Report Machine Learning in Finance

Part 1: To select a model, first we conducted initial tests on each option using the default hyperparameters and measured their accuracy. We selected Gradient Boosting as the algorithm to use based on its preliminary performance. Additionally, to confirm our choice we sought sources such as [2], that highlights the efficacy of gradient boosting in comparison to other methods, particularly when confronted with large, imbalanced data sets in binary classification scenarios (such as the one at hand). A thorough explanation of how Gradient Boosting works can be found in [6]. The steps taken to implement and train the algorithm are enumerated below.

1. We imported the data into Python and preprocessed it. Specifically, we replaced all the expense columns with a single column containing the sum of all expenses.
2. We assigned the binary target to be the vector **y** and all the other columns to form the matrix **X** and split the full dataset into the train and test sets, using an 80/20 proportion.
3. We performed the hyperparameter tuning with the train set using grid search and k-fold cross-validation with the number of folds k=5. The hyperparameter grid that we used to optimize this algorithm consists of 6 parameters (n_estimators, min_samples_leaf, min_samples_split, max_features, max_depth, and learning_rate) and is found in table 1.
4. We then trained the model using the whole train set and the best hyperparameters selected. Using this model, we generated predictions using the **X test set** and compared them with the **y test set** to measure the accuracy.
5. Finally, we retrained the model using the whole data set, and the hyperparameters selected previously in order to generate predictions for the prediction data set.

Part 2: Our reason for replacing the expense columns with a single column containing the sum of all expenses was to simplify the data and reduce dimensionality, and also to make it easier to manage for the gradient boosting algorithm, as decision trees are not suitable for additive data [1]. The 80/20 split of the data into train and test sets was chosen because it is standard practice, as stated in [3]. We used Grid search to systematically explore different combinations of hyperparameters and used k-fold cross-validation to avoid overfitting and reduce the variance of the generalization error estimate, as recommended by [4]. As for the hyperparameter grid, we used [5] and [7] as initial reference for the values, and then considered each case specifically. For $max\_features$ we included both $sqrt$ and log_2 as the options to help reduce overfitting. For the number of estimators, we wanted to find the right balance between computational efficiency and robustness, so we included 100 and 200 as the values in the grid. The values we chose for the learning rate were motivated by [5]. For $min\_samples\_leaf$ and $min\_samples\_split$, we included values in the grid to explore different alternatives that offer tradeoffs between complexity (lower values) and robustness (higher values), to select the ones that better adjust to this type of problem. For max_depth we included values in the grid to explore options from more shallow trees ($max\_depth = 10$) that help prevent overfitting but also are sometimes unable to capture complex relations in the data, and deeper trees ($max\_depth = 30$) that are better able to capture more complex patterns in the data. As the final step, we retrained the model using the whole dataset to give the model as much information as possible to generate predictions for the prediction dataset.

Conclusions:

- The optimal hyperparameters selected using k-fold cross-validation were: n_estimators=200, min_samples_leaf=1, min_samples_split=2, learning_rate=0.1, max_depth=30, max_features = log2.
- Our gradient-boosting predictive algorithm achieved an out-of-sample accuracy of 86.05%.
- Further improvement of the predictive accuracy of this algorithm could be achieved by widening the grid and increasing its granularity, but at the cost of more computational resources.

Appendix

Table 1: Grid Search hyperparameters for Gradient Boosting

Hyperparameter	Grid
Number of estimators	{100, 200}
Learning rate	{0.05, 0.1, 0.15}
Max depth	{10, 20 ,30}
Minimal samples split	{2, 4, 8}
Minimal samples leaf	{1, 3, 5}
Max features	{sqrt, log2}

References

[1] M. Hakan Akyüz. Trees and ensembles, 2023. Lecture slides for Machine Learning in Finance.

[2] Iain Brown and Christophe Mues. An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, 39(3):3446–3453, 2012.

[3] Aurelien Geron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly, 2nd ed. edition, 2019.

[4] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[5] Aarshay Jain. Complete machine learning guide to parameter tuning in gradient boosting (gbm) in python, 2022.

[6] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 12 2013.

[7] Omar El Yousifi. Hyperparameter tuning - gradient boosting, 2021.