma_beta, shape=X.shape[1])
        alpha = pm.Normal('alpha', mu=0, sigma=10)
        mu = alpha + pm.math.dot(X, beta)
        theta = pm.Deterministic('theta', pm.math.sigmoid(mu))
        y_obs = pm.Bernoulli('y_obs', p=theta, observed=y)
```

```
        self.model = model
    return model
```

- Defines a **Bayesian logistic regression**.
- Priors:
  - mu_beta (mean of betas) ~ Normal(0, 1).
  - sigma_beta (scale of betas) ~ HalfNormal(1).
  - beta (coefficients) ~ Normal(mu_beta, sigma_beta).
  - alpha (intercept) ~ Normal(0, 10).
- Logistic transformation (sigmoid) gives probability of default.
- Likelihood: observed loan status (y_obs).

Fitting the Model

```
def fit_model(self, X, y):
    with self.model:
        self.trace = pm.sample(
            draws=self.n_samples,
            tune=self.n_tune,
            chains=self.n_chains,
            target_accept=0.9,
            return_inferencedata=True)
    return self.trace
```

- Runs **MCMC sampling** to approximate posterior distributions of coefficients.
- Saves posterior samples in self.trace.

Evaluation

```
def evaluate(self, X_test, y_test):
    with self.model:
        ppc = pm.sample_posterior_predictive(
            self.trace,
            var_names=['theta'],
```

```
        random_seed=42)

    pred_probs = ppc['theta'].mean(axis=0)
    roc_auc = roc_auc_score(y_test, pred_probs)
    pr_auc = average_precision_score(y_test, pred_probs)

    print(f"ROC AUC: {roc_auc:.3f}")
    print(f"PR AUC: {pr_auc:.3f}")

    y_pred = (pred_probs > 0.5).astype(int)
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    self.plot_roc_pr(y_test, pred_probs)
    return roc_auc, pr_auc
```

- Runs **posterior predictive checks** (sampling predictions).
- Computes evaluation metrics (ROC AUC, PR AUC).
- Thresholds predictions at 0.5 for confusion matrix.
- Calls a plotting function for ROC/PR curves.

ROC and PR Curves

```
def plot_roc_pr(self, y_true, y_pred):
    from sklearn.metrics import roc_curve, precision_recall_curve

    plt.figure(figsize=(12, 5))

    plt.subplot(121)
    fpr, tpr, _ = roc_curve(y_true, y_pred)
```

```
    plt.plot(fpr, tpr, label=f"ROC (AUC = {roc_auc_score(y_true, y_pred):.2f})")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()

    plt.subplot(122)
    precision, recall, _ = precision_recall_curve(y_true, y_pred)
    plt.plot(recall, precision, label=f"PR (AUC = {average_precision_score(y_true,
y_pred):.2f})")
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Plots both **ROC curve** and **Precision-Recall curve**.

Feature Analysis (SHAP)

```
def analyze_features(self):
    background = self.trace.posterior['beta'].values.reshape(-1, len(self.feature_names))
    explainer = shap.Explainer(
        model=lambda X: 1 / (1 + np.exp(-(X @ background.T))),
        masker=background.mean(axis=0).reshape(1, -1),
        feature_names=self.feature_names)

    shap_values = explainer(background.mean(axis=0).reshape(1, -1))

    plt.figure()
    shap.plots.waterfall(shap_values[0], show=False)
    plt.title("Feature Impact on Prediction")
```

```
plt.tight_layout()
plt.show()
```

- Uses SHAP to interpret coefficients.
- Builds a custom logistic regression predictor from posterior samples.
- Plots a **waterfall plot** showing how features increase or decrease default probability.

Running the Full Pipeline

```
model = BayesianCreditRiskModel()
X_train, X_test, y_train, y_test = model.preprocess_data(data)
model.build_model(X_train, y_train)
trace = model.fit_model(X_train, y_train)

print("\n=== Model Evaluation ===")
roc_auc, pr_auc = model.evaluate(X_test, y_test)

print("\n=== Feature Analysis ===")
model.analyze_features()

print("\n=== Bayesian Summary ===")
print(pm.summary(trace, var_names=['alpha', 'beta']))
```

- Creates the model instance.
- Preprocesses the dataset.
- Builds the Bayesian logistic regression model.
- Fits using MCMC sampling.
- Evaluates test set performance.
- Runs SHAP feature analysis.
- Prints Bayesian summary (posterior means, intervals).

Conclusion

- The Bayesian logistic regression model presented here provides a powerful and interpretable approach to credit risk prediction. By modeling uncertainty explicitly, it offers not only probability estimates of loan default but also credible intervals for each parameter, making the results more reliable in decision-making contexts.
- The evaluation metrics demonstrate that the model performs well in distinguishing between good and risky borrowers, while the SHAP analysis highlights the relative importance of key features such as income, credit score, and loan-to-income ratio. These insights can help financial institutions make informed and transparent lending decisions.
- Overall, this project illustrates the advantages of Bayesian modeling in credit risk analysis:
- **Uncertainty quantification** in predictions.
- **Robust parameter estimation** through posterior distributions.
- **Interpretability** with SHAP-based feature impact analysis.
- The framework can be extended by incorporating additional features, using more complex priors, or scaling to larger datasets. This makes Bayesian approaches highly promising for real-world financial risk management.