We assume Gaussian priors for the regression coefficients and a Gaussian likelihood for the observational model. This probabilistic framework allows us to incorporate and leverage prior knowledge about the variability in stock performance metrics effectively. Specifically, the weekly percentage change in DJIA stock prices (y) is modeled as a normally distributed continuous random variable dependent on the predictors ($\mathbf{X}$), as shown in the following equation to calculate the probability for N observed outputs $y = \{y_n\}_{n=1}^N$, using weights w and likelihood precision hyperparameter β:

$$p(y \mid \mathbf{X}, w) = \prod_{n=1}^{N} \mathcal{N}(y_n \mid w^T \mathbf{X}_n, \beta^{-1}).$$

Here, the hyperparameter β accounts for the inherent noise in the financial markets, and the prior distribution is assumed to be a zero-mean isotropic Gaussian governed by a single precision parameter α, so that $p(w \mid \alpha) = \mathcal{N}(w \mid 0, \alpha^{-1}I)$. This allows us to use the posterior distributions given by [1], §3.3.1. Rather than committing to specific weights, we use the closed-form of the posterior distribution giving training data to evaluate the likelihoods of new data as provided by [1], §3.3.2, and fit hyperparameters α and β using a grid search with evidence. That is, for each hyperparameter configuration, we estimate compute the marginal likelihood over the entire training dataset by integrating on the weights, which is worked out in [1], §3.5.1. Once we have picked the hyperparameter configuration with the best score, we will use this with the training data to evaluate the test scores using the posterior predictive distribution over the test data. This will later be compared to the upgraded model's score on the test data to decide which is better at predicting future price movement. The implementation of this Bayesian model will be executed using Python, with NumPy utilized for matrix operations and SciPy for optimization tasks. These tools are chosen for their computational efficiency and widespread use in the data science community. One significant advantage of our Bayesian approach with posterior predictive estimators is its capability to make probabilistic predictions. Utilizing the posterior predictive distribution, we can forecast future stock prices while quantifying the uncertainty of these predictions by not committing to specific weights. This is particularly crucial in risk-sensitive environments like stock trading, as it provides a spectrum of probable outcomes rather than a single point estimate, thereby offering a more nuanced view of potential future scenarios. We will use the `scipy` library to compute log PDFs of normal distributions and will reuse some of our code from CP2.

4 Hypothesis-Driven Upgrade Method

We hypothesize that compared to the baseline Gaussian likelihood model, using a location-scale Student's t-distribution for the likelihood should improve the performance metric of average posterior predictive distribution log probability of the test dataset, especially when there is a higher probability for percentage price change data to occur very far from the mean, because of the ability of these distributions to have heavier tails than the Gaussian distribution. We expect this situation may occur because events like news or surprises in earnings reports may cause a stock to move dramatically up or down in one week when it was not moving much in prior weeks. In this framework, the weekly percentage change in DJIA stock prices (y) is modeled as a location-scale t-distributed distributed continuous random variable dependent on the predictors ($\mathbf{X}$), as shown in the following equation to calculate the probability for N observed outputs $y = \{y_n\}_{n=1}^N$, using weights w and hyperparameters ν controlling the

$$p(y \mid \mathbf{X}, w) = \prod_{n=1}^{N} \mathit{lst}(y_n \mid w^T \mathbf{X}_n, \tau^2, \nu).$$

Here, the hyperparameter ν accounts for the amount of probability mass in the tails and τ is the scale hyperparameter, while the location parameter is given by each $w^T\mathbf{X}_n$ and the prior distribution over w is again assumed to be a zero-mean isotropic Gaussian governed by a single precision parameter α, so that $p(w \mid \alpha) = \mathcal{N}(w \mid 0, \alpha^{-1}I)$. Use of the t distribution results in us no longer having a closed form posterior. In order to overcome this hurdle, we will use a Markov chain Monte Carlo algorithm to draw samples and estimate expectations. We know by Bayes' rule that

$$p^*(w) = p(w \mid y_{1:N}) = \frac{p(w, y_{1:N})}{p(y_{1:N})} = cp(w, y_{1:N}) = c\tilde{p}(w)$$

is the posterior probability distribution over the weights w for some (unknown) constant $c > 0$. By the product rule,

$$\tilde{p}(w) = p(w, y_{1:N}) = p(w)p(y_{1:N} \mid w) = \mathcal{N}(w \mid 0, \alpha^{-1}I) \prod_{n=1}^{N} lst(y_n \mid w^T \mathbf{X}_n, \tau^2, \nu).$$

While we are unable to compute the posterior $p^*(w)$ directly, we can draw samples from it using the Metropolis-Hastings Markov Chain Monte Carlo (MCMC) algorithm since we know $\tilde{p}(w)$ is some positive constant multiple of $\tilde{p}(w)$, which we can compute. We will use the same proposal distribution for the random walk as we used in CP3; that is, we draw samples from a normal distribution with the previous values of the weights as the means. We will use the maximum likelihood estimator of the student's t-distribution to compute initial values of the weights, and use a list of different standard deviation vectors for random walk proposals, each with 0.8 at one position and zero at others, as this was found to work well in CP3 so that each proposal tries to change one thing at a time. Finally, to approximate the posterior predictive estimator, which would be otherwise a very difficult integral, we do a Monte Carlo estimate averaging over S samples of weights drawn using the MCMC algorithm, which we then transform into log-space for numerical stability:

$$p(y^* \mid y_{1:N}) \approx \frac{1}{S} \sum_{s=1}^{S} p(y^* \mid w^s), \qquad w^s \sim p(w, v | t_{1:N})$$

$$\log p(y^* \mid y_{1:N}) \approx \log \sum_{s=1}^{S} \exp[\log p(y^* \mid w^s)] - \log S,$$

from which we can compute the average per-example score over the test set to compare with the average scores for the baseline method in order to see which is better. We will again use the `scipy` library to compute log PDFs, for both the normal and student's t distribution, and will re-use some of our CP3 code for the MCMC algorithm. The hyperparameters are again α from the prior distribution, along with ν and τ in the student t-distribution, and we tune these using a grid search with evidence. That is, for each hyperparameter configuration, we draw samples of the weights again using the MCMC method and compute the log of the marginal likelihood across the entire train data set.

5 Results and Analysis

We were able to successfully complete the PPE on the blocks of data for the baseline method using the adapted techniques from CP2 doing a grid search on the evidence. It was difficult to determine a way to graph the data adequately to show the best model's performance on the train and test because each observation had many input dimensions, unlike the data from CP2. In Figure 3 we show the resulting test log likelihood values across the test observations for the best model.

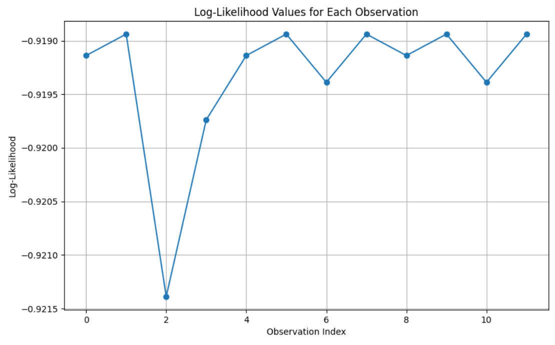

Figure 3: PPE log likelihood values on the test data's weekly observed percentage price change after the fourth week in each period for different test data observations, broken down by individual stock and 4-week interval.