Comparing Boosting Models

2018015723 조영유
2019048704 김정현
2020067856 최천권
2020055950 리쯔신
9059620233 Dai Remi

INDEX

01 What is Gradient Boosting?

Gradient Boosting

- An ensemble learning method that combines several weak models, typically decision trees, in an iterative manner to create a strong predictive model
- Effective predictive performance across various problems, especially in classification and regression tasks
- Less sensitive to feature scaling or outlier handling and evaluate the importance of variables

Gradient Boosting Decision Tree

- An ensemble learning technique where trees are sequentially trained to minimize the residuals of the previous trees
- Each tree is constructed in a way that focuses on improving predictions based on the residuals of the previous tree
- Gradient Decent is employed to reduce prediction errors, adjusting the splitting points and predicted values of the trees using gradients

02

Three Models based on Gradient Boosting

XGBoost

dmlc XGBoost

- Developer: DMLC, Tianqi Chen
- Release Year : 2014

01

Level - Wise tree Growth

- Expand Tree level by level

02

Pre-sorted & Histogram-based algorithm

- Splitting Methods using in XGBoost

03

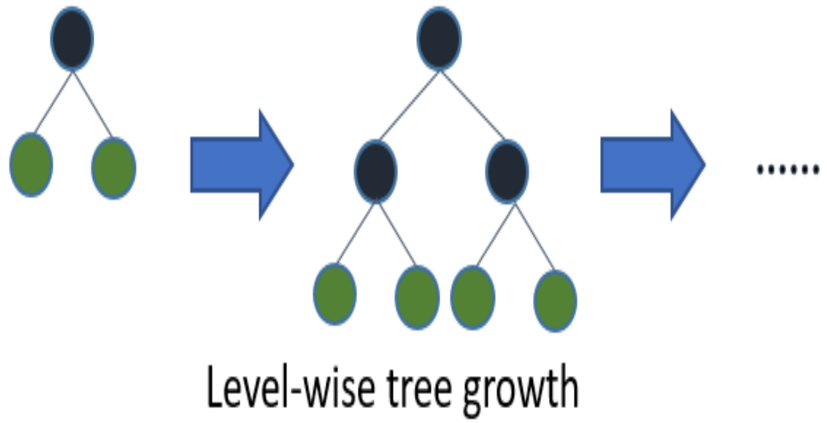
Parallel Computation & Distributed Training

- Help to faster training, handling big data, low memory usage

XGBoost

Tree Growth

Level - Wise Tree Growth



- The tree is expanded level by level, with all the nodes at a particular level being processed together before next level
- Assign current information gain to every nodes in the same depths, then select the split with the largest gain and develop the tree
- Well-performed in small dataset

XGBoost

Spliting Method

Pre-sorted algorithm & Histogram-based algorithm

- Pre-sorted Algorithm considers all feature and sorts them by feature value
- Histogram-based algorithm splits all the data points for a feature into discrete bins and use these bins to find the split value of histogram
- Histogram-based algorithm is more efficient than Pre-sorted algorithm

LightGBM

- Developer: Microsoft
- Release Year : 2016

01 Leaf - Wise tree Growth

- Expand the leaf nodes in a depth-first manner

02 Gradient-based One-Side Sampling

- More improved splitting method than XGBoots's splitting methods

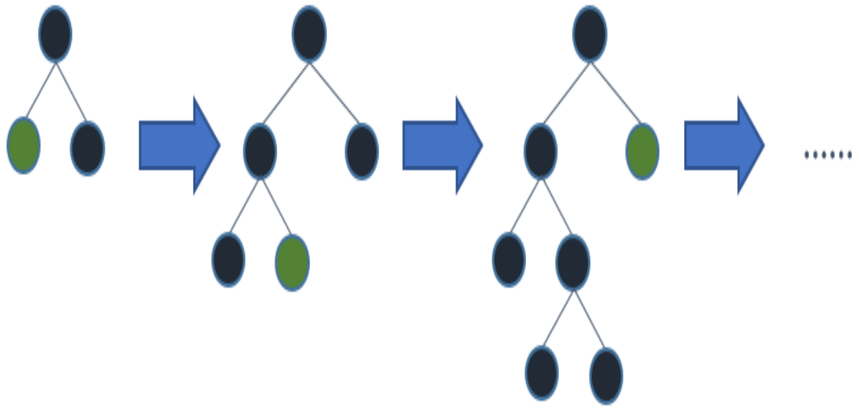
03 Automatic handling of Categorical features

- Handling categorical variables without preprocessing steps

LightGBM

Tree Growth

Leaf - Wise Tree Growth (Best - First Tree Growth)



Leaf-wise tree growth

- Expand the leaf nodes in a depth-first manner
- Select the split with the largest information gain first, then develop the tree by spliting it vertically
- Construct more deeper trees with fewer splits
- Provide faster learning rate and lower memory usage compared to the conventional level-wise tree growth

LightGBM

Spliting Method

Gradient - based One - Side Sampling (GOSS)

- Keep all data instances with large gradients and performs random sampling for data instances with small gradients

- Data points with larger gradients would be important for finding the optimal split point, while data points with smaller gradients would be important for keeping accuracy for learned decision trees

- More efficent and improved than pre-sorted and histogram-based algorithms

CatBoost

- Developer: Yandex
- Release Year : 2017

01 Symmetric trees (or Balanced trees)

- Spliting condition being consistent across all nodes of the same depth
- Faster computation and evaluation and control overfitting

02 Ordered Boosting

- Each model trains on a subset of data and evaluates another subset of data
- Increasing robustness to unseen data

03 Ordered Target Statistic

- Regularization technique in Catboost that utilizes target variable to encode categorical features and improve the model's predictive performance

CatBoost

Ordered Boosting

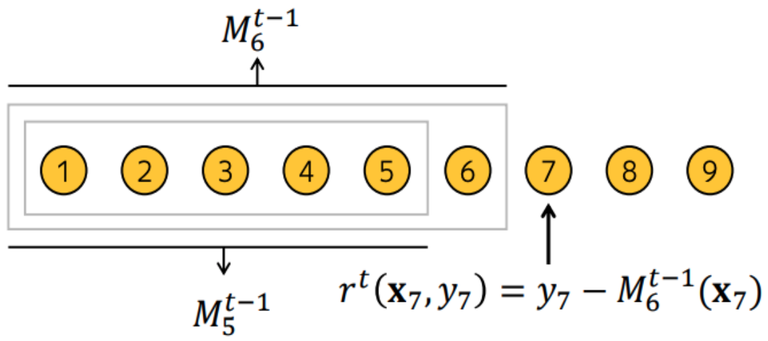


Figure 1: Ordered boosting principle, examples are ordered according to σ.

- When calculating residuals, it can be observed that the previous trained model is used
- Create a random permutation and sequentially calculating residuals to train the trees, which helps prevent target leakage
- Require a separate training model for each data

03 Comparing Boosting models

XGBoost

- Performance
  - Excellent performance on small datasets
- **Speed**
  - Level-wise tree growth
  - Rapid training speed on small datasets
- **Memory Usage**
  - Tendency to use lots of memory on the large datasets

LightGBM

- **Performance**
  - Excellent performance even on large datasets
- **Speed**
  - Leaf-wise tree growth
  - Rapid training speed even on large datasets
- **Memory Usage**
  - Efficient memory usage make it advantageous for processing large datasets

LightGBM

- **Performance**
  - Excellent performances on datasets with diverse features
- **Handling Categorical Variables**
  - Automatic handling and data discretization method
  - Slight decrease in performance compared to CatBoost

CatBoost

- **Performance**
  - Excellent performances on datasets with a large number of categorical variables
- **Handling Categorical Variables**
  - Ordered Boosting and Ordered Target Statistics
  - Built-in support for handling categorical variables

CatBoost

- **Performance**
  - Specialize in handling categorical variables
  - Excellent performances on datasets with a large number of categorical variables
- **Speed**
  - Generally, training speed is longer than XGBoost
- **Memory Usage**
  - Ensure efficient memory usage in large datasets

XGBoost

- **Performance**
  - Excellent performance on small datasets
  - Strengths in continuously reducing training error
- **Speed**
  - Rapid training speed on small datasets
- **Memory Usage**
  - Tendency to use lots of memory on the large datasets

04 Comparing Boosting models in Python

Experiment Model Implementation

- Prepared datasets are 'credit-g', 'adult', 'higgs', 'covertype' from OpenML
- 'credit-g', 'adult', 'higgs' are all binary classification problem
- 'covertype' has 7 categories
- 'adult' has categorical values more than others, and 'higgs' consists entirely of numeric values.
- 'adult' and 'higgs' were chosen to prove that the more a dataset has cateforical variables, the better CatBoost's performance is than LightGBM's performance

Experiment Model Implementation

- The datasets were preprocessed uniformly because all three models are used in one code
- In this manner, it was implemented by applying all models, which are XGBoost, LightGBM, CatBoost, to the four datasets
- To ensure an accurate performance comparison, number of trees, learning rate and maximum tree depth were unified

Experiment Model Implementation

[2]:
```
dataset = openml.datasets.get_dataset(31)

X, y, categorical_indicator, attribute_names = dataset.get_data(target=dataset.default_target_attribute, dataset_format='dataframe')


categorical_columns = [i for i, indicator in enumerate(categorical_indicator) if indicator]
```

[3]:
```
le = LabelEncoder()
y = le.fit_transform(y)
for col in categorical_columns:
    X.iloc[:, col] = le.fit_transform(X.iloc[:, col])
```

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[32]:
```
lgbm_model = lgb.LGBMClassifier(n_estimators=100,learning_rate=0.1,max_depth=6,random_state=42)

start_time = time.time()

lgbm_model.fit(X_train, y_train)

elapsed_time = time.time() - start_time
print("Elapsed Time:", elapsed_time)

memory_usage = psutil.Process().memory_info().rss / 1024**2  # in megabytes
print("Memory Usage:", memory_usage, "MB")

time_cost.append(elapsed_time)
memory_cost.append(memory_usage)

lgbm_accuracy = lgbm_model.score(X_test, y_test)
print("Accuracy:", lgbm_accuracy)
```

```
Elapsed Time: 0.040111541748046875
Memory Usage: 241.5859375 MB
Accuracy: 0.785
```

[33]:
```
catboost_model = cb.CatBoostClassifier(n_estimators=100,learning_rate=0.1,max_depth=6,random_state=42,verbose=False)

start_time = time.time()

catboost_model.fit(X_train, y_train)

elapsed_time = time.time() - start_time
print("Elapsed Time:", elapsed_time)

memory_usage = psutil.Process().memory_info().rss / 1024**2  # in megabytes
print("Memory Usage:", memory_usage, "MB")

time_cost.append(elapsed_time)
memory_cost.append(memory_usage)

catboost_accuracy = catboost_model.score(X_test, y_test)
print("CatBoost Accuracy:", catboost_accuracy)
```

```
Elapsed Time: 0.19858241081237793
Memory Usage: 243.24609375 MB
CatBoost Accuracy: 0.805
```

[34]:
```
xgboost_model = xgb.XGBClassifier(n_estimators=100,learning_rate=0.1,max_depth=6,random_state=42)

start_time = time.time()

xgboost_model.fit(X_train, y_train)

elapsed_time = time.time() - start_time
print("Elapsed Time:", elapsed_time)

memory_usage = psutil.Process().memory_info().rss / 1024**2  # in megabytes
print("Memory Usage:", memory_usage, "MB")

time_cost.append(elapsed_time)
memory_cost.append(memory_usage)

xgboost_accuracy = xgboost_model.score(X_test, y_test)
print("XGBoost Accuracy:", xgboost_accuracy)
```

```
Elapsed Time: 0.0861501693725586
Memory Usage: 244.625 MB
XGBoost Accuracy: 0.775
```

Compare time and memory

- As the previous slides, we recorded elapsed time and memory usage and saved them in an array, then we compared the time and memory to the last time

- The results appeared in this order : LightGBM, CatBoost and XGBoost

Compare time and memory

```
print(time_cost)

print(memory_cost)
```

```
[0.040111541748046875, 0.19858241081237793, 0.0861501693725586]
[241.5859375, 243.24609375, 244.625]
```

credit-g

```
print(time_cost)

print(memory_cost)
```

```
[0.29528331756591797, 2.373206853866577, 3.5202035903930664]
[151.6171875, 151.8984375, 163.20703125]
```

adult

```
print(time_cost)

print(memory_cost)
```

```
[0.46872830390930176, 2.0272743701934814, 9.841265439987183]
[371.29296875, 369.87890625, 374.6015625]
```

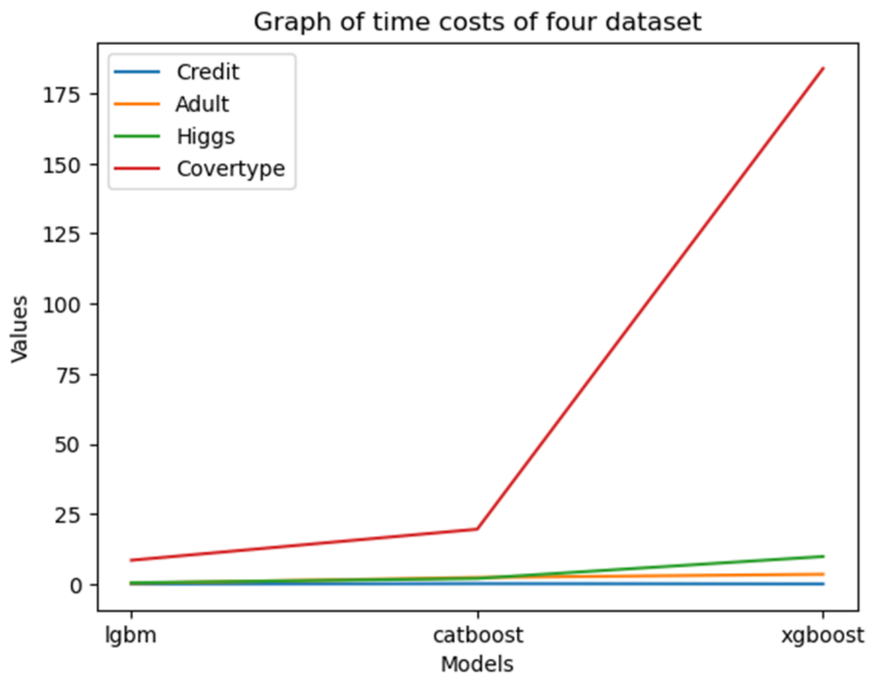
higgs

```
[36]: print(time_cost)

      print(memory_cost)
```

```
[8.525601148605347, 19.5908842086792, 183.84770441055298]
[530.36328125, 463.1953125, 449.21875]
```

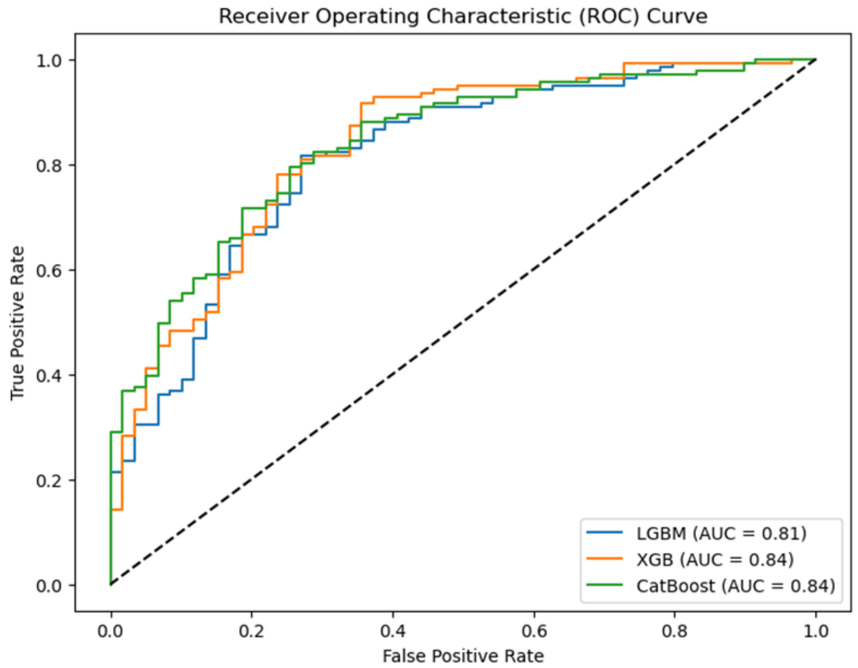
covertype

Compare time and memory

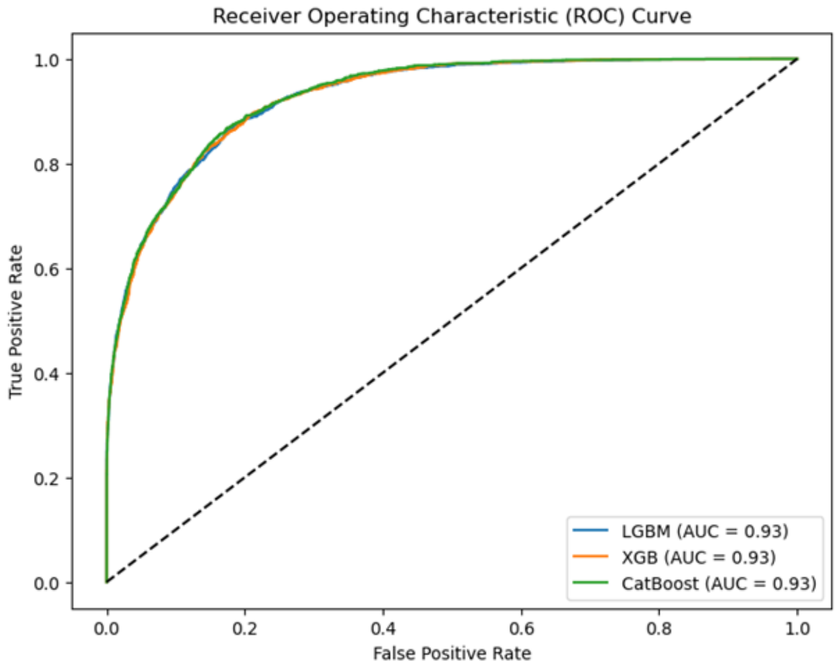


- As the size of the datasat increases, XGBoost slowed down significantly, and CatBoost also slowed down
- On the other hand, LightGBM showed stable speeds even when the dataset gets very large
- We find that LightGBM is extremely fast
- Concerning memory usage, all three models show similar behavior

Compare model performance

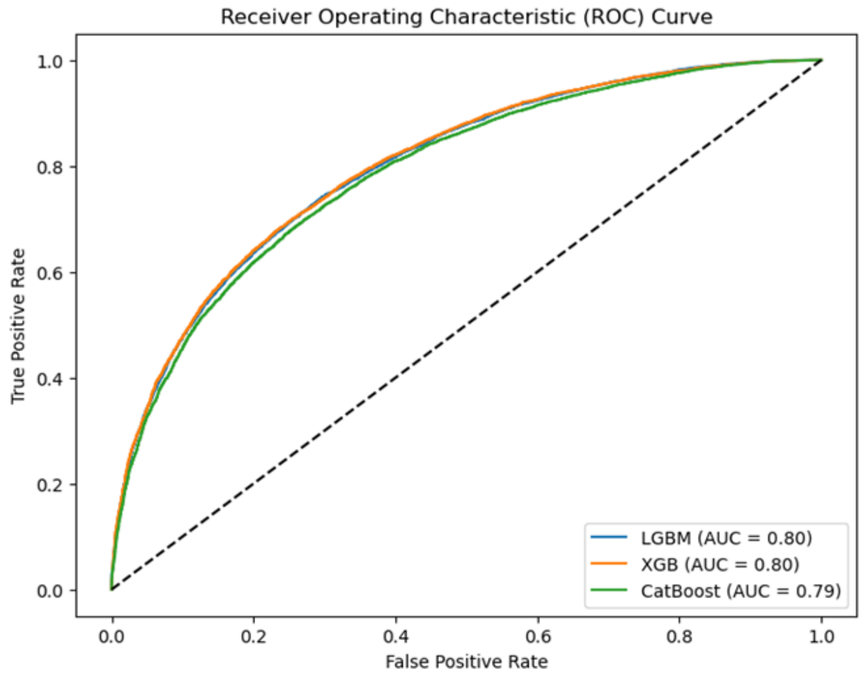


ROC Curve of credit-g



ROC Curve of adult

Compare model performance

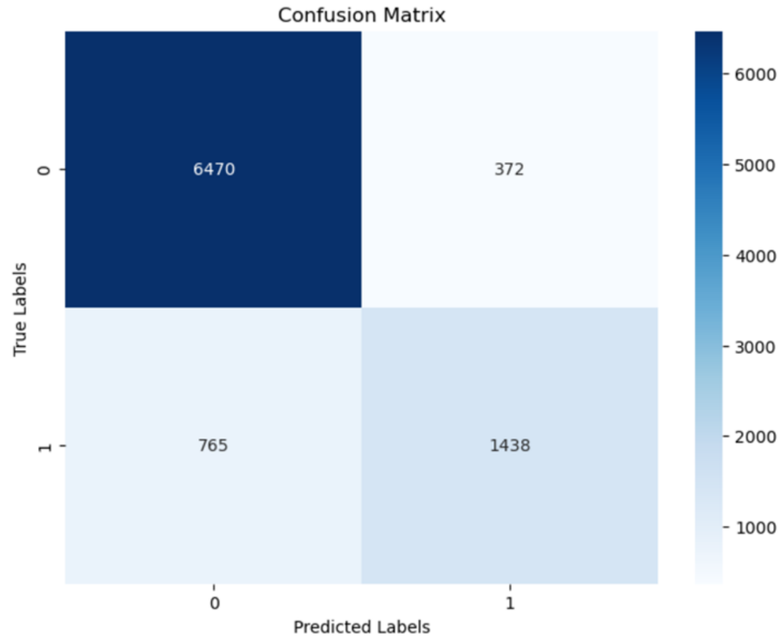


ROC Curve of higgs

- In the previous and current slide, we found the results of ROC Curve of each datasets, excluding covertype
- We concluded that a significant difference in performance wasn't existed

Compare model performance

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      6842
           1       0.79      0.65      0.72      2203

    accuracy                           0.87      9045
   macro avg       0.84      0.80      0.82      9045
weighted avg       0.87      0.87      0.87      9045
```




```
[37]: y_pred_lgbm = lgbm_model.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, y_pred_lgbm))

class_labels = np.unique(np.concatenate((y_test, y_pred_lgbm)))

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_lgbm), annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

- We checked the precision, recall, accuracy and f1-score and represented them as a confusion-matrix

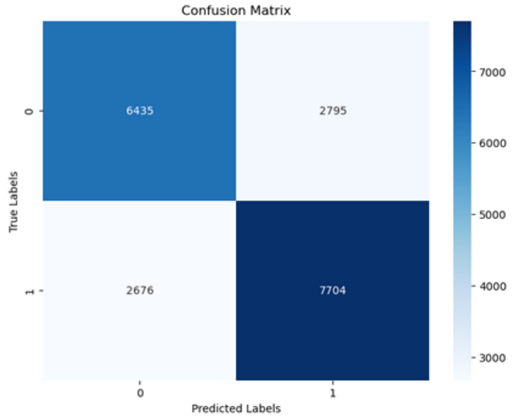
Compare with Logistic Regression

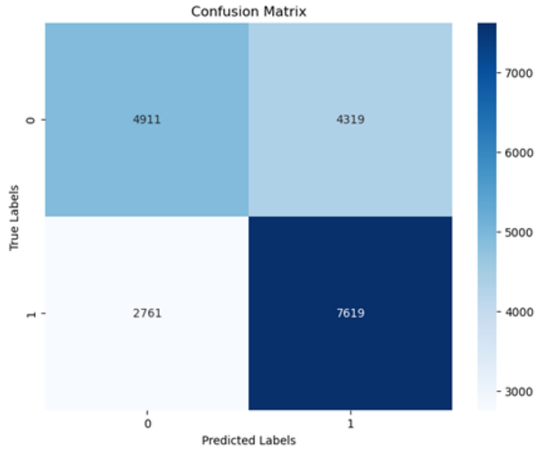
[17]:
```
log_pred = logreg.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, log_pred))

class_labels = np.unique(np.concatenate((y_test, log_pred)))

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, log_pred), annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

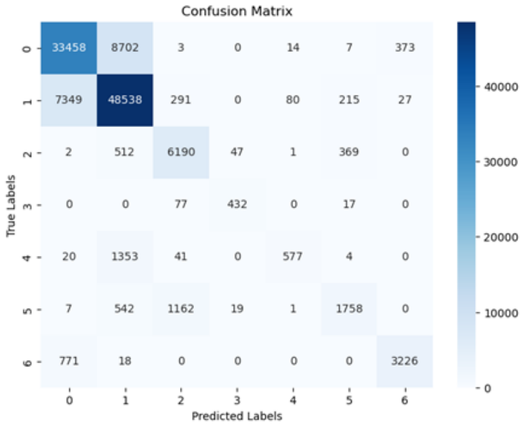
```
Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.53      0.58      9230
           1       0.64      0.73      0.68     10380

    accuracy                           0.64     19610
   macro avg       0.64      0.63      0.63     19610
weighted avg       0.64      0.64      0.63     19610
```

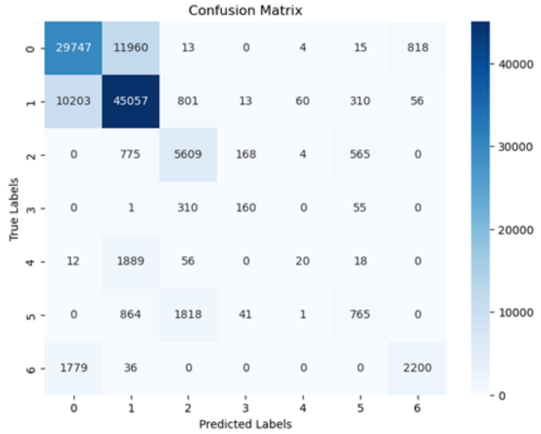
- Comparing performance with logistic regression, we tried to find how well the tree-based gradient boosting model performs

Compare with Logistic Regression









- Above is 'higgs' dataset,
- Below is 'covertype' dataset
- Left one is the confusion matrix of LightGBM
- Right one is the confusion matrix of logistic regression

Compare with Logistic Regression

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.95	0.92	6842
1	0.79	0.65	0.72	2203
accuracy			0.87	9045
macro avg	0.84	0.80	0.82	9045
weighted avg	0.87	0.87	0.87	9045

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.97	0.88	6842
1	0.72	0.27	0.39	2203
accuracy			0.80	9045
macro avg	0.76	0.62	0.63	9045
weighted avg	0.78	0.80	0.76	9045

- Using 'adult' dataset
- Above is the performance of LightGBM
- Below is the performance of Logisitic Regression

Compare with Logistic Regression

- As a result, it was clearly showed that the gradient boosting based models outperform logistic regression
- F1-score is also much higher for gradient boosting based models
- Prediction are also more accurate for gradient boosting based models

Conclusion

- In our research, we found that CatBoost has an advantage in datasets with high categorical values, but this was not the case when we applied the model
- Also, the performance difference between the three models did not increase significantly as the size of the data increased or as the number of features increased
- We concluded that there seems to be a slight difference between theory and practice

Conclusion

- In our project, we unified some hyperparameters in all cases, but in practice, it is possible to improve performance by varying them

- We learned that it is not important to use one model, but to test different models with different hyperparameter settings and cross-validation to find the best model

Thank you for your attention!