1st Place - Fast GPU Experimentation with RAPIDS cuDF cuML

Thanks Kaggle for another fun playground competition! My **June playground** "Predicting Optimal Fertilizers" solution uses techniques from my previous 6 months of playground solutions. Below are links to old solutions with more info. And I will describe the techniques again briefly below.

Techniques from Previous Playground Comps using GPU Acceleration!

Date	**Name**	**Metric**	**Technique(s)**	**Solution Link**	**Rank**
Dec 2024	Insurance Comp	RMSLE	**RAPIDS cuDF Feature Engineering**	link	1st
Jan 2025	Forecasting Comp	MAPE	**Boosting** over Residuals	link	2nd
Feb 2025	Backpack Comp	RMSE	Use **Original Data**	link	1st
Mar 2025	Rainfall Comp	AUC	**RAPIDS cuML**	link	2nd
Apr 2025	Podcast Comp	RMSE	cuML **Stacking**	link	1st
May 2025	Calorie Comp	RMSLE	**GPU HillClimbing**	link	1st

RAPIDS cuDF Feature Engineering

The secret sauce in this competition is cuDF feature engineering, specifically **Target Encoding**. The competition data has 8 features which can each be treated as categorical.

First we make all pairs which is 28 new columns. Then we make all triples which is 56 new columns. Then we make all quadruples which is 70 new columns. From these 162 = 8+28+56+70 columns, we use cuML Target Encoder to encode the 7 binary targets, y==0?, y==1?, ..., y==6?. This produces 1134 new columns. Then we target encode using the original dataset to produce another 1134 new columns. We then train an XGBoost using all these 2268 columns!

Use Original Data

Using relevant external data helps in all Kaggle competitions. This technique is especially important in Kaggle's playground competitions because every competition is synthetic data generated from an original dataset. Therefore we want to utilize the original data.

There are two ways to use external data like the original dataset. We can add the data as **new rows**. Or we can add the data as **new columns**. Most public notebooks in this competition used the former technique. But no public notebook used the latter. (In our final hill climbing ensemble we need models trained with former and models trained with later).

To use the later, we first create a categorical column (by using original column or combination of original columns). Once we have a categorical column, we target encoding it using the original data.

```
TE = original_data.groupby(CAT_COL)[TARGET].agg("mean")

TE.name = f"TE_{CAT_COL}_orig"

train = train.merge(TE, on=CAT_COL, how='left')
```

Stacking (with cuML)

Stacking is especially beneficial for the MAP@3 metric which likes calibrated probabilities. We train an **NN level 2** stacking model with categorical cross entropy loss and it doesn't matter if stage 1 models are calibrated or not.

For stage 1 models, we don't even need multi-class. We can train individual models to predict single binary classification, Is the target y=0?, Is the target y=1? etc etc. We predict OOF probabilities using XGBoost, CatBoost, NN, cuML Linear Regression. Then we train a level 2 NN stacking model using categorical cross entropy loss to predict multi-class probabilities. This uses all the knowledge of stage 1 models and produces calibrated multi-class stage 2 predictions!

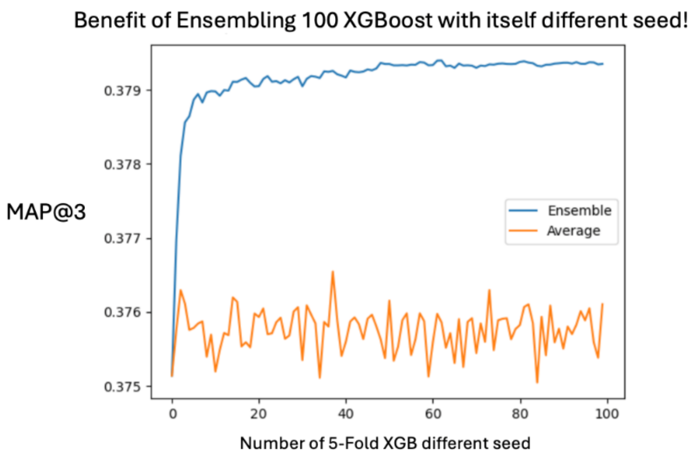
We also build an **GBDT level 2** stacking model using XGBoost and objective='multi:softprob'. The average of **NN level 2** and **GBDT level 2** is better than each stage 2 model individually!

Repeated KFold

Bizen250 published a beautiful public notebook here, which illustrates an important concept about MAP@3 metric. This metric is very sensitive to the randomness of training (and predicted probabilities). Therefore an easy way to boost MAP@3 metric is to train the same model again with different seed and average the predicted probabilities before selecting top3 for MAP@3.

For example, I train all my NNs 100 times and average the probabilities before selecting top 3 fertilizers. And I train all my GBDT dozens of times using 100% data and average the probabilities before selecting the top 3.

In the plot below, we see that each fold of each 5-Fold XGB has average MAP@3=0.376 but when we average the probabilities from 100 5-Fold XGB and then compute MAP@3, we achieve 0.380!



Hill Climbing (with cuML)

The effectiveness of hill climbing comes from creating a diverse set of models. Then when we combine them with weighted average, the result is better than any of the individual models. The key ingredients to tabular data hill climbing is building diverse GBDT, NN, and ML models. The fastest way to produce ML models is using RAPIDS cuML.

Boosting over Residuals

Note that we can give XGBoost starting predictions. First we train a cuML Linear Regression model. Then we use XGBoost's dtrain.set_base_margin(LINEAR_REGRESSION_LOGITS) to begin the boosting from the predictions of the linear regression model. This is an overlooked XGBoost trick!

Pseudo Labeling

A helpful trick in all competitions in the final days is to add test data labeled with your best ensemble. For classification, NN naturally handle the soft target probabilities (with their default cross entropy losses). For XGB, we write a custom objective to train multi-class with soft targets.

Re-Train with 100% Train

Another helpful trick in all competitions in the final days is to retrain all the most important models in your final ensemble using 100% train data. For XGBoost when going from 5-Fold to 100% we use fixed number of iterations equal to 25% more than average fixed during early stopping. And for NN, we just create a fixed step learning schedule from the average reduce on plateau of early stopping from 5-Fold.

Trust Your CV

For my final submission, I chose my best CV ensemble which was also my best LB score. My CV ensemble described below has CV MAP@3 = 0.386! Wow!

Final Submission - CV 0.386

My final submission is an ensemble of **9 models** where each is trained **many times** with different seeds. The CV score is 0.386 and Private LB is 0.38652 and public LB is 0.38450!

- XGBoost **cuDF FE** - use original **data as columns** w/ tree **depth = 4**. (And train 10 models with 100% data)
- XGBoost **cuDF FE** - use original **data as columns** w/ tree **depth = 10**. (And train 10 models with 100% data)
- **Stacking NN** with many stage 1 models (And train 25 models)
- XGBoost **RepeatedStratifiedKFold** - use original **data as rows** - Public notebook by @bizen250 here (Trains 50 XGB)
- **Stacking XGB** over LGBM - Public notebook by @ayushchandramaurya here (And train multiple versions different seeds)
- XGBoost - use original **data as rows** w/ tree **depth = 18** - Public notebook by @elainedazzio here (And train multiple versions different seeds)
- **NN** - Public notebook by @ricopue here (And train 100 NNs)
- **Stacking XGB** with many stage 1 models and with **pseudo label COLUMNS** (i.e KNN features).

- **NN** with **pseudo label ROWS** (And train 25 NNs)

So we use **9 models** and train each **many times** with different seeds. Therefore our final ensemble is approximately combining **300 sets of predictions**! Weights (for the 9 models) are determined by **GPU Hill Climbing** !

Enjoy!

Consider using all these techniques in Kaggle's July playground competition!