Kaggle Competition Report

Haiming Li, Shuxian Chen, Witty Wen

December 3, 2023

1 Preprocessing

As the foundation of our model development, the processing of data is a pivotal step. Our process begins with meticulous data preparation and transformation:

- **Data Loading**: Load data from CSV files
- **Data Cleaning**: Drop column 'SEQN'.
- **Feature Engineering**:
  - Combined 'district' variable into two groups: {1,3,5,7} and {2,4,6}, treat as categorical variable.
  - Treat 'self_eval' and 'teacher_eval' as ordinal variables.
  - For each 7-day period from 'SRP_1' to 'SRP_49', calculate mean, standard deviation, minimum, maximum, interquartile range, and linear trend.
  - For all 50 days, calculate mean, standard deviation, minimum, maximum, interquartile range, and linear trend.

2 Methods

In our submission, we tried two models: XGBoost model and Neural Network model. And at the end, in order to better optimize our model, we stack the two models according to the weights we set.

2.1 eXtreme Gradient Boosting

For the optimization of the XGBoost model, we perform hyperparameter tuning using Randomized Search with Cross-Validation scoring strategy due to the extensive parameter space. This method efficiently explores a wide range of possibilities and is particularly effective in finding the best hyperparameters in a large search space.

Hyperparameters are selected from:

- `n_estimators`: From 300 to 400, increment by 1. This parameter defines the number of boosting rounds or trees to build.
- `max_depth`: From 3 to 15, increment by 1, controlling the maximum depth for each trees.
- `min_child_weight`: A log scale of 1000 numbers from 0.01 to 0.5, influencing the decision of making a new tree split.
- `gamma`: A log scale of 1000 numbers from 0.3 to 20.0, acting as a regularization parameter.
- `learning_rate`: A log scale of 1000 numbers between 0.05 and 0.5, determining the step size at each iteration while moving toward a minimum of a loss function.
- `subsample` and `colsample_bylevel`: Both from 0.7 to 0.9, increment by 0.01, these parameters manage the subsampling of the dataset and the subsampling of features, respectively.
- `reg_alpha` and `reg_lambda`: Spanning a log scale of 100 numbers from 0.00001 to 40, they are L1 and L2 regularization terms on weights, which can help prevent over-fitting.

The random search chooses 300 different combinations of hyperparameters and scores them with 10-fold cross-validation. This setup ensures a robust search through the hyperparameter space while maintaining a balance between computational efficiency and thoroughness. During the search, we used the mean squared error as the scoring metric. This choice aligns with our objective to minimize the error, making the model's predictions as accurate as possible. Upon completion of the random search process, the best estimator was saved for later usage.

This comprehensive approach, leveraging random search process with a well-defined and extensive hyperparameter grid, was instrumental in fine-tuning our XGBoost model, thereby enhancing its performance and reliability for our predictive tasks.

2.2 Feedforward Neural Network

For the development of our neural network model, our approach involved a structured and meticulous configuration of the model's architecture and training parameters. Unlike traditional methods that might require extensive hyperparameter tuning, our focus was primarily on designing a robust and versatile neural network architecture and an efficient training regimen.

- **Layer Configuration**:

Layer	Input	Output
Fully connected Layer 1	100	256
Activation Layer 1	256	256
Dropout Layer 1	-	-
Fully connected Layer 2	256	512
Activation Layer 2	512	512
Dropout Layer 2	-	-
Fully connected Layer 3	512	256
Activation Layer 3	256	256
Dropout Layer 3	-	-
Fully connected Layer 4	256	128
Activation Layer 4	128	128
Output Layer	128	1

- **Activation Function**:
  - The model employs an Exponential Linear Unit (ELU) with an alpha of 0.6, introducing necessary non-linearity to the learning process. Applied after each fully connected layer.
- **Dropout Regularization**:
  - A dropout layer with a rate of 0.3 is incorporated to combat over-fitting, randomly zeroing a fraction of input units during training updates.
- **Weights Initialization**:
  - Weights of each fully connected layer are initialized using the Xavier normal method, ensuring appropriate scaling. Biases are set to a constant 0.0.

For the optimization of the neural network model, we selected 20% of the data set as the validation set to avoid over-fitting. We also set up the neural network training process with several key components, each playing a vital role in the model's learning and performance. The configuration is as follows:

- **Training Setup**:
  - **Model**: The neural network architecture described previously.
  - **Max Epochs**: Training is set for 282 epochs.
  - **Batch Size**: A batch size of 200 is used.
- **Optimizer Configuration**:
  - **Optimizer Type**: The `Adam` optimizer is employed.
  - **Learning Rate**: Set to 0.00018.
  - **Weight Decay**: A weight decay of 0.01 is used.
  - **AMSGrad**: The AMSGrad variant is enabled.
  - **Foreach**: The `foreach` option is set to `True`.
  - **Betas**: Coefficients used for computing running averages of gradient and its square, set to (0.9, 0.999).
  - **Epsilon**: Term added to the denominator to improve numerical stability, set to 1e-8.
- **Training Data Iterator**:
  - **Shuffle**: The training data is shuffled every epoch.

This rigorously detailed and thoughtfully orchestrated setup of the neural network training regimen significantly elevates the credibility of our models. Each component of the configuration has been selected and tuned to optimize the model's performance and reliability.

2.3 Stacked Model

According to a recent study, stacking, a method of combining the output of different models, can significantly improve model performance.[1] The following pseudo-code demonstrates the process of blending predictions from an XGBoost model and a Neural Network model based on their respective cross-validated R^2:

```
weight = xgb_cv_r2 / (xgb_cv_r2 + nn_cv_r2)
xgb_pred = get_predictions(xgb_model)
nn_pred = get_predictions(nn_model)
stacked_pred = weight * xgb_pred + (1 - weight) * nn_pred
```

3 Result and Discussion

Our results from the stacked model approach were enlightening and underscored the synergy between the XGBoost and Neural Network models. By combining these two robust methods, we achieved an R^2 value that surpassed the performance of each individual model. This synergy is likely due to the complementary strengths of the models: XGBoost's ability to handle structured, tabular data with efficiency and the Neural Network's prowess in capturing complex, non-linear relationships.

3.1 Decision Rationale

The decision to employ a stacked model approach was driven by several key considerations:

- **Complementary Strengths:** We recognized that XGBoost excels in handling structured data and can efficiently process various types of variables. Conversely, Neural Networks are adept at identifying complex patterns and interactions in the data. By combining these methods, we aimed to capture a broader spectrum of data characteristics.
- **Performance Enhancement:** Preliminary tests with each model individually showed promising results. However, neither model alone could achieve the level of accuracy we targeted. By blending their predictions, we aimed to enhance overall prediction accuracy.

3.2 Performance Analysis

The stacked model exhibits an R^2 value of 0.88788, which, albeit marginally, surpasses the individual R^2 value of the XGBoost (0.88585) and Neural Network (0.88589) models. This slight increase substantiates our initial hypothesis that an amalgamated modeling approach can capitalize on the unique strengths inherent in each individual model, leading to a more precise predictive performance. The results, therefore, provide empirical evidence supporting the efficacy of model blending techniques in enhancing predictive accuracy, even when the observed improvements in statistical metrics are relatively minor.

4 Submission and Contributions

All the codes created and used in this project have been uploaded to GitHub, available at the following link: https://github.com/haiming12138/stats415_project/tree/main/kaggle. The Kaggle submission account name is hml12138 and this report is evenly contributed by each author.

References

Soni, Brijesh. "Stacking to Improve Model Performance: A Comprehensive Guide on Ensemble Learning in Python". Accessed: 2023-11-27, May 2023. https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28.

[1] Soni.