Kaggle Competition Report

Team: GPT Chasers

Data Inspection

The data contains information collected about students from 7 districts in the US, including a target variable representing scores from a standardized test and 55 predictive features. Given that the target variable is numerical and continuous, our objective is to forecast these scores based on the 55 features, framing this challenge as a regression-based prediction task.

We first inspected the meaning and type of features. `SEQN` denotes the unique student ID, which is not useful in predicting the performance score, and should be dropped in the training data. `extracurricular` is an integer (discrete) numerical feature ranging from 1 to 10 denoting the scores on a given extracurricular, and `self_eval` and `teacher_eval` are both integer (discrete) numerical features ranging from 1 to 5, representing the self-evaluation score and teacher evaluation score by the student respectively. `district` is also an integer feature ranging from 1 to 7 that represents the districts, but should be regarded as categorical since there is no inherent order among the values. Finally, there are 50 student retention predictors that are continuous and numerical, represented in the form of `SRP_*number*`.

Missing values can significantly impact prediction tasks in that it could reduce the model performance. We examined the dataset for any null values across all features and were fortunate to find none, thereby eliminating the need for imputation. Also, we checked for duplicated entries and results showed none, ensuring the integrity of our dataset.

It's also essential to inspect the covariate shift between the training dataset and the test dataset to ensure that our model accurately captures the underlying patterns in the training data. We plotted the feature distributions of both datasets using pairplot and confirmed that the feature distributions looked similar. Additionally, we observed that the continuous features exhibit a near-normal distribution, while the discrete features are evenly distributed, indicating an absence of significant skewness in the data.

To sum up, the data is pretty clean, with no missing values and duplicates, no covariate shift, and little skewness.

Data Preprocessing

In this section, we developed a pipeline in which the 50 SRP features are standardized, `district` one-hot encoded and `SEQN` dropped. Note that we did not standard-scaled the 3 numerical features denoting scores, since they are ordinally discrete and standard scaling would lose this natural information.

The primary reason why I used a pipeline instead of manually preprocessing is to prevent data leakage. For instance, directly feeding the preprocessed data into cross-validation will introduce data leakage, as the test fold is actually not processed identically to the training set,

where the mean and variance used for scaling are based on the whole data instead of the training data, and so for the one-hot encoder. To avert this, we included a pipeline in the cross-validation functions, ensuring that for each fold the train and test set are treated in the same way.

Model Selection

In this section, we initiated our analysis by selecting a handful of baseline models that have a proven track record of strong performance on regression tasks, and compared their 5-fold cross-validation scores on R2 using default parameters: SVM, Random Forest, XGBoost[1], LGBM[2], Catboost[3], and MLP. XGBoost, LGBM, and CatBoost are all gradient boosting methods ensembling decision trees, each with distinct advantages: XGBoost is robust with sparse data and missing values, LGBM is faster with large datasets with leaf-wise growth strategy and histogram feature splitting, and CatBoost excels in directly processing categorical data and reducing overfitting with its ordered boosting. The MLP regressor in sklearn is a fully-connected feed-forward neural network, and is adept at capturing the nonlinearity in large dataset prediction tasks. Note that we controlled the random states at 42 for all baseline models to ensure reproducibility.

We chose Catboost and MLP, which have the top 2 cross-validation score on r2, for hyperparameter tuning.

Model Tuning

In this section, we aimed to tune the hyperparameters to achieve better prediction performance. Instead of using Grid Search which searches over all possible cartesian products of the provided hyperparameter grids, we implemented Optuna[4] which is much more efficient in parameter optimization. Optuna is a Bayesian optimization framework that automatically searches for the best hyperparameters using a probabilistic model. Compared to Grid Search, Optuna is more computationally efficient and requires fewer trials to find the optimal settings.

While determining the hyperparameter grid, I consulted the official documentation to understand the meaning, scope, and default settings of each hyperparameter. Additionally, I reviewed several past Kaggle competitions to gain insights into the practical and effective ranges of values for these parameters.

Selecting a proper objective function for model tuning presents as a rather tricky issue. Traditionally, we should split the dataset, train the model on the training set, and use RMSE or

---

[1] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[2] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30.

[3] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. Advances in neural information processing systems, 31.

[4] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 2623-2631).

R2 on the test set as the tuning objective for regression models. However, this would potentially introduce overfitting since the model would only be tuned on a single pair of train-test split. Therefore, we utilized 5-fold cross-validation score on R2 as the objective for hyperparameter tuning.

Tuning the hyperparameters is an iterative task that requires experience and intuition. For Catboost, we first tuned the hyperparameters for single trees with a low iteration (number of trees built) to control the time, and then tuned the learning rate and number of iterations with other hyperparameters fixed to the best params found in the first round. For MLP, we first controlled the max iteration at a low level, and after several trials, it was clear that using tanh for activation, Adam for solver and an adaptive learning rate would lead to the best result. As for hidden layers, since the feature distribution is quite balanced and our dataset is not large enough, we only tried a single hidden layer and two hidden layers, and selected a few integers around $\sqrt{\#\ features}$ = 8 [5]for the size of the last hidden layer. Note that the size of the layers cannot exceed the # features and should be decreasing progressively, by experience. Similar to Catboost, we tuned the iteration parameter with other hyperparameters fixed to the best parameters found in the first round.

Note that we controlled the random state of both the model and cross-validation at 42 in this section. Plus, the detailed tuning process is not fully shown in the notebook, since most of the work was done online, and we only kept the code and the best results.

Model Prediction

After model tuning, the best cross-validation score of Catboost was improved from 84.62% to 85.97%, and MLP rose even more, from 81.05% to 87.38%. To further improve the prediction performance, we experimented with bagging models with varying random states, as well as bagging both the MLP and Catboost models. Note that here we utilized BaggingRegressor[6], which is pretty similar to Random Forest, bootstrapping the dataset for training and aggregating (taking the average of) the results, only that the BaggingRegressor employs custom models instead of decision trees

To generate predictions, we began by fitting the preprocessing pipeline to the entire training dataset, ensuring consistency in the transformation of both the training and test datasets. Subsequently, we trained the selected models on the complete training data, and predicted the target variable on the test dataset using the trained models.

Teamwork

Xingjian Liu: Did the whole Kaggle project including training and writing the final report. Note: In our group, my role is to work on the Kaggle project, while the other two members (Hongxuan An & Yi Liu) focus on the open-ended investigation project.

---

[5] https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev/

[6] Breiman, L. (1996). Bagging predictors. Machine learning, 24, 123-140.