ALMOSA, JHON MICKO M. **APRIL 29, 2024**

CAS – 05 – 601P **BAYESIAN**

Research About PYMC3

PyMC3 is a Python library for Bayesian statistical modeling and probabilistic machine learning. It provides a high-level interface for defining and fitting Bayesian models using Markov chain Monte Carlo (MCMC) and variational inference algorithms. PyMC3 is built on top of Theano, a library for symbolic mathematical computation in Python.

Here are some key features of PyMC3:

Probabilistic Programming: PyMC3 allows you to specify probabilistic models using a syntax that closely mirrors the mathematical notation used in Bayesian statistics.

Flexible Model Specification: You can define models using a wide range of probability distributions and incorporate complex dependencies between variables.

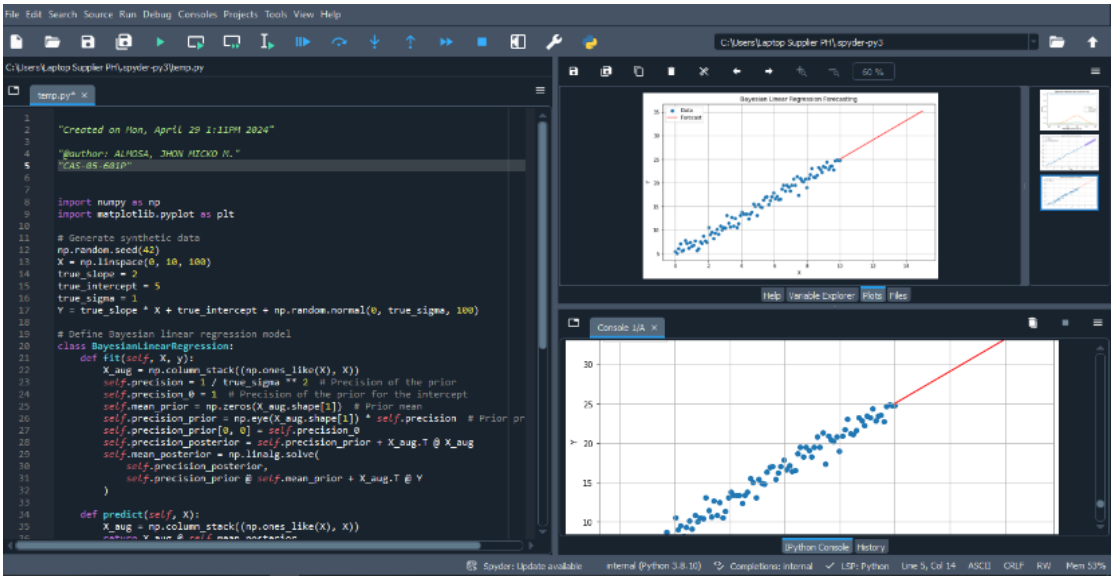
Scalable Inference: PyMC3 supports both MCMC and variational inference algorithms for approximating the posterior distribution of model parameters. These algorithms can handle models of varying complexity and size.

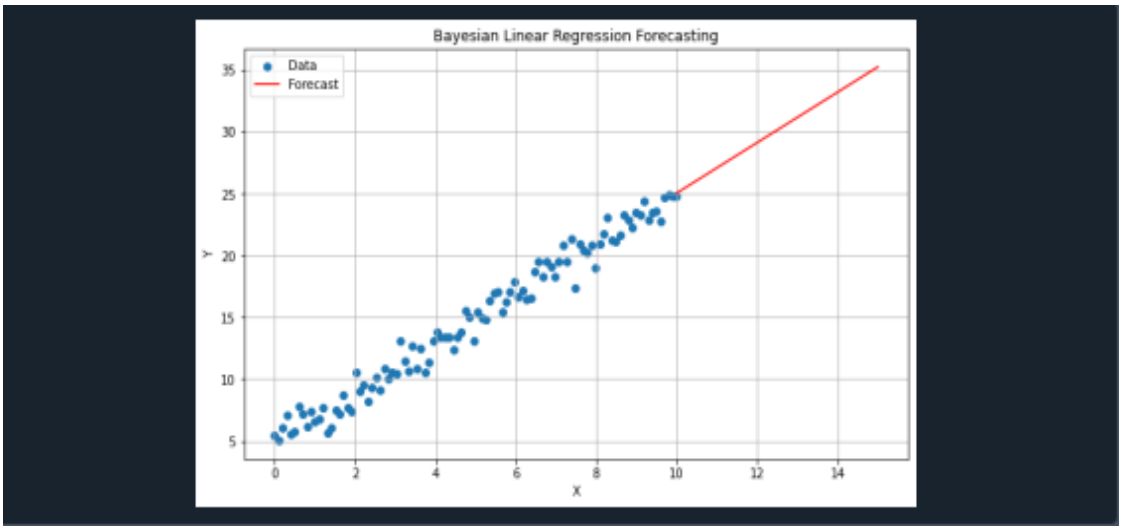
Automatic Differentiation: PyMC3 leverages Theano's automatic differentiation capabilities to efficiently compute gradients required for optimization and inference.

Model Checking and Diagnostics: PyMC3 provides tools for checking the convergence of MCMC chains, diagnosing potential problems with the model, and assessing the quality of inference results.

Overall, PyMC3 is a powerful tool for Bayesian modeling and inference in Python, suitable for both beginners and experienced practitioners in the field of Bayesian statistics and machine learning.

Create a sample model/script about Bayesian model for forecasting/regression (check for online sources) - include your name as author in the script screenshot





CODE (Sample Model/Script):

"Created on Mon, April 29 1:11PM 2024"

"@author: ALMOSA, JHON MICKO M."

"CAS-05-601P"

```
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data
np.random.seed(42)
X = np.linspace(0, 10, 100)
true_slope = 2
true_intercept = 5
true_sigma = 1
Y = true_slope * X + true_intercept + np.random.normal(0, true_sigma, 100)

# Define Bayesian linear regression model
class BayesianLinearRegression:
    def fit(self, X, y):
        X_aug = np.column_stack((np.ones_like(X), X))
        self.precision = 1 / true_sigma ** 2  # Precision of the prior
        self.precision_0 = 1  # Precision of the prior for the intercept
        self.mean_prior = np.zeros(X_aug.shape[1])  # Prior mean
        self.precision_prior = np.eye(X_aug.shape[1]) * self.precision  # Prior precision
        self.precision_prior[0, 0] = self.precision_0
        self.precision_posterior = self.precision_prior + X_aug.T @ X_aug
        self.mean_posterior = np.linalg.solve(
            self.precision_posterior,
            self.precision_prior @ self.mean_prior + X_aug.T @ Y
        )

    def predict(self, X):
```

```
        X_aug = np.column_stack((np.ones_like(X), X))
        return X_aug @ self.mean_posterior

# Fit the Bayesian model
model = BayesianLinearRegression()
model.fit(X, Y)

# Generate forecasts
X_forecast = np.linspace(10, 15, 50)
Y_forecast = model.predict(X_forecast)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(X, Y, label='Data')
plt.plot(X_forecast, Y_forecast, color='red', label='Forecast')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Bayesian Linear Regression Forecasting')
plt.legend()
plt.grid(True)
plt.show()
```