IE 343 Final Report

IE 343: Statistical Machine Learning and Its Applications

Industrial & Systems Engineering, KAIST, Spring 2022

20190849 Hanbit Lee

Introduction

This report will explain the classification model I made for predicting whether a person's income is below or above 50k based on demographic and employment related data. I used Light Gradient Boosting Machine as my main model and also adopted several methodologies to address data imbalance and overfitting.

1) Model Selection

Light GBM is a state of the art gradient boosting framework that uses a tree-based learning algorithm. It differs from other boosting algorithms in that it splits tree leaf-wise with the best fit rather than depth-wise or level-wise. The following are the advantages of light GBM:

- Faster training speed and higher efficiency

  Light GBM uses a histogram-based algorithm. It buckets continuous feature values into discrete bins which fasten the training procedure.

- Lower memory usage

  Light GBM replaces continuous values to discrete bins which results in lower memory usage.

- Better accuracy than any other boosting algorithm

  Light GBM produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy.

- Compatibility with Large Datasets

  Light GBM is capable of performing equally well with large datasets with a significant reduction in training time as compared to XGBoost.

However, there are also disadvantages of using Light GBM.

- Overfitting

  Light GBM split the tree leaf-wise, which can lead to overfitting as it produces much more complex trees.

- Compatibility with datasets

  Since Light GBM is sensitive to overfitting, it may not be suitable for small datasets. There is no threshold on the number of rows, but according to forums I read, it was suggested that it should be used only for data with more than 10,000 rows. Since the given dataset far exceeded 10,000 rows, I decided light GBM was suitable for our task.

I also tried implementing and experimenting with CatBoost, another state-of-the art gradient boosting algorithm. Worthy of its name, it is known to work especially well with categorical variables. Since our dataset include many categorical variables, I thought it would be a suitable model to test. There are several advantages in using CatBoost as it can handle categorical feature automatically, and is known to be quite robust. It reduces the need for extensive hyper-parameter tuning and lower the chances of overfitting, and is known to produce good generalized models. Before tuning hyperparameters, Light GBM and Catboost had similar performance, but after tuning hyperparameters, Light GBM seemed to have better performance, so I selected light GBM over Catboost as my main model in the end.

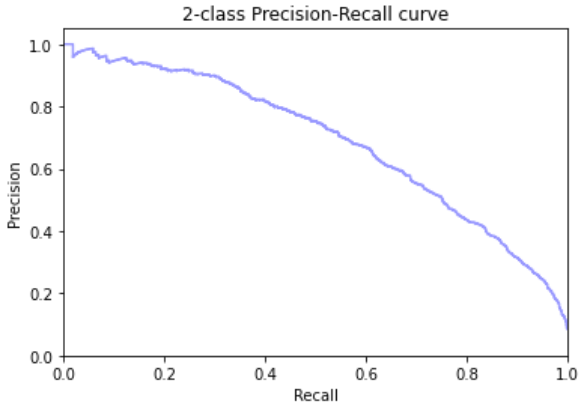

Fig 1-a) Catboost Model's Precision-Recall Curve

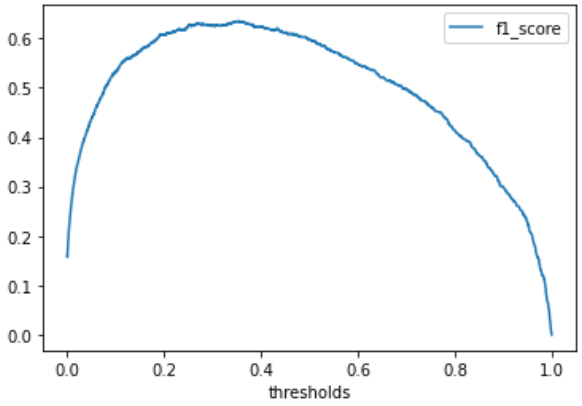

Fig 1-b) Catboost Model's Threshold Tuning Curve

Besides ensemble models such as Light GBM and Catboost, I also tried more traditional models such as logistic regression, support vector machines, naive bayes, and ridge classifier. However, I found that ensemble models outperformed compared to those models. Ensemble methods increase computational cost and complexity, but they can often yield better predictive performance and robustness.

	model	datatype	f1	precision	recall	accuracy
1	Logistic Regression	SMOTE	0.541805	0.605266	0.490389	0.948748
2	Logistic Regression	ADASYN	0.535018	0.597717	0.484224	0.947991
0	Logistic Regression	ORIGINAL	0.506376	0.701623	0.396139	0.952276
4	Ridge Classifier	SMOTE	0.492394	0.474266	0.511964	0.934774
5	Ridge Classifier	ADASYN	0.478075	0.454280	0.504502	0.931933

Table 1. Comparing the performance of different models with different over/under sampling methods

2) Parameter Tuning

Light GBM has more than 100 hyperparameters, and it is important to tune hyperparameters in order to achieve maximum performance. I will discuss few of the important hyperparameters.

- max_depth: describes the maximum depth of tree. It is used to handle model overfitting.
- num_leaves: a main parameter to control the complexity of the tree model. Theoretically, we can set num_leaves=2^(max_depth) to obtain the same number of leaves as depth-wise tree. However, this simple conversion is not good in practice. The reason is that a leaf-wise tree is typically much deeper than a depth-wise tree for a fixed number of leaves. Unconstrained depth can induce over-fitting. Thus, when trying to tune the num_leaves
, we should let it be smaller than 2^(max_depth).
- min_data_in leaf: minimum number of the records a leaf may have. Its optimal value depends on the number of training samples and num_leaves. Setting it to a large value can avoid growing too deep a tree, but may cause under-fitting. In practice, setting it to hundreds or thousands is enough for a large dataset.

- feature_fraction: used when boosting is random forest. 0.8 feature fraction means LightGBM will select 80% of parameters randomly in each iteration for building trees.
- bagging_fraction**:** specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
- bagging_frequency: specifies the frequency for bagging. 0 means disable bagging, where as k means perform bagging at every k iteration.
- early_stopping_round: help reduce excessive iterations and speed up analysis. Model will stop training if one metric of one validation data doesn't improve in last early_stopping_round rounds.
- lambda_l1, lambda_l2: specifies regularization. b
- min_gain_to_split**:** describes the minimum gain to make a split. It can used to control number of useful splits in trees

Light GBM also offers several types of boosting algorithms. The default is the traditional boosting decision tree (gbdt), but there are also random forest (rf), drop out meets multiple additive regression trees (dart), and goss (gradient-based one-side sampling). I tried every boosting algorithms and found that dart gives the best performance.

After spending many hours manually changing parameters, I found it quite tedious. So, I looked for automatic parameter tuners and came across Optuna. Optuna is an automatic hyperparameter optimization software framework, particularly designed for machine learning. It features state of the art algorithms to efficienctly search large spaces and prune uncompromising trials for faster results. Using Optuna saved up a lot of time and helped me achieve better performance.

3) Class Imbalance

One noticeable characteristic of our data set was that it is imbalanced. There were only 12329 people who had income over 50k as compared to 187194 people who had income lower than 50k. Working with imbalanced datasets is difficult because most machine learning techniques will ignore and have poor performance on the minority class. There are several approaches to addressing imbalanced datasets.

The first method I considered was over and undersampling methods. I tried using Adaptive Synthetic (ADASYN), Random Undersampling, and Synthetic Minority

Oversampling Technique (SMOTE). SMOTE was the best functioning method for the given dataset. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically *k=5*). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space. SMOTE helped increase f1 score of models such as logistic regression, but it actually worsened the performance of my model using Light GBM, so I did not use it in the end.

The method that worked best for addressing class imbalance was threshold tuning. Many machine learning algorithms are capable of predicting a probability or scoring of class membership, and this must be interpreted before it can be mapped to a crisp class label. This is achieved by using a threshold, such as 0.5, where all values equal or greater than the threshold are mapped to one class and all other values are mapped to another class. For those classification problems that have a severe class imbalance, the default threshold can result in poor performance. As such, a simple and straightforward approach to improving the performance of a classifier that predicts probabilities on an imbalanced classification problem is to tune the threshold used to map probabilities to class labels. I used precision-recall curve of the validation set in order to find the optimal threshold. Using this method significantly increased my model's performance in terms of F1 score.

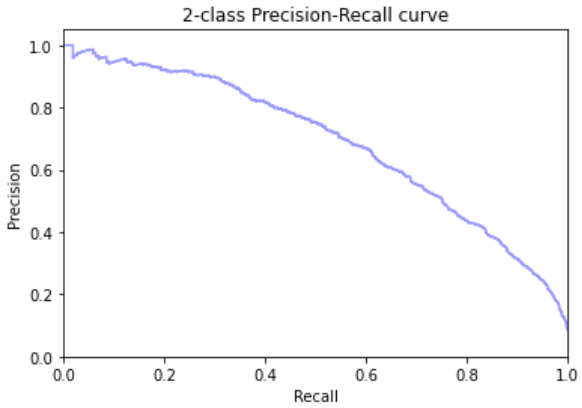

Fig 2-a) Final Model (Light GBM)'s Precision-Recall Curve

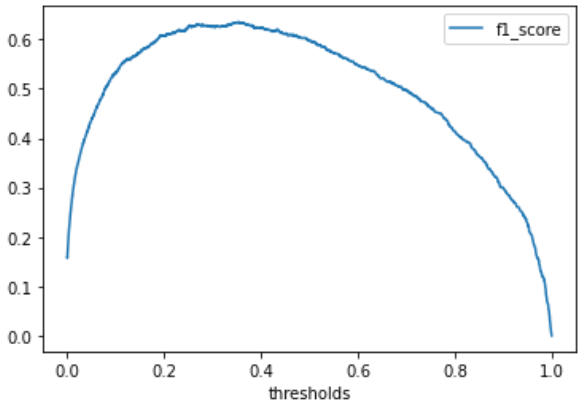

Fig 2-b) Final Model (Light GBM)'s Threshold Tuning Curve

4) Regularization

As with most machine learning problems, finding the optimal model complexity was a significant challenge. In the initial stages of building the model, I focused my efforts on improving the model performance and mainly relied on training error rate to check if my model was learning the patterns inherent in the data correctly. I increased number of iterations and changed various hyperparameters to build a more complex model. However, after a certain point, I realized that test error rate started to increase even though training error rate was decreasing. Overfitting was occurring, and I used regularization methods to make the model more general. The following is the list of methods I tried.

- Lowering the number of iterations
- Adding L1/ L2 regularization
- Using smaller number of trees
- Using bagging by set
- Setting max depth limit to avoid growing deep trees

After many trials and errors, I found a set of hyperparameters where testing error was at its lowest, and the gap between training and testing error rate was reduced compared to before implementing regularization.

5) Miscellaneous

Besides the four main areas of improvement I focused on, there were several other ways I tried to improve my model. First was feature selection. Initially, the model had 40 different features. Few columns had large number of missing values and had low predictive power. I first identified these less useful features by plotting a feature importance plot. Then, I tried training the model after deleting those features and I found that training speed increased and the model performance also increased. I also plotted a correlation heatmap of numerical variables, and checked that most features were not correlated with each other, with the exception of number of persons worked for an employer and weeks worked in a year. However, I decided to keep both features because they both ranked high on the feature importance plot.

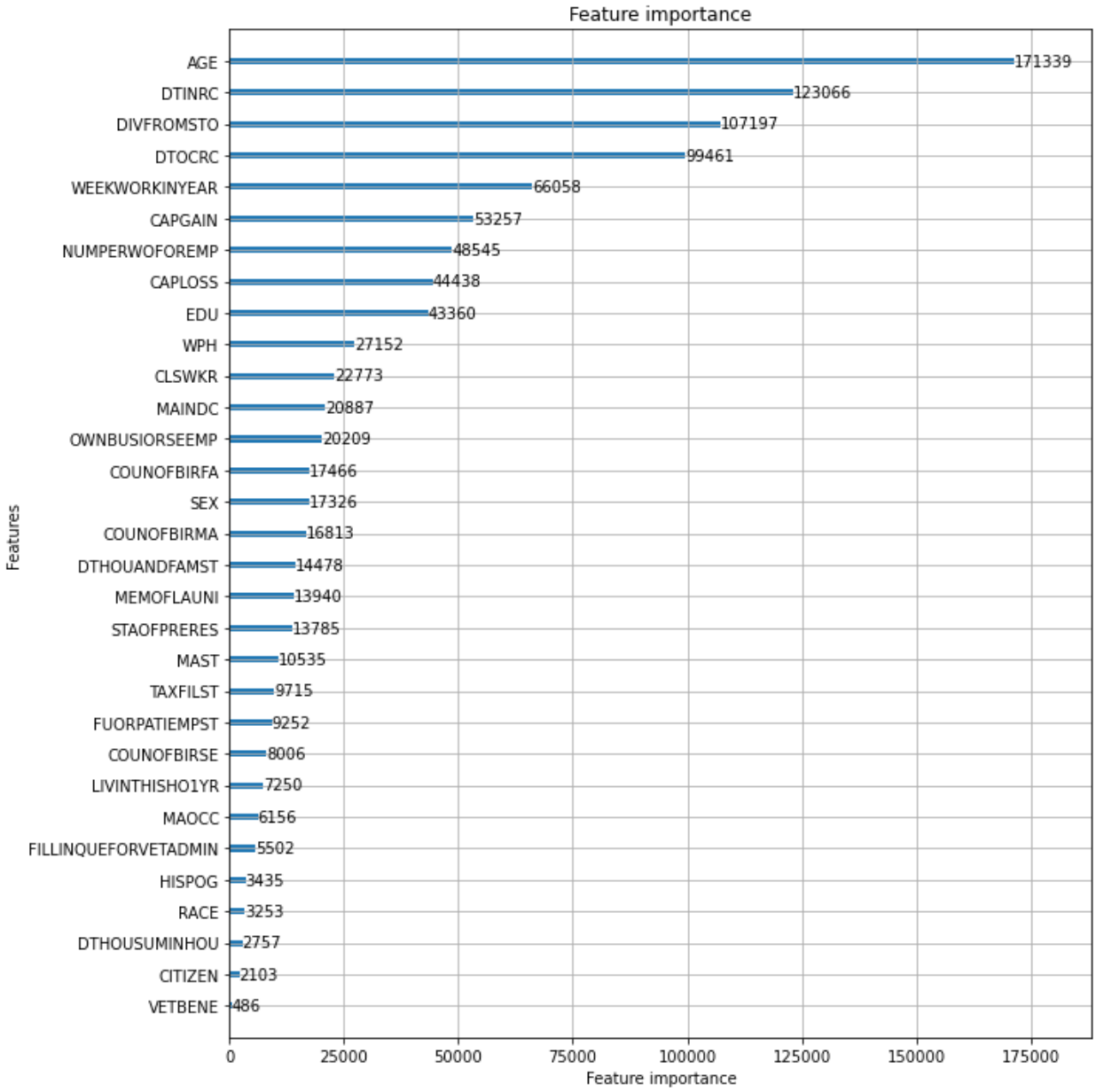

Figure 4. Feature Importance Plot

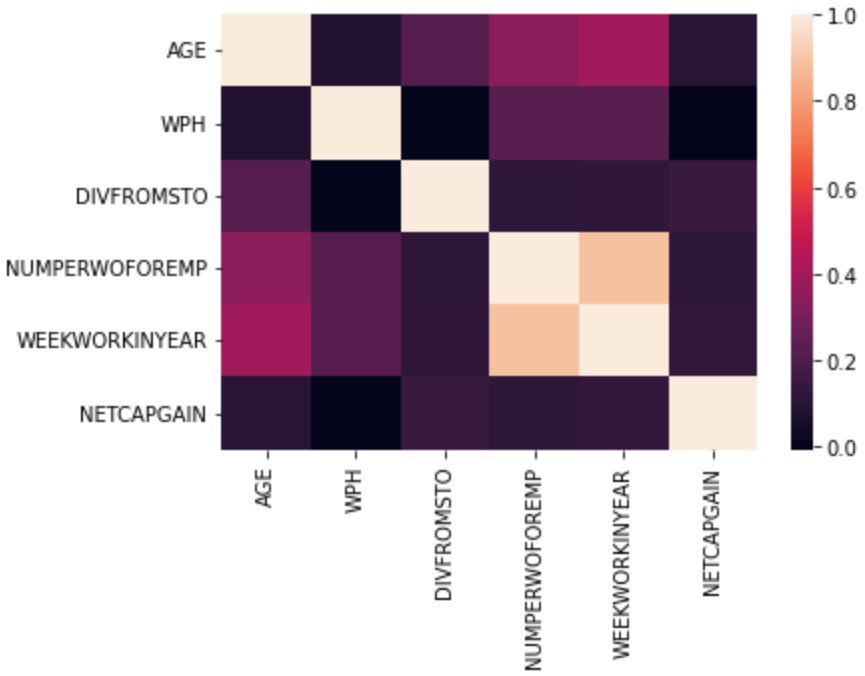

Figure 5. Correlation Heatmap

Another way I tried to improve my model was by trying different scalers. If the data in any conditions has data points far from each other, scaling is a technique to make them closer to each other. In simpler words, scaling is used for making data points generalized so that the distance between them will be lower Decision trees and ensembles of trees are known not be affected by scale of numerical input variables, and I did find that the performance of my model did not change depending on the method of scaling. Still, I tried experimenting with several scalers-Robust Scaler, MinMax Scaler, Standard Scaler, Log transform etc. I used log tansform in the end because it minimizes the skewness of the data. When I plotted the distribution of features, I found that many features had heavily skewed distribution, so I found it appropriate to used log transform using numpy's log1p() function.

Lastly, I tried data binning. Data binning is a pre-processing technique used to reduce the effects of minor observation errors. It groups numbers of more-or-less continuous values into a smaller number of bins. As I mentioned previously, many continuous had skewed distribution, and categorical data was also heavily concentrated for one value and had small number of observations for other categories. Therefore, I thought my model could benefit from binning. Light GBM is a histogram based algorithm, which buckets continuous features into discrete bins to construct feature histograms during training. It has max_bin parameter, which controls how data into bins before going into

learning. Binning can greatly reduce number of candidate splits and thus increase training speed.

Future Works

My model had a relatively good performance, but there are still many ways it could be further improved. Within the LightGBM framework, I could try tuning a wider range of hyperparameters. Light GBM has over 100 hyperparameters, and due to time constraint and limited computing power, I was not able to experiment with all of them. One of the main problem of my model was overfitting, and although I used various methods of regularization, there still quite a significant gap between training and testing error. I think my model's performance could improve if I implemented more methods of regularization. Furthermore, there were quite a large variation in terms of performance when I ran the model with tweaked hyperparameters, so I think it is important to improve the robustness of my model as well.

Going beyond my current model, I think another direction for future work would be to use deep learning. Deep learning refers to a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data. Past research shows that through the effective use of deep learning models, binary classification problems can be solved to a fairly high degree. Neural Networks are remarkably good at figuring out functions from X to Y. Of course, choice of the algorithm to choose needs to be driven by the problem at hand and factors like, how much data size is available, computation power, etc. Neural networks are generally recommended if the available data size is large. I did not implement deep learning model for our problem context, because the data size was not very large. However, if we had more data, or if we could augment the current data by creating synthetic samples, deep learning could be used potentially.

Conclusion

Besides improving the model's F1 score, I was also interested in exploring the data and deriving meaningful relations. It is widely known that gaps between gender, race, family background and other social factors influencing a person's income, but the exact relationship is not well understood. Exploring this income dataset can help policymakers

identify social factors that influence income gap. I was interested in the following three research questions.

1. Which feature is the most important in determining income classification?
2. Ignoring income, what factors appear to be highly correlated with each other?
3. Given a set of social factors, can we predict whether the income of a person, given a set of social factors?

Through exploratory data analysis and development of binary classification model, I was able to address all three of the research questions. By determining the feature importance, I was able to check which variables had greatest impact on income. By checking for correlation among features, I could see that some social factors were correlated each other. Lastly, by developing a model that achieved around F1 score of 65.36 for the test set, I was able to conclude that given a set of social factors, it is possible to predict whether the income of a person is below or higher than 50k.

I think the there are significant implications for studying this data. Researching the income gap and its root causes can allow us to determine what measures are needed to remedy those gaps. It would also be interesting to analyze the census data from different countries to gain insight on how different social factors matter more or less in different contexts. It is quite clear from the data that not every have equal opportunity. Identifying that there is a problem is the first step towards correcting it.