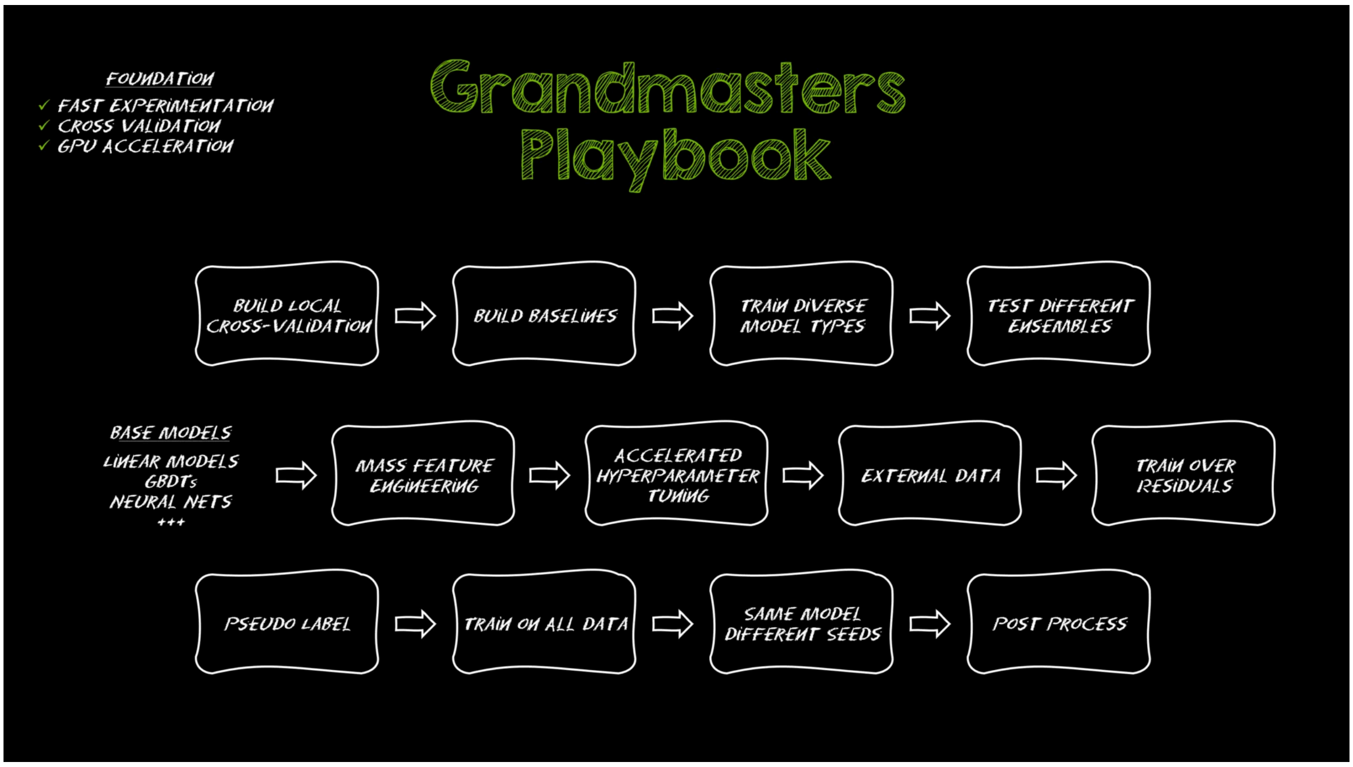
Data Science

English

The Kaggle Grandmasters Playbook: 7 Battle-Tested Modeling Techniques for Tabular Data

Lessons from years of competitions, made practical with GPU acceleration.

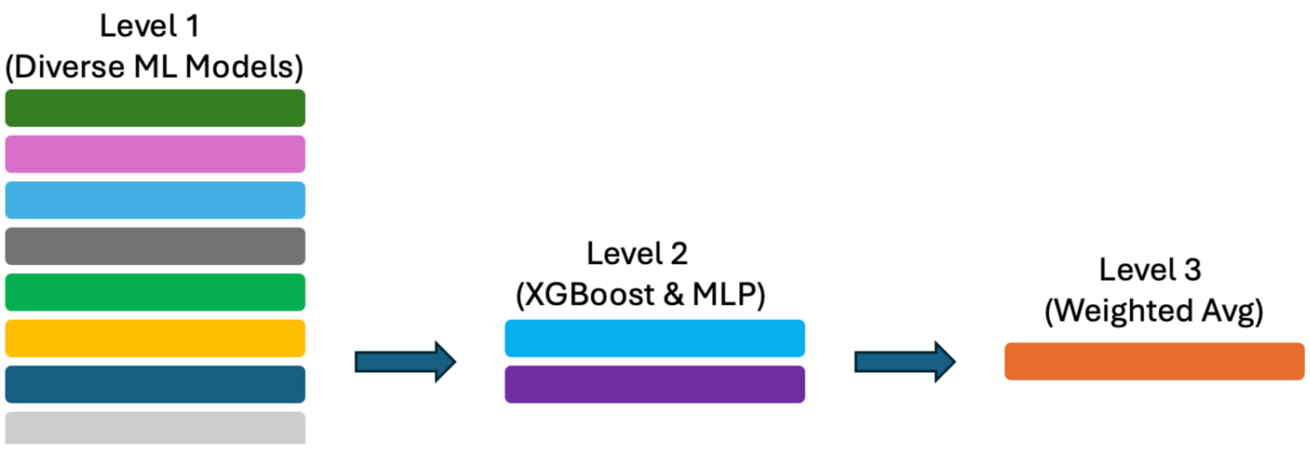


Sep 18, 2025

+43 Like Discuss (1)

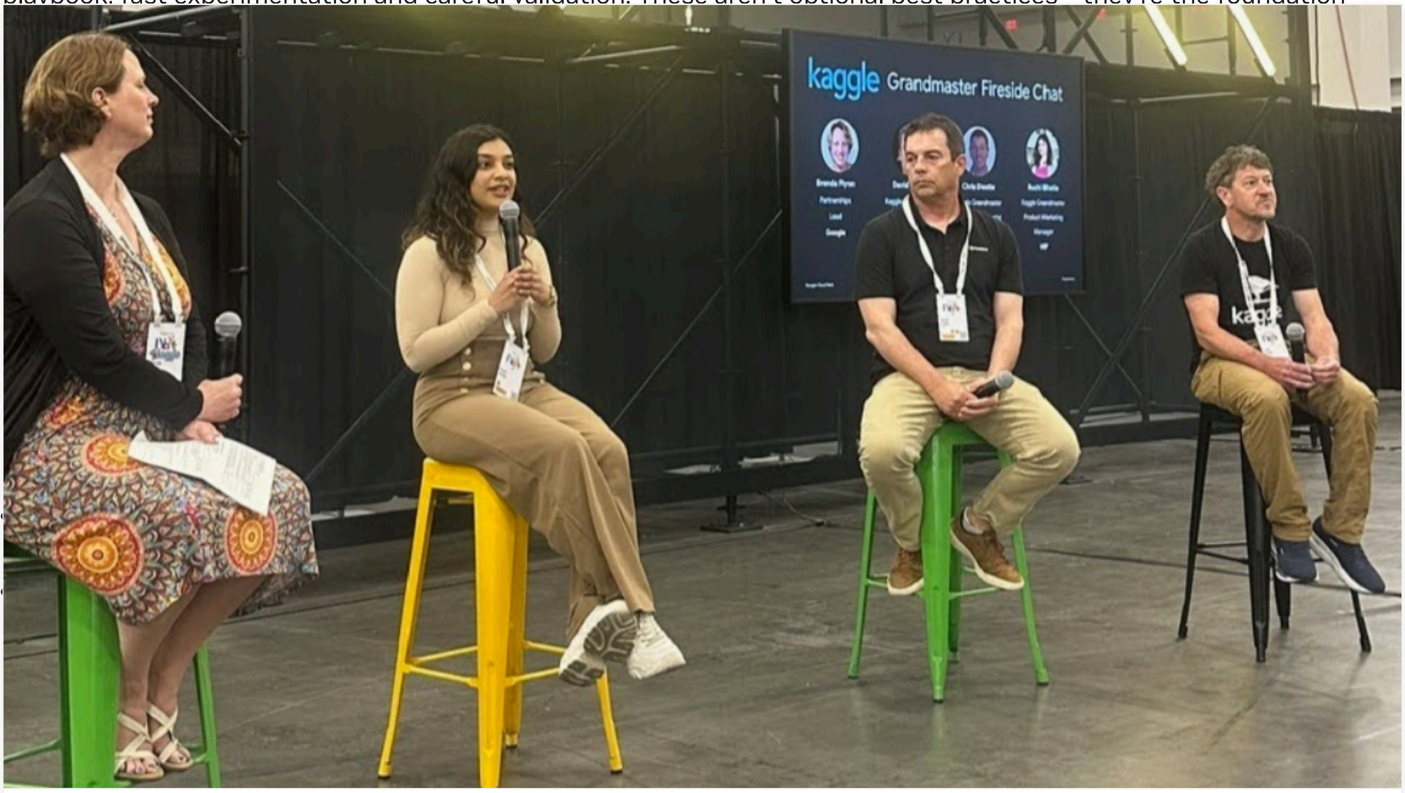
By Kazuki Onodera, Théo Viel, Gilberto Titericz and Chris Deotte

Related posts



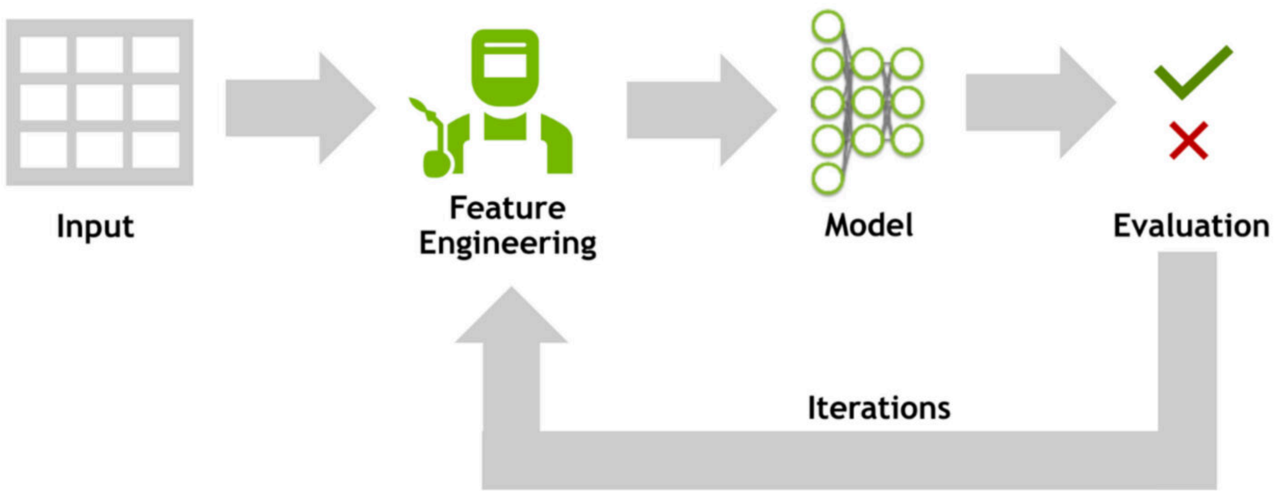
Grandmaster Pro Tip: Winning First Place in a Kaggle Competition with Stacking Using cuML

playbook: fast experimentation and careful validation. These aren't optional best practices—they're the foundation



Kaggle Grandmasters Unveil Winning Strategies for Data Science Superpowers

workflow.

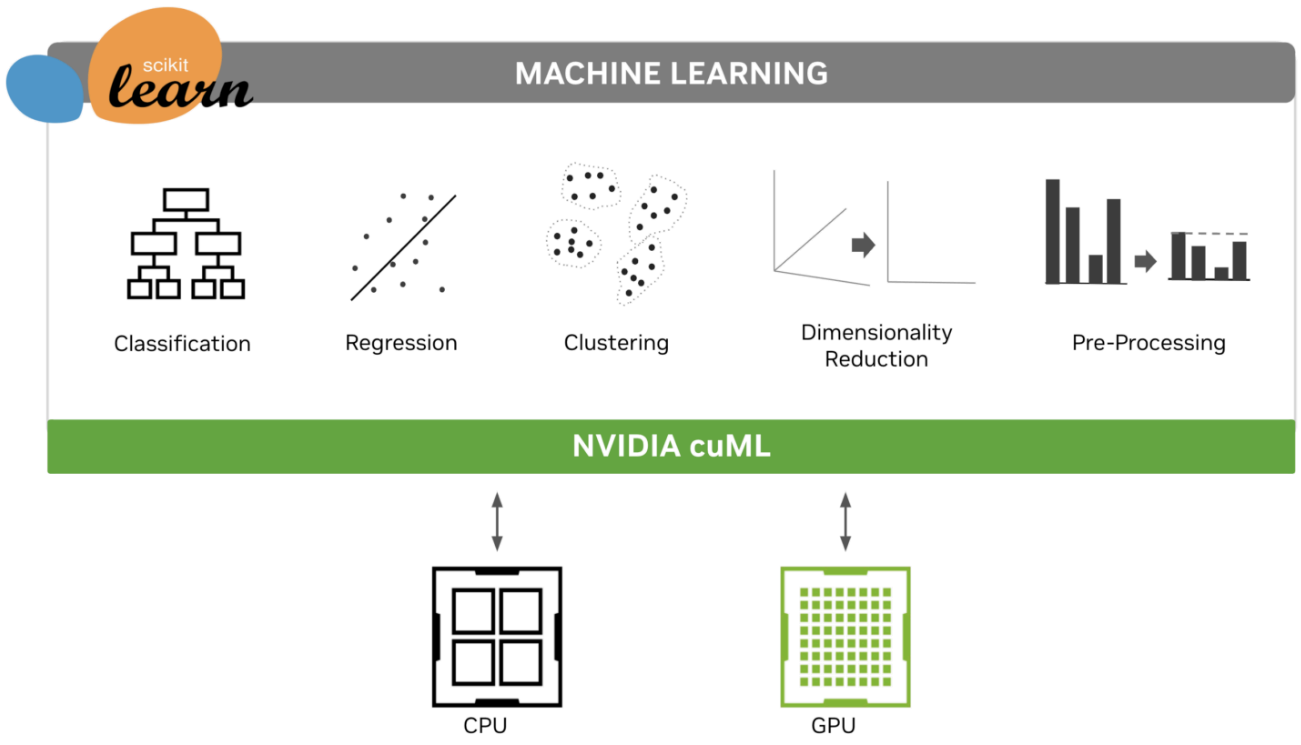


Grandmaster Pro Tip: Winning First Place in Kaggle Competition with Feature Engineering Using cuDF pandas

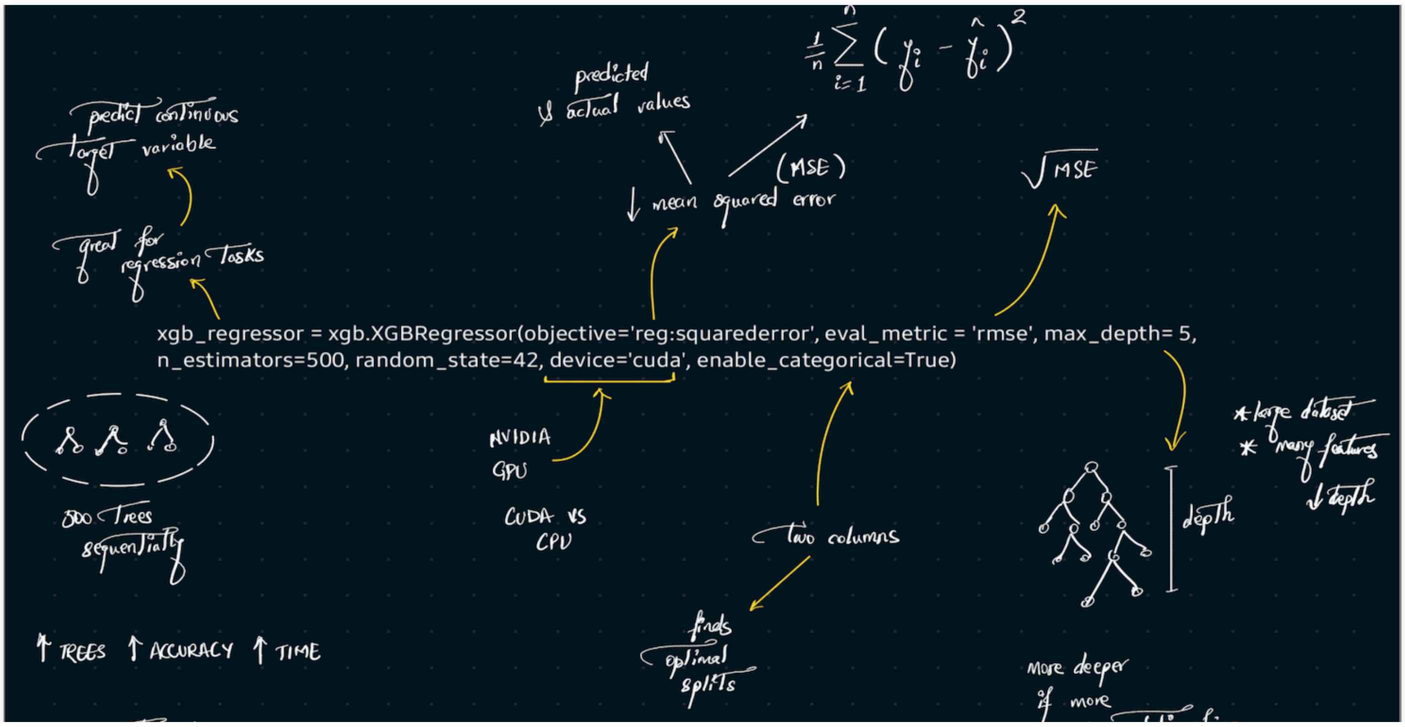
the data a little deeper—a couple of quick checks that we've found useful, but many people miss:

Train vs. test distribution checks: Spot when evaluation data differs from training, since distribution shift can cause models to validate well but fail in deployment.

📊 Explore Data Distribution - Train vs Test



NVIDIA cuML Brings Zero Code Change Acceleration to scikit-learn


predict continuous target variable
great for regression tasks
predicted vs actual values
↓ mean squared error (MSE)
$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
$\sqrt{MSE}$
NVIDIA GPU
CUDA vs CPU
500 trees sequentially
two columns
finds optimal splits
depth
* large dataset
* many features
↓ depth
more deeper if more
↑ TREES ↑ ACCURACY ↑ TIME

```
xgb_regressor = xgb.XGBRegressor(objective='reg:squarederror', eval_metric = 'rmse', max_depth= 5,
n_estimators=500, random_state=42, device='cuda', enable_categorical=True)
```

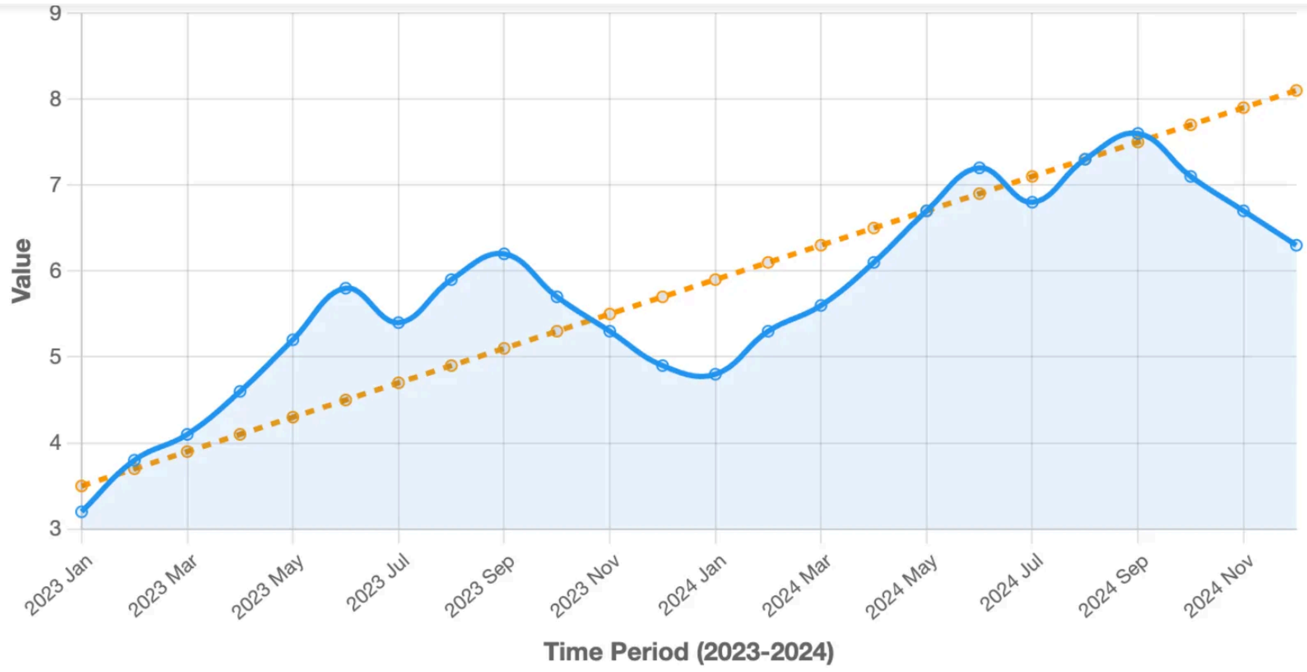


Figure 2. Analyzing the target variable over time uncovers a strong upward trend with seasonal fluctuations and accelerating growth. Ignoring temporal patterns like these can mislead models unless time-aware validation is used.

These techniques aren't brand new—but they're often overlooked, and ignoring them can sink a project.

Why it matters: Skipping these checks can derail an otherwise solid workflow.

In action: In the winning solution to the Amazon KDD Cup '23, the team uncovered both a train—test distribution shift and temporal patterns in the target—insights that shaped the final approach. Read the full write-up >

Made practical with GPUs: Real-world datasets are often millions of rows, which can slow to a crawl in pandas. By adding GPU acceleration with NVIDIA cuDF, you can run distribution comparisons and correlations at scale in seconds. Read the technical blog >

2. Build diverse baselines, fast

Most people build a few simple baselines—maybe a mean prediction, a logistic regression, or a quick XGBoost—and then move on. The problem is that a single baseline doesn't tell you much about the landscape of your data.

Our approach is different: We spin up a diverse set of baselines across model types right away. Seeing how linear models, GBDTs, and even small neural nets perform side-by-side gives us far more context to guide experimentation.

Why it matters: Baselines are your gut check—they confirm your model is doing better than guessing, set a minimum performance bar, and act as a rapid feedback loop. Re-running baselines after data changes can reveal whether you're making progress—or uncover problems like leakage.

Diverse baselines also show you early which model families fit your data best, so you can double-down on what works instead of wasting cycles on the wrong path.

and Support Vector Regression (SVR) models, without any feature engineering, was enough to earn us second place. And while exploring other baselines, we found that even a single Support Vector Classifier (SVC) baseline would have placed near the top of the leaderboard. Read the full write-up >

Made practical with GPUs: Training a variety of models can be painfully slow on CPUs. With GPU acceleration, it's practical to try them all—cuDF for quick stats, cuML for linear/logistic regression, and GPU-accelerated XGBoost, LightGBM, CatBoost, and neural nets—so you can get better insight in minutes, not hours.

3. Generate more features, discover more patterns

Feature engineering is still one of the most effective ways to boost accuracy on tabular data. The challenge: generating and validating thousands of features with pandas on CPUs is far too slow to be practical.

Why it matters: Scaling beyond a handful of manual transformations—into hundreds or thousands of engineered features—often reveals hidden signals that models alone can't capture.

Example: Combining categorical columns

In one Kaggle competition, the dataset had eight categorical columns. By combining pairs of them, we created 28 new categorical features that captured interactions the original data didn't show. Here's a simplified snippet of the approach:

```
for i,c1 in enumerate(CATS[:-1]):
     for j,c2 in enumerate(CATS[i+1:]):
        n = f"{c1}_{c2}"
        train[n] = train[c1].astype('str')+"_"+train[c2].astype('str')
```

In action: Large-scale feature engineering powered first-place finishes in the Kaggle Backpack and Insurance competitions, where thousands of new features made the difference.

Made practical with GPUs: With cuDF, pandas operations like groupby, aggregation, and encoding run orders of magnitude faster, making it possible to generate and test thousands of new features in days instead of months.

Check out the technical blog and training course below for hands-on examples:

- Blog: Grandmaster Pro Tip: Winning First Place in Kaggle Competition with Feature Engineering Using cuDF pandas
- Course: Best Practices in Feature Engineering for Tabular Data

Combing diverse models (ensembling) boosts performance

We found that combining the strengths of different models often pushes performance beyond what any one model can achieve. Two techniques that are particularly useful are hill climbing and model stacking.

Hill climbing is a simple, but powerful way to ensemble models. Start with your strongest single model, then systematically add others with different weights, keeping only the combinations that improve validation. Repeat until no further gains.

Why it matters: Ensembling captures complementary strengths across models, but finding the right blend is hard. Hill climbing automates the search, often squeezing out accuracy and outperforming single model solutions.

In action: In the *Predict Calorie Expenditure* competition, we used a hill climbing ensemble of XGBoost, CatBoost, neural nets, and linear models to secure first place. Read the write-up >

Made practical with GPUs: Hill climbing itself isn't new—it's a common ensemble technique in competitions—but it normally becomes too slow to apply at large-scale. With CuPy on GPUs, we can vectorize metric calculations (like RMSE or AUC) and evaluate thousands of weight combinations in parallel. That speedup makes it practical to test far more ensembles than would be feasible on CPUs, often uncovering stronger blends.

Here's a simplified version of the code used to evaluate Hill Climbing ensembles on GPU:

```
import cupy as cp

def multiple_rmse_scores(actual, predicted):
    if len(actual.shape)==1:
        actual = actual[:,cp.newaxis]
    rmses = cp.sqrt(cp.mean((actual-predicted)**2.0,axis=0))
    return rmses

def multiple_roc_auc_scores(actual, predicted):
    n_pos = cp.sum(actual)
    n_neg = len(actual) - n_pos
    ranked = cp.argsort(cp.argsort(predicted, axis=0), axis=0)+1
    aucs = (cp.sum(ranked[actual == 1, :], axis=0)- n_pos\
    *(n_pos + 1)/2) / (n_pos*n_neg)
    return aucs
```

5. Stacking

Stacking takes ensembling a step further by training one model on the outputs of others. Instead of averaging predictions with weights (like hill climbing), stacking builds a second-level model that learns how best to combine the outputs of other models.

Why it matters: Stacking is especially effective when the dataset has complex patterns that different models capture in different ways – like linear trends vs nonlinear interactions.

Pro tip: Two-ways to stack:

- Residuals: Train a Stage 2 model on what Stage 1 got wrong (the residuals).
- OOF Features: Use Stage 1 predictions as new input features for Stage 2.

Both approaches help squeeze more signal out of the data by capturing patterns that base models miss.

In action: Stacking was used to win first place in the Podcast Listening Time competition, where a three-level stack of diverse models (linear, GBDT, neural nets, and AutoML) was used. Read the technical blog >

Figure 3. The winning entry in the Kaggle April 2025 Playground competition used stacking with three levels of models, with the results of each level used in subsequent levels.

Made practical with GPUs: Stacking is a well-known ensembling technique—but deep stacks quickly become computationally expensive, requiring hundreds of model fits across folds and levels. With cuML and GPU-accelerated GBDTs, we can train an evaluate stacks an order of magnitude faster, making it realistic to explore multi-level ensembles in hours instead of days.

6. Turn unlabeled data into training signal with pseudo-labeling

Pseudo-labeling turns unlabeled data into training signal. You use your best model to infer labels on data that lacks them (for example, test data or external datasets), then fold those "pseudo-labels" back into training to boost model performance.

Figure 4. Pseudo-labeling workflow—use a trained model to generate labels for unlabeled data, then fold those pseudo-labels back into training to improve performance.

Why it matters: More data = more signal. Pseudo-labeling improves robustness, acts like knowledge distillation (student models learn from a strong teacher's predictions), and can even help denoise labeled data by filtering out samples where models disagree. Using *soft labels* (probabilities instead of hard 0/1s) adds regularization and reduces noise.

Pro tips for effective pseudo-labeling:

- The stronger the model, the better the pseudo-labels. Ensembles, or multi-round pseudo-labeling usually outperform single-pass approaches
- Pseudo-labels can also be used for pretraining. Fine-tune on the initial data as a last step to reduce noise introduced earlier.
- Use soft pseudo-labels. They add more signal, reduce noise, and let you filter out low-confidence samples.
- Pseudo-labels can be used on labeled data—useful for removing noisy samples.
- Avoid information leakage. When using k-fold, you must compute k sets of pseudo-labels so that validation data never sees labels from models trained on itself.

In action: In the *BirdCLEF 2024* competition, the task was species classification from bird audio recordings. Pseudo-labeling expanded the training set with soft labels on unlabeled clips, which helped our model generalize better to new species and recording conditions. Read the full write-up >

acceleration (via cuML, XGBoost or CatBoost GPU backends), you can run several pseudo-labeling cycles in hours.

training

Even after optimizing our models and ensembles, we found two final tweaks that can squeeze out extra performance:

- **Train with different random seeds**. Changing initialization and training paths, then averaging predictions, often improves performance.
- **Retrain on 100% of the data**. After finding optimal hyperparameters, fitting your final model on all training data squeezes out extra accuracy.

Why it matters: These steps don't require new architectures—just more runs of the models you already trust. Together, they boost robustness and ensure you're making full use of your data.

In action: In the *Predicting Optimal Fertilizers* challenge, ensembling XGBoost models across 100 different seeds clearly outperformed single-seed training. Retraining on the full dataset provided another leaderboard bump. Read the full write-up >

Figure 5. Ensembling XGBoost with different random seeds (blue) steadily improves MAP@3 compared to single-seed averages (orange).

Note: MAP@3 (Mean Average Precision at 3) measures how often the correct label appears in the model's top three ranked predictions.

Made practical with GPUs: Faster training and inference on GPUs make it feasible to rerun models many times. What might take days on CPU becomes hours on GPU—turning "extra" training into a realistic step in every project.

This playbook is battle-tested, forged through years of competitions and countless experiments. It's grounded in two principles—fast experimentation and careful validation—that we apply to every project. With GPU acceleration, these advanced techniques become practical at scale, making them just as effective for real-world tabular problems as they are for climbing leaderboards.

If you want to put these ideas into practice, here are some resources to get started with GPU acceleration in the tools you already use:

- **Intro Notebook: Accelerating pandas with cuDF** (Zero-code-change acceleration for pandas workflows)
- **Intro Notebook: Accelerating scikit-learn with cuML** (Drop-in speedups for common ML models)
- **Intro Notebook: GPU-Accelerated XGBoost** (Train gradient-boosted trees on millions of rows in minutes)
- **NVIDIA cuDF overview**: Learn more about accelerating pandas and Polars.
- **NVIDIA cuML overview**: Scikit-learn–style machine learning with GPU acceleration.

Discuss (1) +43 Like

Tags

Data Science | General | CuDF | CuML | RAPIDS | Intermediate Technical | Tutorial | Featured | Pandas | XGBoost

About the Authors

About Kazuki Onodera
Kazuki Onodera-san is currently working as senior deep learning data scientist at NVIDIA. Before that, Kazuki was working at DeNA as Data Scientist. Kazuki has been a Kaggle Competition Grandmaster since 2019 and has had five top two competition rankings.
View all posts by Kazuki Onodera

About Théo Viel
Théo Viel is a four-time Kaggle Grandmaster and deep learning scientist at NVIDIA. Alongside other renowned Grandmasters, Théo is part of the KGMoN team, whose mission is to share the best data science practices—acquired by winning competitions—with NVIDIA and the data science community.
View all posts by Théo Viel

About Gilberto Titericz
Gilberto Titericz is a data scientist on the RAPIDS team at NVIDIA and a Kaggle grandmaster.
View all posts by Gilberto Titericz

About Chris Deotte
Chris Deotte is a senior data scientist at NVIDIA. Chris has a Ph.D. in computational science and mathematics with a thesis on optimizing parallel processing. Chris is a Kaggle 4x grandmaster.
View all posts by Chris Deotte