Machine Learning MBS Internal Competition

Executive Summary

This report demonstrates the development of a supervised classification model to predict a binary variable (described only as 'class') using 31 contextless features ('f1', 'f2', …, 'f31').
The development was approached in four parts, often overlapping. First, learners were compared using 'area under the ROC curve' (AUC) on a test partition of the data. Second, features (as-given, transformed, and engineered) were selected using forward/backward stepwise search and Top-N. Third, learner parameters were tuned using grid search. Fourth, logistic regression was used as a meta-algorithm to stack and blend models.
Our top model uses XGBoost and achieved an AUC of 0.80939 on the Kaggle public leaderboard.

Technology used

The exercise was conducted primarily using R. 'Core' R scales poorly because of in-memory and single-core limitations. Because intensive computationally task had to be performed, speeding up R was critical. Microsoft's ScaleR packages (in particular, RevoScaleR) increased R's speed by utilizing multiple CPU cores and perform processing in 'chunks' when using Microsoft's Revolution own algorithms, including Logistic Regression (rxLogisticRegression), Random Forests (rxFastTrees) and Neural Networks (rxNeuralNet). However, XGBoost proved most promising (elaborated further below) which is an algorithm not supported by RevoScaleR, and so more computational power was required. For this we utilised three cloud platforms; Microsoft Azure HDInsight R-Server, Amazon Workspace Services, and Google Cloud Platform.

Data Analysis

The data includes 25,000 training instances (few enough to be kept in a flat-file), each having class of '1' or '0' and 31 features. The data is undescribed, with the class listed as 'class' and the features listed as 'f1', 'f2', …, 'f31'.
The observations appears to be balanced, with 13k rows in class '1' and 12k in class '0'. Most of the features were continuous with range [0, 1] (or a subset). Four appeared to be categorical. Some appeared to be transformations of others (e.g. f1 ~ f20, f2 ~ f25) but were not considered redundant until feature selection.

Model Selection

The provided training set was partitioned into local training and local test sets. A variety of algorithms were compared by the AUC achieved on the local test set to see which might perform better in this exercise. The results are presented below:

Model ID	Model	TRAINING		TEST
		AUC	*RMSE*	AUC
0	**ZERO-R**	0.50	*0.528*	0.50
1	**NAÏVE BAYES**	0.5862	*0.6312*	0.5814
2	**LOGISTIC REGRESSION**	0.6435	*0.5919*	0.6392
3	**DECISION TREE**	0.7650	*0.2950*	0.7223
4	**SVM:** RADIAL|LINEAR|POLYNOMIAL|SIGMOID|LOGIT	0.49- 0.85	*0.40-0.51*	0.50 – 0.73
5	**NEURAL NETWORKS**	0.8690	*0.4576*	0.7507
6	**RANDOM FOREST** (RXFASTTREES)	0.8250	-	**0.7851***
7	**XGBOOST**	0.9829	*0.1172*	**0.7836***

The basic classification model of Zero-R was used as a baseline to serve as reference for future models and determine how well they were performing.

For models 1 to 3 in general their AUC was quite low on their training data (underfitting) which suggested that we needed a boosting algorithm such as XGBoost that could increase the predictive power. The error term was also high suggesting the need of a bagging algorithm (since we could not increase the amount of data) such as Random Forest that could decrease the variance.
For SVM without tuning parameters (i.e. using default R package 'e1071' values), sigmoid function produces poorer AUC than radial, linear, and polynomial kernels (the four default kernels available in e1071). In general, SVM requires a long time to devise and add kernels to find meaningful interactions. In this case due to the high number of variables and hidden relationship without the possibility to use domain knowledge this method was discarded.
Neural Network provided a better AUC but was discarded in favor of Random Forest and XGBoost which provided the best metrics and were considered for the remaining of the analysis. XGBoost can also provide a less complex model by tuning their parameters which could give us flexibility to avoid overfitting in the end.

Feature Engineering

We hypothesized that there exist transformations of, and interactions between, the given features that yield additional information useful for increasing model accuracy. To find them without intuitive context, we created many 'engineered' features and conducted feature selection (discussed below).

Features transformation

- **Capping:** Extreme values of continuous variables were reduced/increased to their 98th/1st quantile.
- **New Scale:** Variables had long decimals that would add more noise than predictive power. To diminish this effect all variables were multiplied by 1.000, 10.000 or a combination of both depending on the median of the values (median < 0.09 by 10,000, otherwise by 1,000) and rounded without decimals.
- **Collapse Categories:** There were 4 different categories but each of them had enough classes by itself. We created a new category that had all possible combinations among them (10 dummy variables).
- **Discretization:** Sometimes numeric data works better when it's separated into bins. We did this with all continuous variables creating 4 bins for each separated by their 25th, 50th and 75th quartile.
- **Binary transformation *(new variables)*:** Values > 0.0001 with value of 1, 0 otherwise.
- **Nonlinear behaviors *(new variables)*:** The following transformations were created for all continuous variables: ***x^2, x^3, sqrt(x), exp(x), 1/(1+x)*** in the hope of finding separations in a new dimension.

Features interaction

- **Double-Interactions:** Only applied if they had a correlation > 0.2. This number was devised by performing multiple models being 0.2 with the highest AUC in XGBoost. It also gave the possibility to diminish the number of features.

Variable	Linear	Polynomial 2	Polynomial 3	Kernel Type
Type 1	x*y	x^2 + y^2	x^3 + y^3	(1+ x*y)^2
Type 2	x-y	x^2 - y^2	x^3 - y^3	(1- x*y)^2
Type 3		(x + y)^2	(x + y)^3	(1+ x*y)^3
Type 4		(x - y)^2	(x - y)^3	(1- x*y)^3

- **Triple-Interactions:** The product of the variables and only applied if they had a correlation > 0.2 among the three.
- **PCA:** New features based on the principal component analysis to reduce the data into lower dimensional spaces that could predict with a higher impact.

Features Selection

Five methods to select the main features that could improve the performance and reduce the training time.

Wrapper Method - In-house Grid Search: After creating all the variables we tested our assumptions in XGBoost running the model with each combination of possible feature transformation/creation. The options were:

- New Scale (*4 options*: No scale, by 1,000, by 10,000 or by median criteria)
- 14 different double-interactions types as discussed above, each interaction had two options; being done or not (that is *2^14 options*).
- Three interactions, binary transformation, collapse categories, discretization and all nonlinear behaviors each had 2 options. Among all of them there were *2^9 options.*

There were **2^25** possible combinations, because of this we could compute only **~200** possibilities (at **10 minutes** each). Most of them were performed through an automated procedure, however only those that gave a higher AUC when they were included alone were considered in the automatic procedure. In the end we reduced the necessary computations to **2^12.** By selecting the best of them we reached an **AUC of 80.46**. The following methods considered only these variables.

Filter Method - Ranking based: Two methods were used. The first selecting the Top-N features based on their independent AUCs on the test set using Random Forest and another based on the Gain Ratio Importance using XGBoost. The main reason to use this method was because of its simplicity. Unfortunately, the accuracy didn't improve the performance for XGBoost (our best model up to this point) but it did for Random Forest with the Top-30:

Top-N by AUC	**20**	**25**	**30**	**40**	**100**	**N/A**
RF	0.7983	0.8021	**0.8045**	0.8044	0.8036	0.8020
XGB	0.7926	0.7978	0.8025	0.8016	0.8036	**0.8046**

Wrapper Method - In-house Forward and Backward Selection: Implemented only for Random Forest since for XGBoost it would have required at least 3 days running non-stop. Forward selection started with a single variable and would be adding new ones only if they improved the AUC getting in the end and AUC of 80.12. For backward it would start with all variables and would deleted them if it improved the AUC ending with an AUC of 80.20. Both scores were lower than TOP-30.

Wrapper Method - Genetic Algorithm: Caret Package provides useful functions to perform genetic algorithms, however the speed is extremely low and we could not use it for more than 6 features at a time and the method had to be discarded due to computational constriants.

In summary, the features selected with the Top-N method gave the best results for Random Forest, however XGBoost outperformed all of them indicating that by cleaning the worst performing features in the first part (**Wrapper Method - In-house Grid Search)** was enough.

Parameters tuning

Two approaches were performed when using AUC as the evaluation metric.

Caret package through grid search: Using method = cv and nfold = 5.

In-house built function to loop through different combinations: Similar to Caret but simpler to implement and much faster to run since no cross validations was considered.

Because of the large amount of combinations possible we first optimized certain parameters and when those were selected we continued for the next sets. All parameters were set at default except eta (0,1) and nround (1000). The steps were:

1. **max-depth** (tree depth) **[default=6]** and **min_child_weight** (minimum sum of *weight* instances for each node) **[default=1]:** Combining the values [4,6,8,10,12,100] and [1,5,10,15,20] respectively. The best combination (**max_depth = 8** , **min_child_weight = 2**) had a ROC of 0.8106. This made sense since a larger max_depth value would overfit and a larger min_child_weight would give a more conservative model likely to underfit. Thus, this combination was not extreme in either case. The AUC increased from 0.8046 to 0.8049.
2. **Gama [default=0]:** Testing the values [0,1,2,3,4] the best combination (gamma = 0.5) had a ROC of 0.8100 and made sense since a larger gamma means that the minimum loss reduction for a partition would be too high making the model simple and prone to underfit. However, the best performance for XGBoost kept at gamma = 0 with AUC 0.8049.
3. **Subsample [default=1]** and **colsample_bytree [default=1]:** Testing the values [0.7,0.8,1] and [0.3,0.5,0.8,1] respectively the best combination (with 0.7 and 0.8 respectively) had a ROC of 0.8108. This level is "just right" since it indicates the amount of data to consider as sample and test, and higher values would overfit and lower would underfit. The AUC increased from 0.8049 to 0.8088.
4. **Regularization via alpha [default=0] and lambda [default=1]:** By testing the values of [0,1,2] and [0,1,2] respectively the best combination gave a jump in the AUC from 0.8088 to 0.8097 using alpha = 1 and lambda = 0. Higher values make the model simpler avoiding overfit and was desired.
5. **Scale_pos_weight [default=1]:** To control the balance of classes, we tested Count(class 0)/Count(class 1) giving a lower/higher AUC 0.8087 which confirmed our previous finding that the classes were not unbalanced.
6. **Learning rate (eta)** and **number of rounds** (number of trees): Lower values for eta requires higher rounds. The right mix to avoid underfit and a slow training process was eta = 0.007 and nround = 3050. AUC increased to 0.8094.

We then repeated the whole process in case some parameters could be optimized further, finding that lambda could be set to 3 instead of 0 and alpha at 0.18 instead of 0 increasing the AUC from 0.8094 to 0.8111. With a higher lambda, the model was less likely to overfit which was desirable. Cross-validation also confirms this.

7. **Cross Validation:** We observed that the best fit with the lower error was at iteration 2105 when lambda = 3, and 1814 when lambda = 0. However, as expected, the lower lambda had more standard error being more prone to overfit:

n-fold = 5	**Lambda = 3**		**Lambda = 0**	
nround	**2105**	**3050**	**1814**	**3050**
AUC	0.8111	0.8083	0.8102	0.8095
AUC std	0.0064	0.00587	0.0078	0.0080
RMSE	0.4224	0.4253	0.4220	0.4254
RMSE std	0.0042	0.0037	0.0047	0.0054

Stacking

With the objective to reduce the risk of overfitting and lowering variance (and with the hope to increase the predictive power, although there's never assurance it will be the case) we performed an stacking ensemble technique. As a first step we built and trained three models, two XGBoost as discussed before (one with lambda = 3 and nround = 2105 and another with

lambda = 0 and nround = 1814) and a RevoCaleR RandomForest (*rxFastTrees*). As a second step we built a meta classifier algorithm using a logistic regression model to perform the blending as can be seen in the image below:

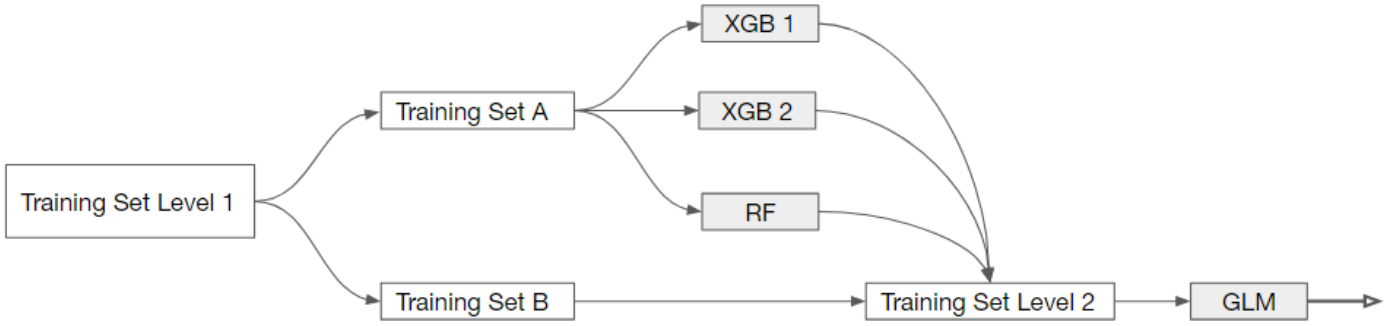

Image 1.

The blender logarithmic model is:

β_0: $Intercept = -2.30527$ β_1: $Coefficient\ of\ XGBoost\ 1 = -0.18081$ β_2: $Coefficient\ of\ XGBoost\ 2 = 4.28480$ β_3: $Coefficient\ of\ RF = 0.49822$

$$\Pr(Class = 0 \mid Model_i Probability) = \frac{\beta_0 + \beta_i * Model_i Probability}{1+\exp(\beta_0 + \beta_i * Model_i Probability)} \Pr(Class = 1 \mid Model_i Probability) = \frac{1}{1+\exp(\beta_0 + \beta_i * Model_i Probability)}$$

And the final results for this model are shown in the table below:

	On training Level 2			**On Test Data**
	XGBoost 1	**XGBoost 1**	**RF**	GLM Blended
AUC	0.8076	0.8093	**0.8093**	**0.8018**

Final Submission Selection

In order to select only 2 submissions for the final competition submission we prioritized those models with high AUC but also who had lower error on new test data to ensure its generalization. On one hand, we prioritized the model who had parameters that increases generalization such as lambda = 3 instead of lambda = 0. On the other hand, by looking at **cross-validation** on the different XGBoost models created (with different parameters) we aimed to select the one that had the lowest error, i.e that achieved higher generalization (comparing each at their optimal iteration), and determine which model was at risk of overfitting (especially since we would be predicting against a yet unseen test data and a private leaderboard). This analysis strengthens our choice for XGBoost with lambda = 3. However, in the end since the stacking model could not improve the predictive power (mainly due to limited data) we didn't choose it as a solution.

Further Investigation

Our approach was limited by computational power. With more computing resources and/or parallelization (e.g. with Hadoop or Spark), we could have:

- Considered even more relationships between features (such as C(3, 31));
- Employed more sophisticated feature selection such as genetic algorithms, Simulated Annealing, or and Caret's Recursive Feature Elimination;
- Performed further parameter tuning to optimize the XGBoost model even more; and
- Tested more stacked model combinations, such as by blending XGBoost, RandomForest, Neural Networks, SVMs, and KNN algorithms together.