Credit Card Fraud Analysis

Shreyash Kondakindi

May 1, 2025

Part 1

Tests of different reasonable hyper-parameters for a variety of model types

Model		Parameters						Average FDR @ 3%		
Decision Tree	Iteration	criterion	splitter	max_depth	min_samples_split	min_samples_leaf	ccp_alpha	Train	Test	OOT
	1	gini	best	5	20	10	-	0.675	0.663	0.539
	2	entropy	random	17	70	50	-	0.665	0.651	0.546
	3	entropy	best	10	100	50	-	0.726	0.704	0.494
	4	gini	random	12	80	40	-	0.678	0.660	0.551
	5	gini	best	10	100	50	-	0.718	0.690	0.512
	6	gini	best	-	10	5	0.0001	0.719	0.695	0.523
XGBoost	Iteration	sub_sample	learning_rate	n_estimators	max_depth	gamma	reg_lambda	Train	Test	OOT
	1	-	0.1	100	6	10	10	0.742	0.728	0.576
	2	-	0.05	200	3	1	1	0.753	0.731	0.567
	3	-	0.05	30	7	5	5	0.748	0.729	0.582
	4	0.5	0.1	100	10	6	3	0.764	0.740	0.574
	5	0.6	0.01	100	12	8	3	0.733	0.720	0.579
	6	0.5	0.01	200	6	7	3	0.734	0.719	0.576
Multi-Layer Perceptron	Iteration	hidden_layer_sizes	activation	solver	alpha	learning_rate	max_iter	Train	Test	OOT
	1	(10,)	relu	adam	0	constant	150	0.681	0.666	0.559
	2	(10,5,2)	tanh	sgd	0	constant	250	0.665	0.658	0.548
	3	(100,20)	logistic	adam	0.01	adaptive	150	0.682	0.679	0.568
	4	(100,20,10,30)	relu	adam	0.1	invscaling	100	0.732	0.715	0.583
	5	(50,50)	relu	adam	0.1	constant	100	0.698	0.680	0.567
	6	(10,8,6,4,2)	relu	adam	0.01	constant	150	0.722	0.694	0.587
LightGBM	Iteration	boosting_type		num_leaves	max_depth	learning_rate	n_estimators	Train	Test	OOT
	1	gbdt		31	4	0.1	50	0.751	0.732	0.565
	2	dart		40	5	0.01	150	0.715	0.690	0.564
	3	gbdt		100	3	0.05	100	0.736	0.711	0.570
	4	gbdt		7	5	0.05	150	0.756	0.734	0.568
	5	gbdt		25	3	0.06	100	0.744	0.710	0.565
	6	dart		40	4	0.04	120	0.716	0.693	0.569
Catboost	Iteration	iterations		learning_rate	depth	l2_leaf_reg	border_count	Train	Test	OOT
	1	300		0.05	6	3	200	0.801	0.757	0.590
	2	400		0.02	7	4	300	0.781	0.751	0.581
	3	500		0.01	7	4	150	0.762	0.730	0.579
	4	1000		0.005	8	4	150	0.774	0.746	0.596
	5	200		0.01	9	6	120	0.733	0.702	0.578
	6	1000		0.006	8	5	200	0.785	0.741	0.589

Figure 1: Model Exploration

Part 2

Model Performance Analysis using comparision plots

This section presents an analysis of the performance of various classification models for fraud detection, based on their Fraud Detection Rate (FDR%) across training, testing, and out-of-train (OOT) datasets, as visualized in the accompanying box plots. The primary objective is to identify the model exhibiting the best generalization capability, indicated by its performance on the OOT dataset.

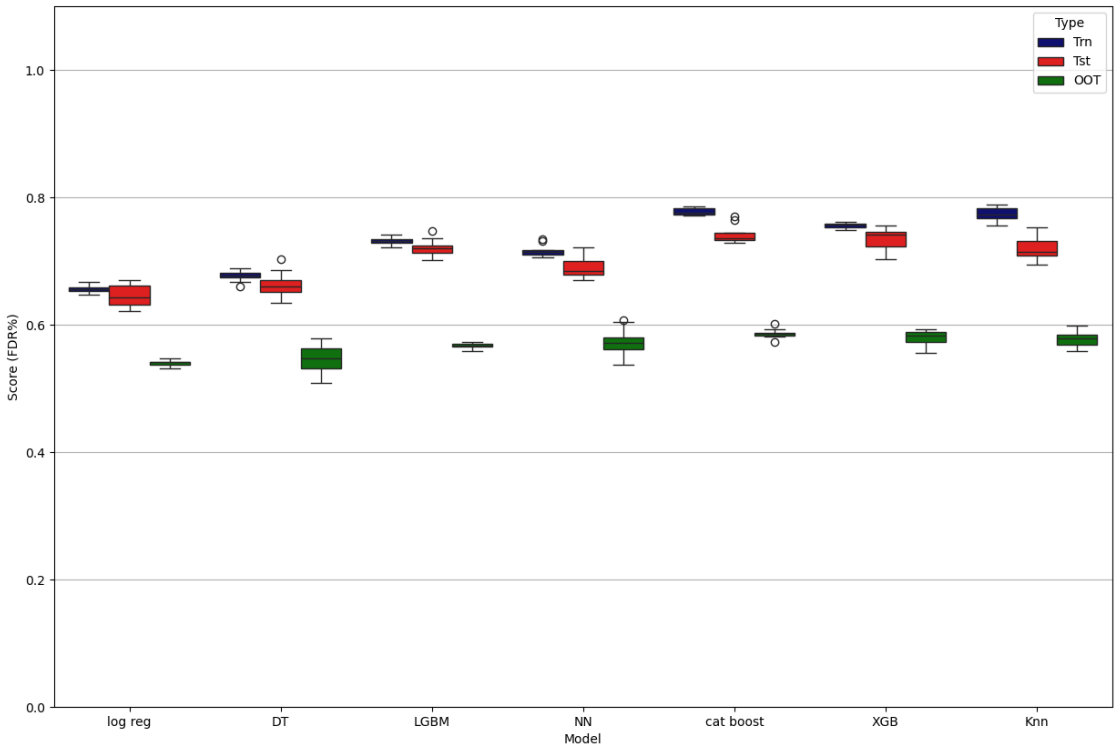

Figure 2: Model Performance

Models such as Logistic Regression and Decision Tree demonstrate relatively lower FDR% on the OOT set. While Logistic Regression shows consistency across datasets, its overall performance is limited. Decision Tree exhibits significant drop in performance from training to testing and OOT data. K-Nearest Neighbors shows considerable overfitting, with high training scores that do not translate to test or OOT performance.

The gradient boosting models, LightGBM and Neural Network, show improved performance on the test and OOT sets compared to simpler models. However, they still exhibit a notable performance drop from training.

CatBoost and XGBoost emerge as the top-performing models, achieving the highest median FDR% on the critical OOT dataset. Although both models show a performance gap between training/testing and OOT , their ability to generalize to unseen data, as measured by the OOT score, surpasses other models. Visually, CatBoost appears to have a marginally higher median OOT score and a relatively tight distribution, suggesting slightly better and more consistent performance on truly unseen data compared to XGBoost, which is a very close second.

Considering the imperative of strong performance on unseen data for fraud detection, **CatBoost** is identified as the most promising model based on its superior OOT Fraud Detection Rate in these tests. While overfitting is present in the top models, and there is no overlap in the train and test split for CatBoost unlike XGB which seems to have a closer gap between the two splits, it still demonstrates the most favorable balance between learning the underlying patterns and generalizing effectively to new instances. **XGBoost** seems to be a close second and I would also not mind having XGBoost as my favoured model.

Part 3

Demonstration of overfitting

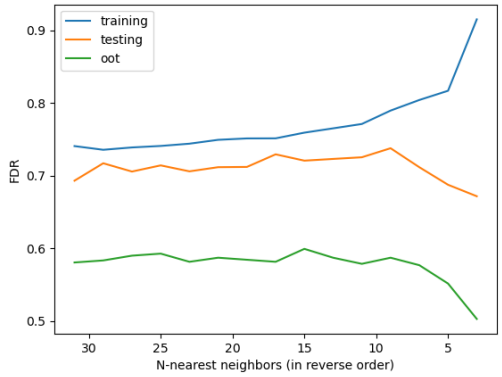

Figure 3: **KNN overfitting**
X axis shows number of neighbors used for kneighbors queries $\longrightarrow (n\_neighbors = i)$

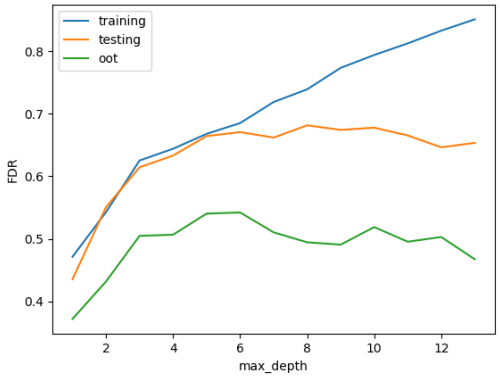

Figure 4: **Decision Tree overfitting**
Parameters $\longrightarrow (max\_depth = i)$

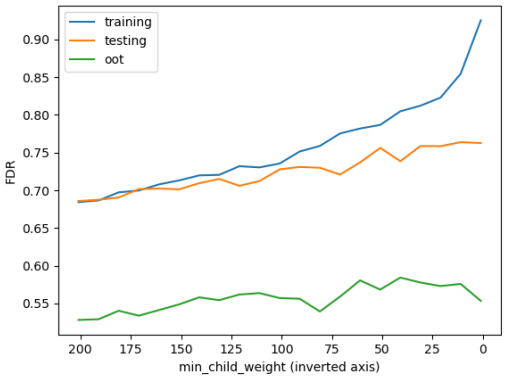

Figure 5: **XGBoost overfitting**
Parameters $\longrightarrow (min\_child\_weight = i)$

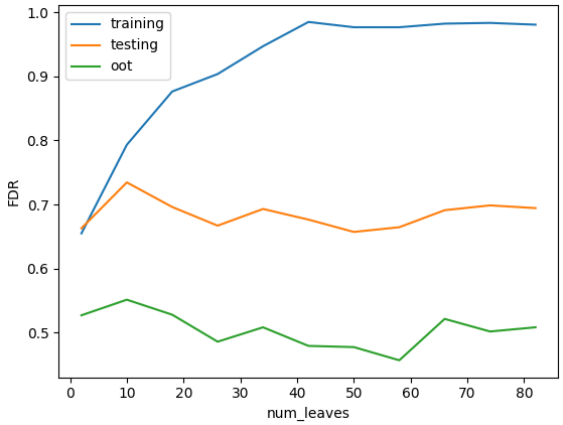

Figure 6: **LGBM overfitting**
By increasing num_leaves, we see the overfitting between training and testing data. Though the decrease in performance is visible on oot data, it is not aggressively decreasing as expected.
Parameters as a function of i $\longrightarrow$
$(num\_leaves = 2 * i)$;
$(max\_depth = i)$;
$(min\_split\_gain = 0)$