Project 3 - Adult Census Income

YANG Xuan
12112729

I. Introduction

The content of this project is to learn a model on an Adult Census Income dataset and to predict whether income exceeds $50K/yr based on census data. This dataset stores basic information about multiple adults, including age, educational program, and literacy level, and finally records whether the individual's annual income exceeds 50K. We are required to use the model we trained from traindata to predict whether these individuals' income exceeds $50K/yr in testdata. It is a classification problem.

II. Data Processing

A. Read in Data

The training dataset and testing dataset are stored in traindata.csv and testdata.csv, with file format to be .csv type. I use function `read_csv()` from a Python data analysis library called pandas to read the two .csv file first, where the data will be loaded into a dataframe. And for the label of traindata stored in trainlabel.txt, I use another library numpy's function `genfromtxt()` to read this .txt file and store the content in a nparray. The label distribution of the training data is as follows:

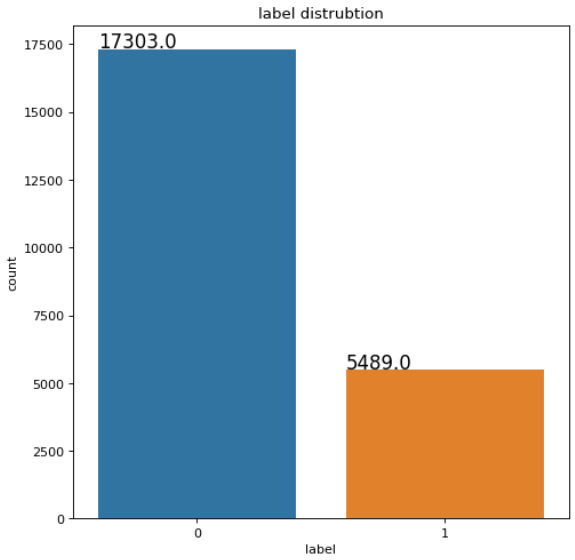

Fig. 1. label distribution

Identify applicable funding agency here. If none, delete this.

B. Handling outliers

After reading in data, next step is analysis and process data. I first analysis the type of each features using `dataframe.dtypes` and find there are two types: int64 and object. To figure out whether there are null-type in dataframe, just calling `.isnull().sum()`, and find there are no null-type expect three features: workclass, occupation, native.country, which have '?' type in the dataset, and '?' counts for 5.607231%, 5.629168%, 1.855914% respectively. Due to the high percentage of '?', I just treat it as a class of each features. To distinguish them, I changed their respective names to: 'unknown_workclass', 'unknown_occupation', and 'unknown_country'.

TABLE I
Feature name and types

Feature name	type
age	int64
workclass	object
fnlwgt	int64
education	object
education.num	int64
marital.status	object
occupation	object
relationship	object
race	object
sex	object
capital.gain	int64
capital.loss	int64
hours.per.week	int64
native.country	object

C. Encoding

As there are many features' type are object, which sklearn models cannot directly read and process, so they must be converted to data type using a method called one-hot encoding.

The one-hot coding can transform a categorical variable containing m categories into a binary matrix of n*m, where n is the number of observations and m is the number of categories. sklearn provides OneHotEncoder class to implement unique hot coding.

D. Reduce dimension

For a feature with m categories, the one-hot encoding will introduce m new features, representing each of the m categories of the categorical variables, which will Significantly increasing dimension and causing dimensional disasters. So what I should do next is to reduce the dimension.

At first I used variance to reduce the dimension and use correlation coefficient to select features, unluckily, this method

failed. If some features variance is to small, which means that most data are consistent, so the validation of this data is not high and can be dropped. I set the threshold value to 0.1 and only 23 of the 107 features were retained. To analyze the correlation coefficient, I draw a heatmap to visually analyze. However, most correlation coefficients are lower than 0.25, so I can't replace one feature by another.

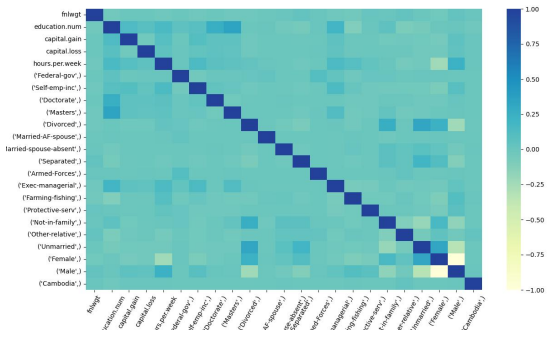

Fig. 2. heatmap of these 23 features

I try another method to reduce the dimension. Random forest is a widely used feature selection algorithm, which automatically calculates the importance of each feature. So I build a random forest model to calculate the importance of each feature and select the most important 15 features. Also, the random forest model is imported from sklearn library.

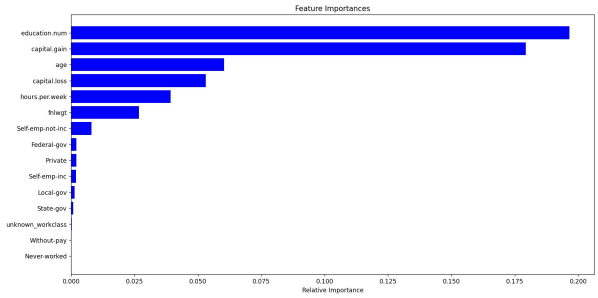

Fig. 3. Importance of features

From Fig.3, the most important features are: 'education.num', 'capital.gain', 'age', 'capital.loss', 'hours.per.week', 'fnlwgt', 'Self-emp-not-inc','Federal-gov','Private', 'Self-emp-inc','Local-gov','State-gov'.

III. Building the Model

A. Model selection

I mainly used the decision tree model and compared the differences between decision tree models with different libraries and different parameters. The main two libraries that I use are sklearn and lightgbm. First, sklearn library provide a decision tree model whose criterion can be either entropy or gini, and there are totally 12 parameters provided by the interface that can be adjusted to get a better model. For lightgbm, it is a framework for implementing the GBDT algorithm, which supports efficient parallel training and has the advantages of faster training speed, lower memory consumption, better accuracy, and distributed support for fast processing of large amounts of data. The main advantages of lightgbm are:

1) Histogram-based decision tree algorithm
2) Leaf-wise leaf growth strategy with depth limitation
3) Direct support for Categorical Feature

The best thing of lightgbm is that it can directly support categorical feature, which means that one-hot encoding is not necessary so the disadvantages brought by one-hot encoding such as dimension disaster won't happen.

B. Model evaluation

To evaluate the model, I split the traindata to two parts: train part and test part, each with a share of 0.9 and 0.1. The train part is used to train a model and the test part is for predict and evaluate. The model trained by the train part will be used to predict the labels of test part, and I will compare the predict labels with the true labels to evaluate this model's accuracy. Sklearn library has functions called `train_test_split` that can split dataset and `accuracy_score` to evalate the model's accuracy. The accuracy can also be calculated as follow:

$$accuracy = \frac{diff(Lable_{true}, Lable_{predict})}{|testdata|} * 100\%$$

IV. C4.5 Decision Tree

To build a simple decision tree whose criterion is entropy, just call the following functions:`DecisionTreeClassifier`, and set the criterion equals to entropy, then a entropy decision tree is built and the other 11 parameters are set to default. To train the model using data, just call `DT.fit` and to predict label, call `DT.predict`.

A. Adjustment of parameters

The parameters of sklearn.DecisionTreeClassifier are listed below and can be divided into three categories.

TABLE II
Parameters of DecisionTreeClassifier

Parameter name	Effects
criterion	Classification criteria
max_depth	Maximum depth of the tree
min_samples_split	minimum number to split the internal nodes
min_samples_leaf	Minimum number of samples on leaf nodes
class_weight	Sample weights
max_features	Number of features
min_impurity_decrease	Minimum impurity of node division
max_leaf_nodes	Maximum number of leaf nodes
random_state	Random seed
splitter	Controlling the randomness
min_weight_fraction_leaf	Sum of the minimum sample weights of the leaf nodes
ccp_alpha	Complexity Parameters

The most important paramters that we should mainly adjust are: criterion, max_depth, min_samples_split, min_samples_leaf. However, we are building an entropy decision tree, the criterion is entropy and not need to adjust, so there are only three parameters to be adjusted.

1) max_depth: First I search the parameter range of max_depth in a wide range, a.k.a from 1 to 100, and the cross_val_score function is used for cross-validation to evaluate the effect of the parameters on the model. The variations of score according to different max_depth is shown in Fig.4.

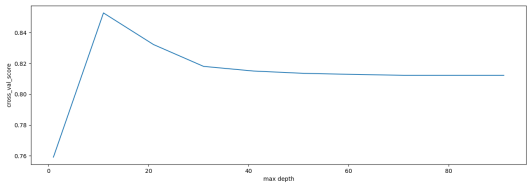

Fig. 4. Adjustment of max_depth

When the depth of the tree reaches 10, increasing the depth of the tree again, the model correct rate no longer rises. Then we search further around 10 using the same method. Finally, when the max_depth is 11, the cross_val score reach maximum, so I temporarily set the max_depth to be 11.

B. min_samples_split and min_samples_leaf

The adjustment process of min_samples_split and min_samples_leaf are similar to max_depth: Rough search in a large area firstly, determine a small area and then search precisely. Also cross_val score is used to judge the effect. The variation of these two parameters are shown in Fig.5 and Fig.6. The optimal min_samples_split is 32 and optimal min_samples_leaf is 25.

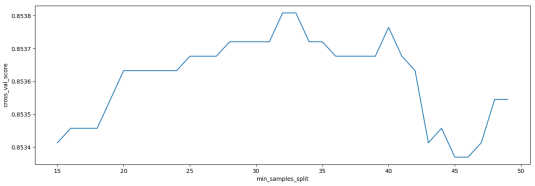

Fig. 5. Adjustment of min_samples_split

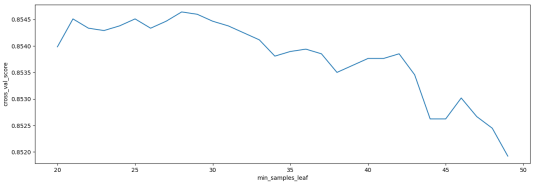

Fig. 6. Adjustment of min_samples_leaf

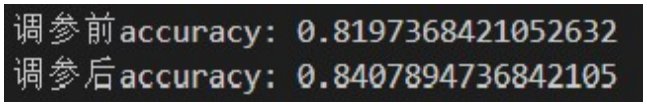

Fig. 7. Accuracy change in C4.5 decision tree

C. Grid Search

The optimal parameters that are adjusted individually are not necessarily optimal when put together, so it is also necessary to adjust the three parameters together in a small range. Sklearn library provides a function `GridSearchCV` which can adjust parameters together. After grid search, the best value of max_depth, min_samples_split and min_samples_leaf are:

- max_depth = 8
- min_samples_split = 32
- min_samples_leaf = 28

which are a little bit different from adjusting separately.

D. Model accuracy

To evaluate the accuracy of the model, just use `accuracy_score` mentioned in Model evaluation on test part. The change in accuracy of the model before and after the tuning is as follows:

Accuracy improved by around 2% through adjustment.

V. CART Decision Tree

The decision tree model provided by the sklearn library is able to use the gini coefficients as a criterion for classification. When dealing with high-dimensional data, CART decision trees generally outperform C4.5 decision trees. To build a CART decision tree, just set the criterion equals to gini and the rest is the same as the C4.5 decision tree. And there are also 12 parameters that can be adjusted manually. Due to space limitations, the detailed tuning process is not shown here, which is almost the same as C4.5 decision tree.

A. Model accuracy

Fig.8 show the accuracy improving after tuning the parameters.

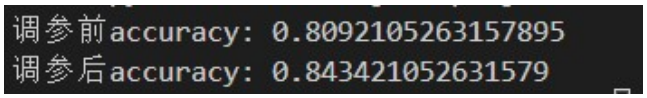

Fig. 8. Accuracy change in CART decision tree

B. Visualization

Matplotlib library and sklearn library provide tools to visualize decision trees. to visualize a tree, first call `plot_tree()` from skearn, an then call `Matplotlib.show()` to plot the decision tree on the canvas. Fig.9 is the visualization of CART decision tree

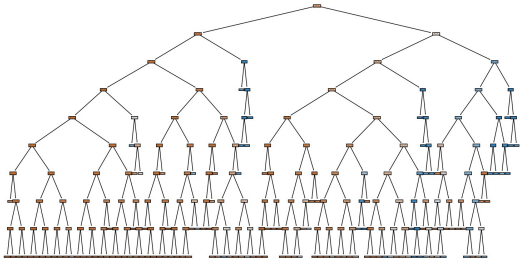

Fig. 9. Visualization of CART decision tree

VI. LightGBM

GBDT (Gradient Boosting Decision Tree) is a long-lasting model in machine learning, whose main idea is to use weak classifiers (decision trees) to iteratively train to get the optimal model, which has the advantages of good training effect and less overfitting. LightGBM (Light Gradient Boosting Machine) is a framework that implements the GBDT algorithm, using Leaf-wise instead of Level-wise of XGBoost which is another decision tree algorithm to get better classification results. The most wonderful thing that lightgbm support is that, it can directly support categorical feature so the category features do not need to be processed by the one-hot encoding.

A. Building a model

The steps to build a decision tree model based on the lightgbm library are simple:

1) Importing lightgbm
2) Calling `LGBMClassifier()` from lightgbm to build a tree
3) Calling `fit()` to train the tree using train part data

B. Adjustment of parameters

A total of 11 parameters can be adjusted, but the actual need to adjust only 6, they are:

TABLE III
PARAMETERS OF LGBMCLASSIFIER

Parameter name	Effects
learning_rate	learning rate
min_child_sample	Number of iterations of boosting
num_leaves	Number of leaf nodes on a tree
max_depth	Maximum depth of the tree model
subsample	The ratio of selecting data
colsample_bytree	The ratio of selecting features

1) learning_rate: Generally, the smaller the learning rate, the better the final performance of the model, but too small a learning rate often leads to overfitting of the model and affects the training time of the model. Generally, a fixed value such as 0.1 or 0.05 is preset during the tuning process, and then a good value between 0.05 and 0.2 is searched for as the final model parameters after the other parameters are determined.

The variation of score with learn_rate is shown in Fig 10. The temporarily optimal learn_rate is 0.09.

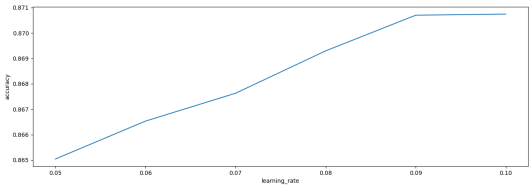

Fig. 10. Learning_rate

2) min_child_sample: This parameter needs to be determined according to the data set, generally small data sets with the default 20 is enough, but large data sets still use the 20 will make the amount of data generated on the leaf nodes is too small, which will lead to the problem of unrepresentative data sets. After searching, the temporarily optimal min_child_sample is 60.

3) max_depth: max_depth is an important parameter, too low will lead to poor classification and too high will lead to overfitting, so it is critical to determine an appropriate max_depth. By local search, when max_depth is equal to 5, the classification effect is better and not overfitting.

4) num_leaves: In LightGBM, the number of leaf nodes is set to match with max_depth to be less than $2^{max\_depth} - 1$. In parameter search, max_depth is needed to limit the range of values of num_leaves. Consider the value of the max_depth, I set the value of num_leaves to be 90.

5) subsample and colsample_bytree : These two parameters do not need too fine tuning, just take a better value in a wide range. In general, subsample and colsample_bytree are taken between [0.8, 0.9, 1.0]. After further comparing the classification effects of different values of subsample and colsample_bytree, I set subsample to be 0.8 and colsample_bytree to be 0.9.

6) Grid Search: To find the best combination of parameters, I used a grid search on a small scale to determine the effect of combining all parameters. The optimal values are:

- learning_rate = 0.1
- n_estimators = 300
- min_child_samples = 60
- max_depth = 5
- num_leaves = 90
- subsample = 0.8
- colsample_bytree = 0.9

C. Model accuracy

The classification result of the decision tree built with lightgbm is significantly better than that of the decision tree built with sklearn, with an accuracy of nearly 90% after adjusting all parameters, with a 5% improvement

Fig. 11. Accuracy of Decision tree from lightgbm

D. Visualization

To visualize the decision tree from lightgbm, another tool is also needed: Graphviz. Graphviz is an open source graph visualization tool, ideal for drawing structured icons and networks. To use Graphviz, The path to the Graphviz bin file needs to be loaded into the system path. After loading, `plot_tree` can be use to draw the decision tree. The decision tree of my model is shown in Fig.12.

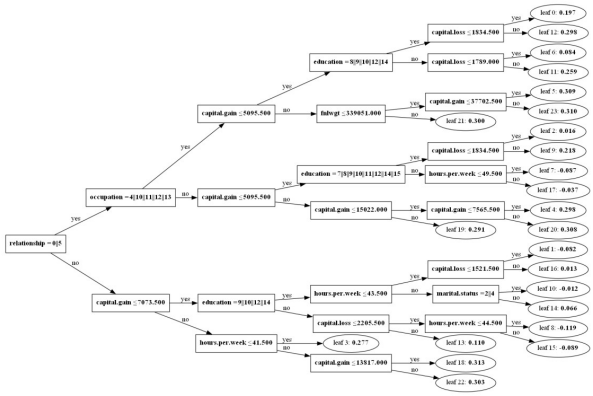

Fig. 12. Decision tree from lightgbm

VII. Results comparison and analysis

The changes in accuracy of these three models before and after tuning are shown in Fig.12

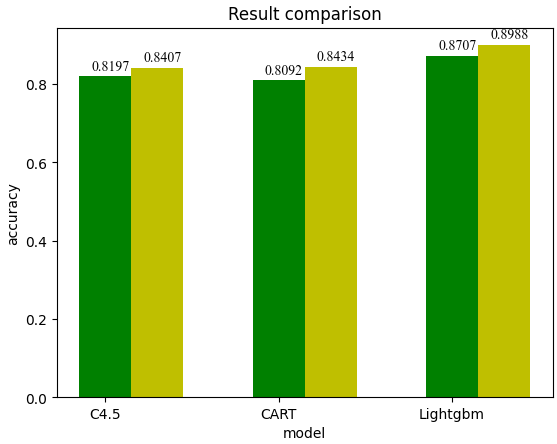

Fig. 13. Result comparison

The C4.5 decision tree and the CART decision tree from sklearn are better in C4.5 before tuning and CART after tuning, but both are not as good as the decision tree from lightgbm, with an accuracy of about 84%. The accuracy of the lightgbm decision tree reaches 87% before and 89.88% after the conditioning, so it is obvious that the decision tree model trained with lightgbm is much better than the model trained with sklearn. I think there are three possible reasons for this result:

1) one-hot encoding causes dimension disaster.
2) one-hot encoding and random forest selection leads to important features being discarded
3) The GBDT algorithm underlying lightgbm outperforms the entropy and gini coefficient algorithms

VIII. Limitations and Reconsideration

I think this project has the following difficulties:

- Handling of missing values
- Processing of category features
- Adjustment of parameters

For the first one, by analyzing the data, it can be seen that the missing value '?' only exists in the category feature. And in 'workclass' column, '?' has a high percentage of 5.6%, so delete missing value directly is not a wise choice. So I just treat '?' as one of the classes of the feature, maybe there are better solutions such as replacing the missing values with the plural class.

For the second one, I think it is the main reason why sklearn's decision tree model becomes less effective. As sklearn can only process data features, these category features must be converted into data. The only method I know is encoding, including one-hot encoding and label encoding. However, label encoding is not suitable because it will assign arithmetic meaning to category features. And for one-hot encoding, the problem is dimension disaster. So maybe there are some better methods to treat category features and that's why lightgbm's performance is better than sklearn.

For the last one, there are many uncertainties in the adjustment process. Not only performance should be taken into account, but also efficiency as well. So search in a wide range first and then do a small precise search, finally conduct a comprehensive search using a grid should be a good way to adjust the parameters.