# Faculty of Electronics and Information Technology - WUT

## EARIN - Data Mining
## Kaggle: Don’t get kicked!

Filippos Malandrakis

## Objective

This dataset contains instances of cars bought in several auctions. For each car, we get an extensive list of its characteristics, auction specifics, and most importantly, its previous and current value in the market. Our aim is to determine whether a purchase is deemed to be profitable("good buy") or not("bad buy", aka "we got kicked") with the highest possible accuracy. We therefore have to deal with a classification problem.

## Feature analysis - engineering

Our dataset has 15 discrete and 19 continuous attributes. Detailed descriptions for each and every one of them can be found in *Carvana_Data_Dictionary.txt*, located on the dataset page(link on code). Our target attribute is *IsBadBuy*, which can take 2 values. We notice that the frequencies of those 2 values are not equal, thus we have an unbalanced dataset. In order to even them out, we downsample the prevailing class(0). Upsampling the weaker class(1) was attempted too(using *SMOTE*), as well as combining these two methods(using *SMOTEENN*), but we ended up with poorer results. We proceed to remove the following attributes:

- *Refid*, *Model* and *Submodel* as irrelevant/too specific features.
- *PRIMEUNIT* and *AUCGUART* as features with too many missing values.
- *VehYear* and *WheelType* as features in highly correlated pairs(*VehicleAge* and *WheelTypeID* respectively).
- *PurchDate* as feature with too many unique values. However, the month in which the auction took place seemed to influence the quality of the purchases, so we extracted them into a new feature, *MonthOfPurchase*.

After reducing our attributes, we advance to imputing both categorical and numerical features(we also convert the latter to ints, for increased homogeneity) using *CategoricalImputer* and *SimpleImputer* respectively. As for the categorical ones, we encode them into dummy attributes(using panda's *get_dummies*). Now that every attribute is numerical, we standardize them so as to follow normal distribution with zero mean and unit variance(using *StandardScaler*). Concluding this stage, we split our

dataset into training - validating and testing, with a ratio of 0.3(for the testing one), using *train_test_split*.

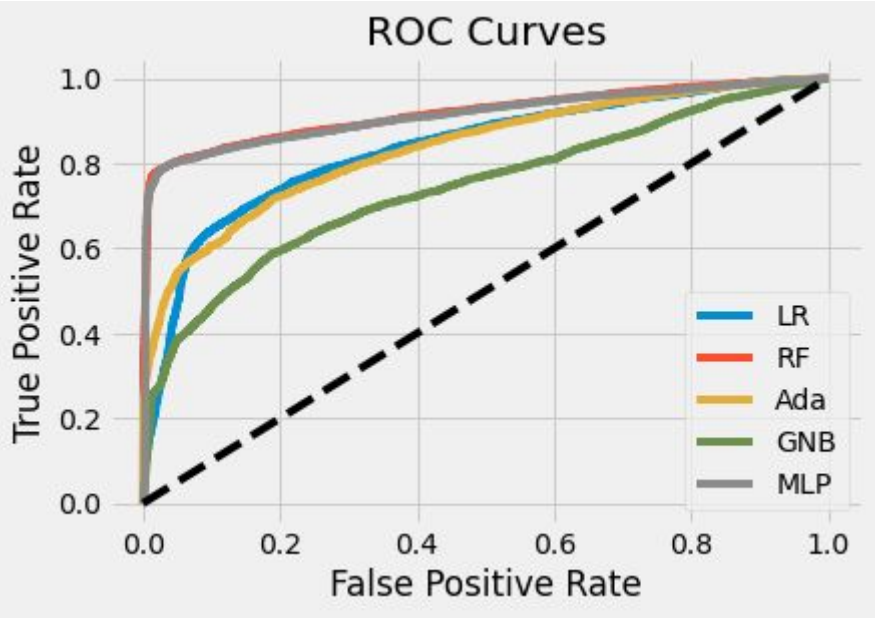
## Training - Validating phase

At this point, we are ready to try our models. For each model, first we apply hyperparameter tuning on its most important parameters(using *GridSearchCV*), and we rate the resulting setups based on the accuracy score(since the dataset is now balanced, accuracy is an eligible metric). For the best setup, we calculate the roc-auc(using *cross_val_score* with *StratifiedKFold* cross-validation model of 10 splits). Afterwards, we will compare the best representative of each model, in the pursuit of the optimal classificator. *For each model, all of its non-mentioned parameters are set to their default values.*

Firstly, we present the best setup for each model, along with its accuracy and roc-auc score.

| Model | Parameter set | Accuracy | Roc-auc |
|---|---|---|---|
| LR | C = 0.001 | 0.7714 | 0.838 |
| RF | max_depth = 20 | 0.8795 | 0.92 |
| Ada | learning_rate = 0.1, n_estimators = 5 | 0.7601 | 0.838 |
| GNB | default | 0.626 | 0.79 |
| MLP | alpha = 0.3334, activation = logistic, solver = adam | 0.881 | 0.916 |

Now, we will attach a graph containing the ROC curve for every model.




It is obvious that Random Forest and MLP are by far the best classifiers for this dataset, delivering very good results. AdaBoost and LR come second with minor differences, and further behind is GNB.

## Testing phase

Prioritising accuracy over roc-auc score, we will choose MLP as our best classifier. We should note here that ensemble voting(by MLP and RF) was also attempted, but it delivered a little bit weaker results. Below we present a plot denoting the results of MLP run over the testing set. Our final accuracy score is **0.87**.