MEM 679 Project Summary

Predicting Concrete Strength with Uncertainty

December 3, 2025

1. Task

Concrete can fail. When it does, buildings crack and bridges buckle. The stakes are high. My goal here is to predict how strong a concrete mix will be based on its ingredients. But I'm not stopping at a single number. I want the full picture—how sure am I about that prediction?

The pipeline works like this. Take a mix recipe. Predict its strength in MPa. Wrap that prediction in a measure of doubt. Then decide: should we approve this batch for use in a real structure? I'll only say yes when I'm confident the strength clears 35 MPa at least 95% of the time. That's the threshold. Conservative, but concrete failures aren't something you want to gamble on.

2. Data Set

I'm using the UCI Concrete dataset. It has 1,030 samples. Each sample lists eight things about the mix: cement, slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate, and curing age. The target is strength in megapascals.

All inputs are zero or positive. Some can hit zero—you don't always add slag or fly ash. Others stay positive—you always need cement and water. Age runs from 1 day to 365 days. It's discrete in nature, but I treat it as smooth. The output ranges from about 2 MPa (weak) to 83 MPa (strong). Always positive.

In math terms: inputs live in $\mathbb{R}^8_+$, output in $\mathbb{R}_+$. After I add new features, inputs expand to $\mathbb{R}^{11}$.

I split the data 80/20. Training gets 824 samples. Testing gets 206. I fix the random seed so results stay the same each run.

3. Transformation

Raw data needs some massaging. First, I build three new features grounded in how concrete actually works.

One: log of age plus one. Strength grows fast early, then slows down. A log handles that curve well. Two: water divided by cement. This ratio is famous in the field. More water means weaker concrete. Three: total bite—cement plus slag plus fly ash. All three react and harden over time.

Next, I standardize. For each feature, subtract the training mean and divide by the training standard deviation. This gives me:

$$\tilde{x} = (x - \mu) / \sigma$$

I fit the scaler on training data only. Then I apply it to both sets. This stops information from leaking.

What about the output? I keep it in raw MPa. The range isn't huge—maybe 40 times from low to high. A log transform might help a bit, but it would make the results harder to read. I value clarity here.

4. Model Setup

I'm testing two models. The first is simple. The second is flexible.

Bayesian Linear Regression. The model assumes strength is a weighted sum of features plus noise:

$$y = X\beta + \varepsilon$$

The weights β get a normal prior centered at zero with wide spread ($\sigma = 10$). The noise level σ gets a half-normal prior, also with scale 10. Both are vague on purpose. I let the data do most of the talking.

$$\beta \sim \text{Normal}(0, 100)$$

$$\sigma \sim \text{HalfNormal}(10)$$

$$y \mid \beta, \sigma \sim \text{Normal}(X\beta, \sigma^2)$$

Parameters: β sits in $\mathbb{R}^{11}$, σ in $\mathbb{R}_+$. The prior widths are fixed, not learned. Key assumptions: the link between features and strength is straight, and the noise stays constant across all samples.

Gaussian Process. This one can bend. It models strength as a smooth random function:

$$f \sim \text{GP}(0, k)$$

$$y = f(x) + \varepsilon$$

The kernel k measures how similar two inputs are. I use an RBF kernel with a twist—each feature gets its own length scale ℓ_i. If a length scale blows up, that feature doesn't matter much. This is called automatic relevance detection.

$$k(x, x') = \sigma^2_f \cdot \exp(-\tfrac{1}{2} \Sigma_i (x_i - x'_i)^2 / \ell_i^2)$$

Hyperparameters to learn: signal variance σ^2_f, eleven length scales ℓ_i, and noise variance σ^2_n. All positive. I bound them during fitting to keep things stable.

5. Fit the Model

For BLR: I go full Bayesian. That means sampling from the posterior, not just finding a peak. I use the NUTS sampler in PyMC. Four chains run side by side. Each draws 2,000 samples after 1,000 warmup rounds. This gives me 8,000 posterior samples total.

I check that chains mix well. R-hat should stay below 1.01. Effective sample size should top 400. If either fails, something went wrong.

For GP: Full Bayesian would be slow. Instead, I find hyperparameters that maximize the marginal likelihood—a MAP approach. The optimizer is L-BFGS-B. I

restart it ten times from random spots to dodge local traps. Once I lock in the hyperparameters, the predictive distribution at new points is exact.

6. Validation

How do I know the model works? Several checks.

Accuracy metrics. On the test set, I compute RMSE, MAE, and R^2. I'm aiming for R^2 above 0.85. That would mean the model captures most of what's going on.

Calibration. If I build a 90% interval, do 90% of real values land inside? I test this at 50%, 68%, 90%, and 95%. The gap between expected and observed coverage tells me if the model is too confident or too timid.

Posterior predictive checks. I draw fake datasets from the fitted model. Do they look like the real data? I compare means, variances, and residual shapes. If the fakes look odd, the model is missing something.

7. Prediction

I care about two things for each test mix. First, what strength do I expect? Second, how likely is that strength to clear a safety bar?

I work with samples, not summaries. For BLR, I pull β and σ from the posterior. Then I sample predicted y values. For GP, the predictive is already a nice normal—I sample from it directly. Each test point gets 1,000 samples.

From those samples, I report the mean and standard deviation. I also give intervals: 50%, 90%, and 95% credible bands. These come straight from sample quantiles.

When possible, I split uncertainty into two parts. Epistemic uncertainty comes from not knowing the true model. It shrinks with more data. Aleatoric uncertainty is baked-in noise. No amount of data fixes that.

8. Decision

Here's where uncertainty earns its paycheck. The decision rule is simple:

$$\textit{Approve if } P(\text{strength} \geq 35\ \text{MPa}) \geq 0.95$$

I compute that probability from samples. Count how many land at or above 35. Divide by the total. Done.

Why does this matter? Picture two mixes. Both have a predicted mean of 40 MPa. But one has tight spread ($\sigma = 2$), the other wide ($\sigma = 8$). The tight mix passes easily—nearly all samples clear 35. The wide mix fails. Too many samples dip below the bar. Same average, different fates. That's exactly how you want a safety system to behave.

I'll also test how sensitive this is. What happens if I move the bar to 30 MPa? Or 40 MPa? What if I demand 99% reliability instead of 95%? And do BLR and GP agree on which mixes pass? These questions round out the analysis.

—

Current status: Both models run. The data pipeline works. I'm now wiring together the prediction and decision pieces. Full results should be ready soon.