ML Kaggle Competition

Al-A-H data prophets

1 Feature Engineering

1.1 Feature Transformation

1.1.1 Label Encoding

After we have finished doing the Explanatory Data Analysis, we have started by doing a label encoding on the categorical variables. By assigning to each unique category a number (e.g. *Time_of_Day*: "afternoon" $\rightarrow$ 0, "evening" $\rightarrow$ 1, "morning" $\rightarrow$ 2), those categories can be used for machine learning models that require numerical input, are simple to implement, and conserve memory. This initial step can allow us to work on more data to build our models better.

1.1.2 Data Imputing

The data imputing process involves filling in missing values in a dataset. In order to try to fill the data with more precise values as possible we used a custom version of KNN algorithm, since it preserves the relationships and patterns in the data.

Through an iterative process, it handles both numerical and categorical columns by applying KNN regressors for numerical data and KNN classifiers for categorical data. The imputation process begins by initializing missing values and iteratively imputing each column, treating other columns as predictors, while refining the imputations across multiple epochs. It dynamically selects the best number of neighbors for KNN based on RMSE (numerical) or F1-score (categorical). During this process, it incorporates preprocessing steps such as scaling and encoding to standardize data for KNN computations, ensuring robust and accurate imputations for mixed-type datasets.

1.1.3 Adapting quantitative features

We plotted the distribution for each feature and found that some features, such as Engagement_Score, had a right-skewed distribution, with positive skewness defined as:

$$\text{Skewness} = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s} \right)^3 \tag{1}$$

This led to the use of mathematical methods to address the skewness. Specifically, we applied the following transformations:

$$x' = \sqrt{x} \qquad \text{(Square Root Transformation)} \tag{2}$$

$$x' = \ln(x+1) \qquad \text{(Logarithmic Transformation)} \tag{3}$$

to Engagement_Score, Reviews_Read, Items_In_Cart, and Socioeconomic_Status_Score. For Age, we implemented:

$$x' = \frac{1}{x + 0.1} \tag{4}$$

to address the right-skewed distribution while enhancing the importance of differences between smaller values.

2 Model Tuning and Comparison

2.1 Data preprocessing

2.1.1 Train-Test Splitting

Since the data is unbalanced (there are much more 1s in the Purchase column rather than 0s), we firstly tried to perform an oversample method called **SMOTENC** (Synthetic Minority Oversampling Technique for Nominal and Continuous features). For continuous features, SMOTENC creates synthetic samples by interpolating between existing minority-class instances. For categorical features, it uses the mode (most frequent value) of the k-nearest neighbors to ensure that synthetic data respects the discrete nature of the categories. We registered an important increase in our results. Another option to address classes imbalance was to set class weight parameter for our model. It appeared to be even more efficient comparing to SMOTENC oversampling. So, we used class_weight afterall.

2.2 ML models tuning with cross-validation

2.2.1 Approach

To find the best strategy, we used different approached and implemented several models: Logistic Regression, Linear Support Vector Machine (Linear SVM), Radial basis function network (RBF) SVM, Random Forest, and AdaBoost.

The performance of these models was evaluated using two approaches:

- *Approach A (No PCA)*: Uses the original standardized features without dimensionality reduction.
- *Approach B (With PCA)*: Applies PCA to reduce the dimensionality of the feature set while retaining 80% of the variance.

2.2.2 Parameter Tuning and Overfitting Prevention

For Logistic Regression and Linear SVM, the regularization parameter C was tuned over six values ([0.001, 0.01, 0.1, 1, 10, 100]) to optimize model performance.

For RBF SVM, both C and the gamma parameter for the RBF kernel were tuned, exploring combinations to strike a balance between underfitting and overfitting.

Random Forest was optimized by tuning the number of estimators (*n_estimators*) and tree depth (*max_depth*) to balance model complexity and predictive accuracy. Similarly, for AdaBoost, the number of estimators and learning rate (*learning_rate*) were adjusted to maximize the efficiency of the boosting process.

2.2.3 Results

The results show that **without PCA, Random Forest achieved the highest F1 (macro) score (0.7396)**, followed by Linear SVM (0.7135) and RBF SVM (0.7112).

With PCA, Linear SVM performed best (0.7280), demonstrating its effectiveness in handling reduced-dimensional data, while Random Forest and RBF SVM experienced a drop in performance, likely due to the loss of important feature interactions caused by dimensionality reduction.

2.3 NN model tuning with cross-validation

2.3.1 Parameter Tuning and Overfitting Prevention

The neural network had various parameters to estimate during cross-validation. To better capture non-linear dependencies, we used activation functions such as ReLU and Tanh. We also experimented with different numbers of hidden layers and neurons in them. One hypothesis was that we could compress the features from the input layer in the first hidden layer (16 neurons), effectively performing something similar to dimensionality reduction but within the neural network. However, cross-validation revealed that it was actually better not to constrain the model with such a bottleneck and instead use more neurons (64) in the first hidden layer. We also applied dropout to prevent the model from overfitting.

To minimize the loss function, we used the Adam optimizer with a weight decay parameter. This parameter implements a regularization technique that adds a penalty to the weights of the model during the optimization process, further reducing the risk of overfitting. Given the size of the training dataset and sufficient computational resources, a batch size of 128 was considered a reasonable choice. Since the initial data had an imbalance in class distribution, we adjusted the class weights in the Binary Cross-Entropy Loss for each batch: a weight of 1.0 was assigned to class '0', and for class '1', the weight was set to the ratio of the number of '0' to the number of '1' in the batch.

Results: achieved mean macro F1-score across 5 folds on train data was equal 0.7252, showing higher performance, than all other models except for Random Forest