COMP4432 Individual Assignment

Name: Wang Yuqi
Student ID: [redacted]

Abstract

In this report, I have documented my thought process and experience crafting my solution for the Kaggle challenge *Don't Overfit! II*, as well as any insights gained along the way. In each section, I will discuss my observation and reasonings, as well as how they lead up to the subsequent improvements, thereby detailing a natural procedure on how I reached my final solution.

1. Exploratory Data Analysis

The *Don't Overfit! II* challenge presents us with a training dataset of 250 samples and 300 features. At first glance, this seems like an impossible task: fitting a function with data points < number of axes leads to infinitely many solutions. Therefore, any naive solution that attempts to directly fit a baseline model will not work.

This inherent constraint poses great challenge in preventing overfit. Three approaches naturally emerge from this: 1) simpler models 2) regularization 3) feature elimination. All of my subsequent iteration of improvements will be centered around these three main ideas.

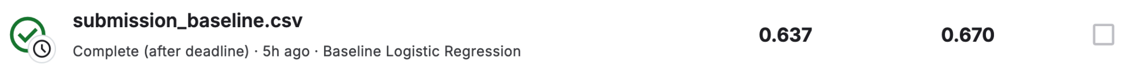
1.1 Baseline Method

For my baseline model, I chose simple logistic regression model. For the regularization term, I consider *L1* to be superior considering that it provides greater sparsity to our model weights, as it penalizes the absolute values of weights, effectively driving some weights down to exactly 0, making it a stronger candidate given the problem's constraints. My hypothesis was justified by a greater cross-validation accuracy when using *L1* regularization instead of *L2*.

The baseline model scored 0.65 in accuracy in cross-validation, yet it unexpectedly scored just 0.47 accuracy on Kaggle, meaning that it is no better than random guessing.

Upon investigation, I quickly noticed that out of the 250 samples, 90 were negative samples and 160 were positive samples. This imbalance could introduce bias into our models, as the model will get higher training accuracy scores merely by predicting positive samples with higher accuracy, hence requiring resampling.

I wrote a python script named `clean.py` that up-samples the negative data points, creating a balanced dataset of 160 positive samples vs. 160 negative samples. Additionally, I noticed that, while most features are well normalized and centered, some were slightly off. Hence, I further normalized these samples by subtracting their mean and dividing by their standard deviation. After normalization, the largest absolute value of mean and standard deviation went from 0.19 -> 0.00 and 1.118 -> 1.000 respectively. The new baseline model re-trained on this balanced dataset yields a much better public score of 0.670 accuracy on Kaggle.



submission_baseline.csv
Complete (after deadline) · 5h ago · Baseline Logistic Regression
0.637 0.670

1.3 Bayes Optimization on Baseline

To understand the full potential of this baseline model, I went a step further and perform Bayes optimization on the inverse regularization term C. The resulting cross-validation accuracy is plotted against C.

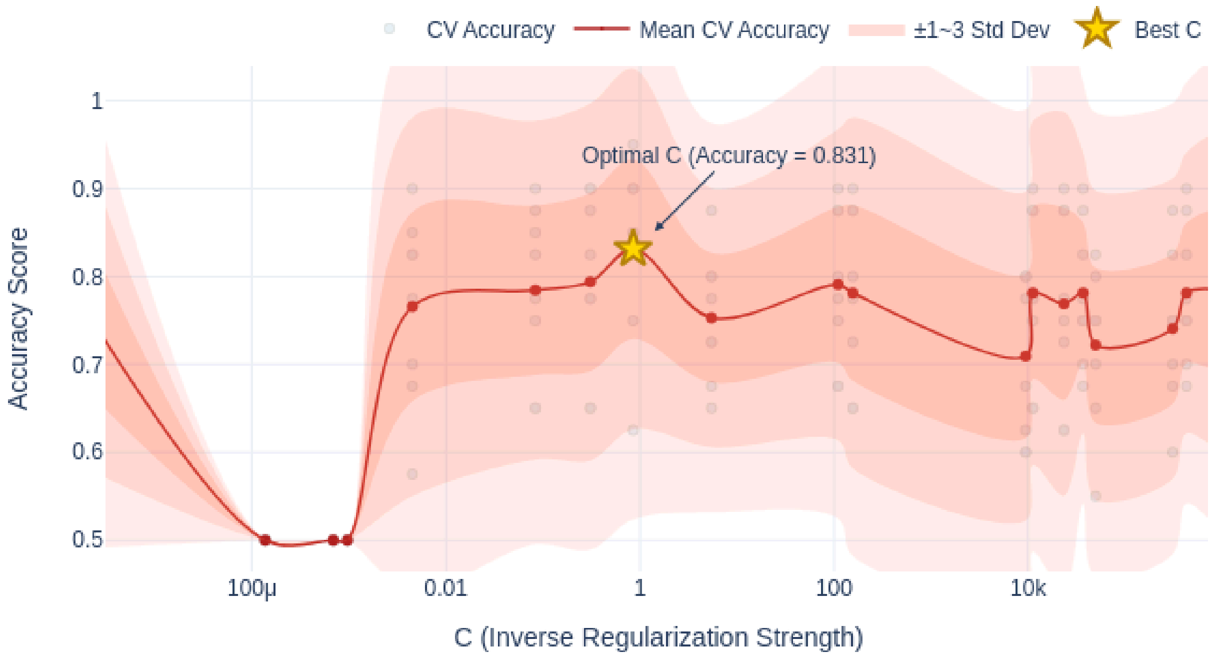
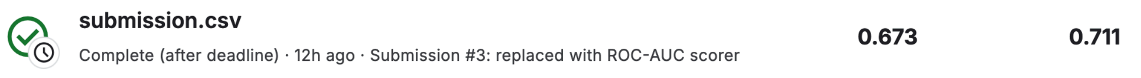


Figure 1: Bayes Optimization on inverse regularization strength.

As shown in the figure 1, the Bayes hyperparameter search gives an optimal C value of approximately 0.85, which yields a cross-validation accuracy of 0.831. I then refit the model to the entire dataset (it was originally fitted to just the $k - 1$ folds) and submitted to Kaggle. As expected, the accuracy score improved by a large margin, going from the $0.670 \to 0.711$.



submission.csv
Complete (after deadline) · 12h ago · Submission #3: replaced with ROC-AUC scorer

0.673 **0.711**

2. Obtaining Insights

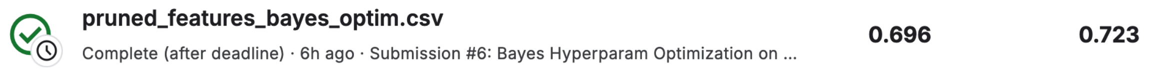
2.1 Observe Model Weights

Now that I have created a baseline model with descent starter accuracy, the natural follow-up question I asked myself is: *out of all features, how many actually contributed to the final prediction?* To answer this question, I counted the number of coefficients equals to 0 in the baseline logistic regression model:

```
print(f'coefficients equal to 0: {np.sum(estimator.coef_ == 0)}')
```

The output shows that 180/300 coefficients (not including intercept) is exactly zero. This indicates that, only 120 features were actively contributing to model prediction. This natural next step to this discovery is to eliminate these unused features and retrain the Bayes optimized logistic regression model on the pruned features.

The retrained model obtained an slightly higher accuracy score of 0.85 on stratified cross-validation, and a slightly higher public score on Kaggle ($0.711 \to 0.723$).



pruned_features_bayes_optim.csv

Complete (after deadline) · 6h ago · Submission #6: Bayes Hyperparam Optimization on ...

0.696 0.723

Main Insight: Heuristically, it can be assumed that there exists a large number of irrelevant features which we can safely eliminate. However, directly basing our feature elimination strategy on the coefficients of the lasso logistic regression might cause *information leakage* due to the feature selection process itself inherently based on the entire dataset, resulting in over-confident estimation during Bayes optimization. Therefore, we need a different approach when it comes to feature elimination.

2. Feature Elimination

2.1 Multicollinearity

When considering alternative strategies to feature elimination, the first idea is to remove highly correlated features (i.e., multicollinearity), which negatively affect linear models.

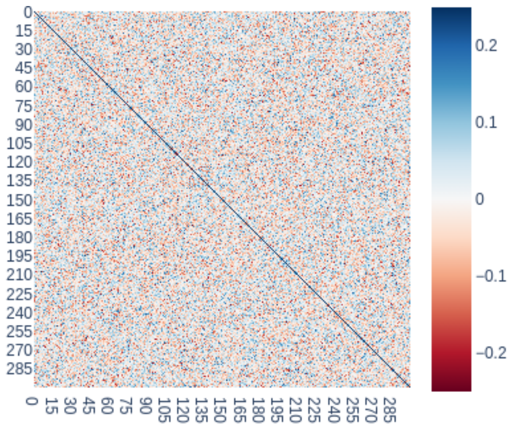


Figure 2: Correlation matrix of the 300 features

To identify linearly dependent variables, I wrote a python script that computes a correlation matrix. As shown in figure 3, even when clipping the upper and lower bounds of the correlation value at $[-0.25, 0.25]$, correlations are still sparse and weak. The top highest R^2 among all features is merely 0.298. This meant that there does not exist pairs of features strongly dependent enough that justifies elimination.

2.2 Weight of Evidence (WOE) and Information Value (IV)

An alternative approach is to compute the information value provided by each variable. The Information Value (IV) summarizes the overall predictive power of the predictor X, with high values indicating stronger predictive capacity:

However, since information value only works on categorical variable, we need to discretize our continuous variables into several bins $B = \{B_1, B_2, ..., B_n\}$. After which, we can compute the IV score as follows:

$$\mathrm{IV}(X) = \sum_{i=1}^{N} \left(\frac{G_i}{G} - \frac{B_i}{B} \right) \cdot \mathrm{WOE}_i$$

Where WOE is the weight of evidence, measuring the log ratio of the distribution of positive to negative samples in each bin:

$$\mathrm{WOE}_i = \ln\left(\frac{G_i}{G} \div \frac{B_i}{B} \right) = \ln\left(\frac{G_i \cdot B}{B_i \cdot G} \right)$$

Python Implementation

```python
def calc_woe_iv(feature, target, bins=10):
    df = pd.DataFrame({'feature': feature, 'target': target})
    df['bin'] = pd.qcut(df['feature'], q=bins, duplicates='drop')

    bin_stat = df.groupby('bin', observed=False)['target'].agg(['count', 'sum'])
    bin_stat.rename(columns={'count': 'total', 'sum': 'bads'}, inplace=True)
    bin_stat['goods'] = bin_stat['total'] - bin_stat['bads']

    # distribution calc
    total_goods = bin_stat['goods'].sum()
    total_bads = bin_stat['bads'].sum()
    bin_stat['dist_goods'] = bin_stat['goods'] / total_goods
    bin_stat['dist_bads'] = bin_stat['bads'] / total_bads

    # prevent division by 0
    bin_stat['dist_goods'] = bin_stat['dist_goods'].replace(0, 0.0001)
    bin_stat['dist_bads'] = bin_stat['dist_bads'].replace(0, 0.0001)
    bin_stat['woe'] = np.log(bin_stat['dist_goods'] / bin_stat['dist_bads'])
    bin_stat['iv'] = (bin_stat['dist_goods'] -
    iv = bin_stat['iv'].sum()
    return iv, bin_stat.reset_index()
```

Results

feature_id	IV
33	0.792611
91	0.599823
117	0.582456
116	0.522028
235	0.506087
...	...
20	0.050634
54	0.047169
22	0.039736
27	0.031903

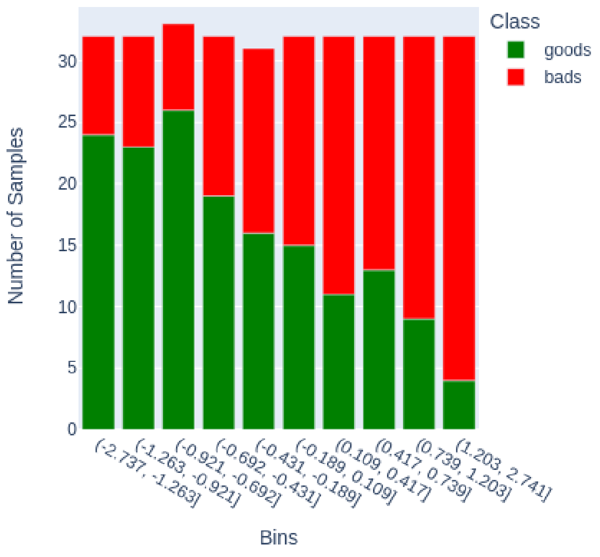
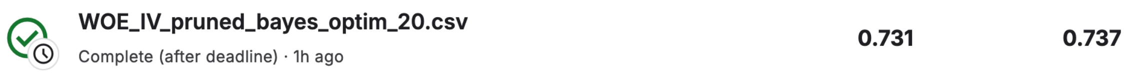


Table 1: **Left Table**: the list of features ranked by IV in descending order. **Right Figure**: the 10 bins of the feature with highest IV score; *good* represents postiive sample, and *red* represents negative samples.

By selecting the top 20 features with greatest IV score, we successfully boost the public prediction accuracy from $0.723 \to 0.737$, and the private accuracy score from $0.696 \to 0.731$.



WOE_IV_pruned_bayes_optim_20.csv
Complete (after deadline) · 1h ago

WOE_IV_pruned_bayes_optim_20.csv	**0.731**	**0.737**