Summary

Best Gini: 0.517997

Stand Dev: 0.06114332752194878
Total Submissions: 182

Steps	Explanation	What worked	What didn't work	Wishlist
EDA	**Quick EDA revealed:** - small train set (864 observations); very skewed classes: only 10% fraud cases, 90% non-fraud (legitimate) transactions - large number of variables (no feature name or info provided); features: binary, categorical and float - missing values: only few variables with missing values, no duplicate rows - if_var_68 and if_var_69 most likely transaction amounts (given value distribution); test-dataset contained an additional variable (contract date) - distribution: some variables skewed or log distribution			n/a
Preprocessing/ Feature Transformation	Minimal Preprocessing was necessary given invariance to scale of Random Forest as well as relative cleaniness of data (only few missing values)	**1. Replacing 0's:** a. binary variables: with either 0 or 1, dpending on column mean (1 if >0.5, else 0) b. categorical variables: mode c. float variables: mean 2. Removing var with no information content (only 1 unique value): ""ib_var_12"	**1. Removing Outliers** – this was expected, since Fraud cases can often represent itself as outliers and by removing those we would not train our model on those frauds **2. Scaling Features** - explicitly tried for skewed variables if_var_68 and if_var_69, however this did not impact results. With Random Forest as final model, scaling was not necessary	**1. Binning Categorical Variables** - although no variable names were given this could have been performed, e.g. using scikit learns "KBinsDiscretize" **2. Creating Dummies with Categorical Variables** - relevant for distance-based algorithms (e.g. KNN) to ensure that a value of 4 (Category4) is not read as 4x as big as the value of 1 (category1)
Sampling	**Hypothesis:** Improve the classifier by providing a balanced dataset. Sampling performed at two steps in the process: 1. Sampling before calculating correlation, IV and PSI: this was performed in order to understand correlations and information value of all variables to then conduct feature selection. For feature selection, key features were selected from the unsampled X_train and X_test and entered training unsampled (see, 2. Sampling during model training). 2. Sampling during model training: To prevent data leakage, over-/undersampling should be performed during cross-validation, not before. Therefore, sampling was conducted through a pipeline during training and cross-validation	**1. Oversampling & Undersampling vs. Only Oversampling:** mixed performance of conducting both Oversampling (using SMOTE) and Undersampling (using RandomUnderSampling), before hypertuning the model performed better with only oversampling (to a ratio of 1:1), after hyperparameter tuning using both Oversampling & Undersampling, each with 0.5 sampling strategy (resulting in 2:1 ratio of legitimate to fraud cases) performed best (also tested vs. other sampling ratios) **2. Oversampling with SMOTENC:** although improvement was very minor, a slightly better performance was achieved when using synthetic oversampling technique for nominal and categorical features (treats categorical features differently) **3. Using Sampling during Cross-validation:** following common literature, to prevent data leakage although no actual performance improvement was observed vs. sampling before model training	**1. SMOTE alternatives:** BorderlineSMOTE, KNNSMOTE and ADASYN SMOTE did not provide better performance (Gini between 0.49-0.5x; although tested only during a very small sample of trials and not systematically due to time constraints) **2. class_weight="balanced"** of scikit-learns RandomForestClassifier as alternative to imbalanced-learn's SMOTE, this did however significantly decrease performance	**1. Trying SAMPLING alternatives using smote-variants, which provides a larger number of different smote methods,** e.g. weighting fraud cases differently depending on some criteria, however smote-variants was incompatible with my conda set-up. **2. Research methods to augment the entire dataset** (both fraud and non-fraud cases) since so little training data was available, possibly leading to the severe overfitting
Feature Selection:	**Hypothesis**: improving the classifier by selecting only those features which would best classify fraud and non-fraud cases). However, given RandomForest built in Feature Selection, no significant improvement expected. Features were assessed using correlation (Spearman), IV as well as PSI for a sampled (=balanced) dataset. Based on this, key features were selected.	**1. Filter-based method: feature selection using correlation (Spearman)** with target variable - very quick'n'dirty method **2. Keeping majority of the features:** Classifier performed best when dropping only a very limited number of features (15) and keeping the majority (66 features)	**1. Dropping a large share of features** -> Gini dropped to values below 0.45, indicates that most features contained relevant information and/or that Random Forest did a better job in Feature selection than the filter-based method (using correlation)	**1. PCA** - although this would only work for the numerical (non-binary, non-categorical) features (16 in total), so this would need to be applied alongside other feature selection methods **2. GAN** - Genetic Algorithm **3. Further Filter-Methods:** such as Chi-² Test 4. **Wrapper-based methods:** recursive feature elimination
Model Choice:	**Hypothesis:** Ensemble methods provide superior prediction power for this usecase since they combine multiple 'individual' (diverse) models together for a prediction, thus are more robust and better in fighting the "local minima problem" To obtain a baseline, several models were tried (without any hyperparameters) to compete against each other (Ensemble and non-Ensemble Methods): - Logistic Regression - Random Forest - Decision Tree - ExtraTrees Classifier, out of which **Random Forest performed best**. This was also confirmed in class, so Random Forest was the main model chosen for the classification.	**1. Ensemble Methods -> Random Forest:** ensemble of decision trees, trained via bagging method and thus averaging predictions (fighting the local minima problem). Additionally, it introduces more randomness through searching for the best feature among a random subset of features vs among all features, resulting in more bias and more robustness.	**1. ExtraTreesClassifier** -> did not improve scores **2. Stacking:** Hard Voting implemented (combinations of Random Forest and ExtraTreesClassifier as well as LogRegression with different configurations tested), however due to time constraints not well-developed and optimized, thus it performed similar/not better than only Random Forest (Gini: 0.48-0.5x)	**1. Improve/Tune Voting Classifier** **2. Testing BaggingClassifier** (tbd if it can also do hard voting or only soft voting) **3. Testing Stacking** (using e.g. DESlib or own method), but tbd due to small training set **4. Testing further classification models:** - KNN - Naïve Bayes - Neural Nets - Boosting Methods (XGBoost, GradientBoost, AdaBoost) Not tested due to time constraints and due to general opinion/information shared, that Random Forest (or other bagging methods) would perform best, however it could have been interesting to test if, for example, the Voting algorithm worked better when using bagging methods (for robustness and randomness) with boosting methods, despite boosting methods tending to overfit (and lacking randomness)
Hyperparameter Fine-Tuning	**Hypothesis:** Improving Model performance (precision) with the correct hyperparameters Different combinations of hyperparameters for Random Forest were extensively tested using general understanding of the hyperparamters on model behaviour plus RandomSearch first, then GridSearch.	Best results were obtained with: 1. N_Estimators of 1,100 trees -> averaging results over 1,100 trees 2. Min_samples_split: 3 3. Max_depth: 15 4. Max_features: log2 5. Random-Seed: alternating the random-seed as the very last step on the tuned model provided randomly better (and worse :)) scores -> all other parameters were left on default values 6. RandomSearch and Gridsearch	**1. Bootstrapping:** best results were obtained with bootstrapping=False, which means that for each tree all observations were used. This was most likely due to the small training data, with bootstrapping=True only a sample would have been used (via max_sample), resulting in even less training data at a time.	**1. max_features:** try float instead of default auto, sqrt, log2
Cross Validation	**Hypothesis:** Using cross-validation to average training results and preventing overfitting Using Cross-validation improved model-performance; it was used in a pipeline together with the above mentioned sampling strategy so the model would be trained on a balanced training set.	**1. Cross-validation:** improved model performance, using the following hyperparamters: cv = StratifiedKFold(n_splits=10,shuffle=True)	The following cross-validation hyperparameters/methods didn't impact model results (no change in Gini): **1. Testing different number of splits** during cross-validation didn't impact model results **2. Testing StratifiedKFold vs. RepeatedStratifiedKFold** didn't impact model results	
General/ Evaluation	**Conclusion: T**he model completely overfit during training which could be seen in the evaluation results: Gini-Index=1.0 on the test set vs. 0.5-ish as actual result and an ROC_AUC score of 0.5-0.6 (indicating that the classifier is just slightly better than random guessing). Interestingly, the model still returned best results when the hyperparameters were tuned to low regularization values that would have overfit most models: very high max_depth (15), very little min sample_split (1), etc.). Most likely this was due to the model having only very little training data to learn from.			**1. Data Augmentation** **2. Implementing a cost-sensitive cost-function** **3. Other Libraries** than Scikit Learn (e.g. fastai, weka) or ML Tools, e.g. Dataiku