byu93-analysis

September 23, 2022

1 CS7641 Course Project: Supervised Learning

1.1 Part 1. Fraud Detection

Welcome! For this project, I decided to work on the fraud detection of credit cards. It is one of the classical classification problems in payment industry. As a data scientist working at Mastercard, fraud is one of the top 'headache' for the cardholders, the merchants and the banks. Clearly no one likes frauds.

However, fraud detection is a challenge problem, since the percentage of frauds among all card transactions is very small, which totally makes sense - a high fraud rate will cause significant financial damage to the card holder, banks and payment network companies. As a result, most of the fraud datasets are very imbalanced. We will get some hands on experience of modeling with imbalanced dataset. Common strategies include using up/downsampling on the training data and precision/recall as measurement of success. Because we can easily achive a high accuracy by simply predict everything as the majority class - which means nothing in this problem.

The dataset was found in Kaggle, link provided as follows: https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud

1.1.1 Environment Setup

Before we 'officially' start the ML job, let's make sure the environment is successfully set up. First, let's confirm the python version is 3.9.10 using the following code:

```
Current version of Python is  3.9.10 (main, Jan 15 2022, 11:48:00)
[Clang 13.0.0 (clang-1300.0.29.3)]
```

After the python version is confirmed, we can install all required packages through the code below.

Now let's import all required packages for further usage.

1.1.2 Exploratory Data Analysis

The EDA session is the key to the success of the model development work. Before train any models, we need to understand the data, analyze the distributions, identify outliers etc. So first of all, let's load the data into memory and see what kind of information is included.

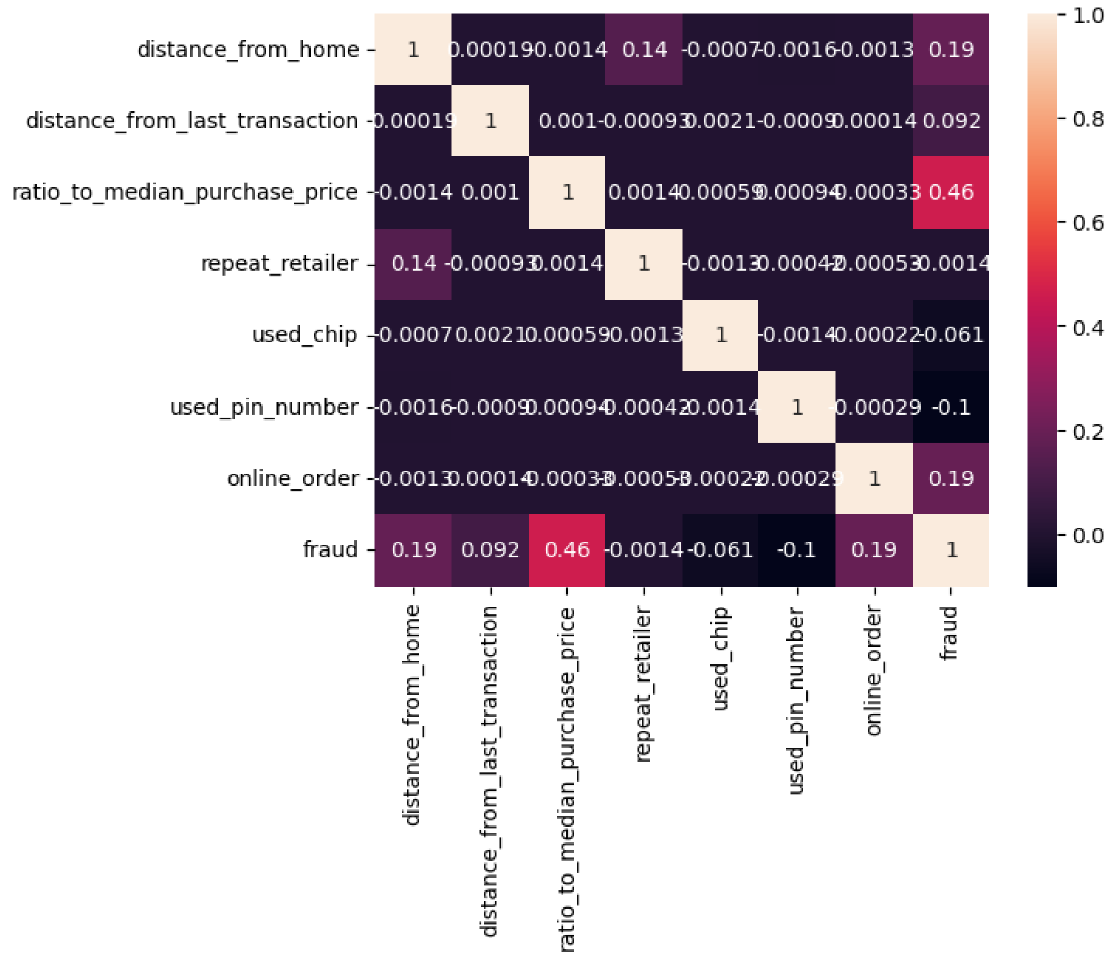
Feature Explanation (from Kaggle):

- distancefromhome - the distance from home where the transaction happened
- distancefromlast_transaction - the distance from last transaction happened
- ratiotomedianpurchaseprice - Ratio of purchased price transaction to median purchase price

- repeat_retailer - Is the transaction happened from same retailer
- used_chip - Is the transaction through chip (credit card)
- usedpinnumber - Is the transaction happened by using PIN number
- online_order - Is the transaction an online order
- fraud - Is the transaction fraudulent

Next step is to check the correlations between features:

```
[156]: <AxesSubplot:>
```



	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order	fraud
distance_from_home	1	0.00019	-0.0014	0.14	-0.0007	-0.0016	-0.0013	0.19
distance_from_last_transaction	0.00019	1	0.001	-0.00093	0.0021	-0.0009	0.00014	0.092
ratio_to_median_purchase_price	-0.0014	0.001	1	0.0014	0.00059	0.00094	0.00033	0.46
repeat_retailer	0.14	-0.00093	0.0014	1	-0.0013	0.00042	-0.00053	-0.0014
used_chip	-0.0007	0.0021	0.00059	-0.0013	1	-0.0014	-0.00022	-0.061
used_pin_number	-0.0016	-0.0009	0.00094	0.00042	-0.0014	1	-0.00029	-0.1
online_order	-0.0013	0.00014	0.00033	-0.00053	-0.00022	-0.00029	1	0.19
fraud	0.19	0.092	0.46	-0.0014	-0.061	-0.1	0.19	1

Some highlights are: 1. We have 1,000,000 data points and 8 columns in the data 2. The label column "fraud" is imbalanced, given there are roughly 9% is frand 3. There are some extreme values existing in the numeric columns (distance_from_home, distance_from_last_transaction, ratio_to_median_purchase_price), since the max value is far different from the 75% quantile value 4. The heatmap shows the ratio_to_median_purcahse_price is highly correlated to the fraud, this makes sense since the pattern of transactions is very different from the rountine behaviour, it's very likely to be a fraud

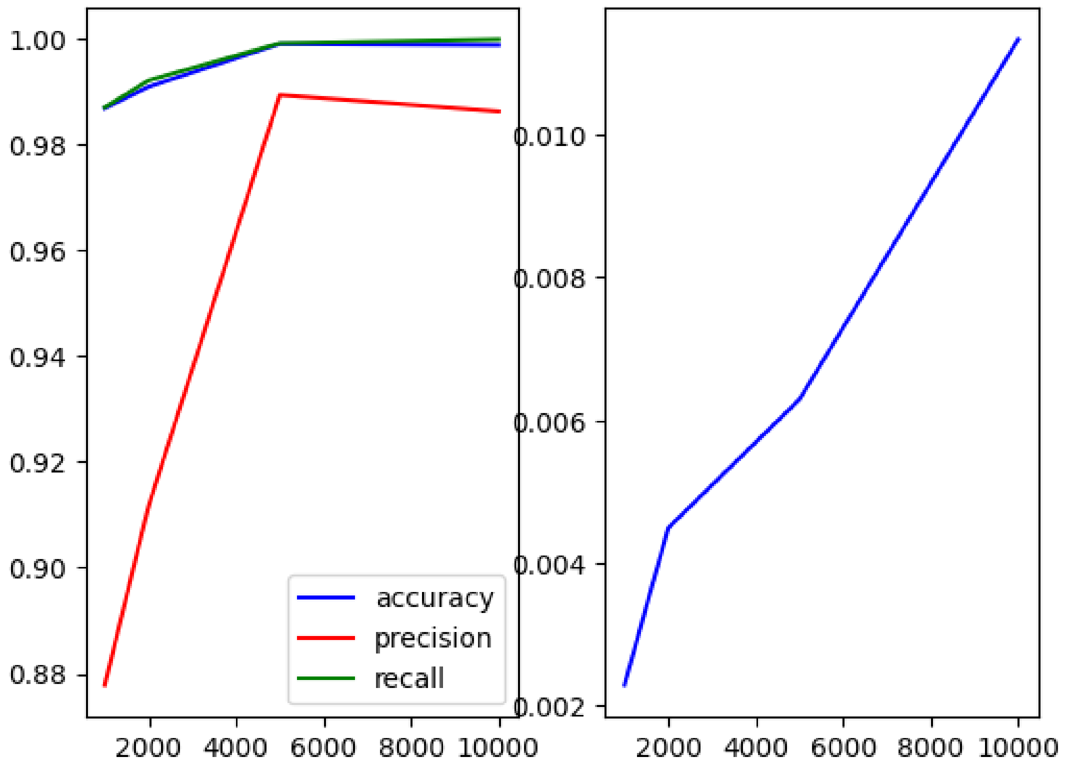
1.1.3 Sampling the Training Data

First, we need to split our downsampled data into training (80%) and testing (20%). A typical way of handling large imbalanced dataset is sampling. In this project, I chose to downsample the majority class to balance the label distribution before running models. We will also do some sampling on the training dadtaset to ensure the running time is feasible. The training dataset sizes in this trail are 1000, 2000, 5000, 10000.

1.1.4 Model Development

Decision Tree We will implement the Decision Tree algorithm with post-pruning and 5-folder cross-validation.

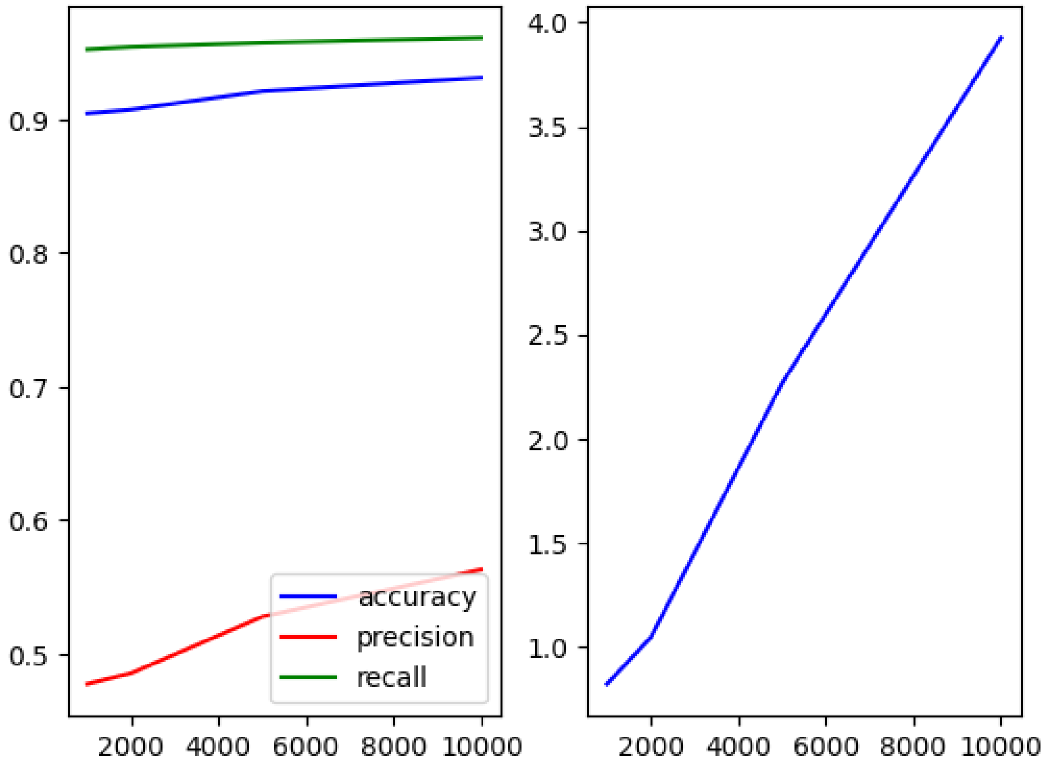
For each training dataset with different size, we will train the decision tree with post-pruning. The post-pruning strategy requires a full-path model training as set up. The full-path tree is the decision tree trained without any pruning. Then we can start testing the pruning with different parameters using cross validation. During this process, we will use accuracy, precision, and recall score to evaluate the result. Note that with imbalanced data, accuracy is not the most important KPI as the key of success is to classify the minority class correctly.



In the first plot, x-axis represents training size, while y axis represents the accuracy/precision/recall, ranged between 0 to 1. The second plot shows the relationship between training size vs training time.

Observations from the plots: 1. Accuracy and recall seem to be "stable"; as training size increases, minor improvements are observed. 2. The precision score increases “dramatically” from training size 2000 to 5000. From 5000 to 10000, it shows a slight decreasing trend. 3. Running time linearly increases as training size increases.

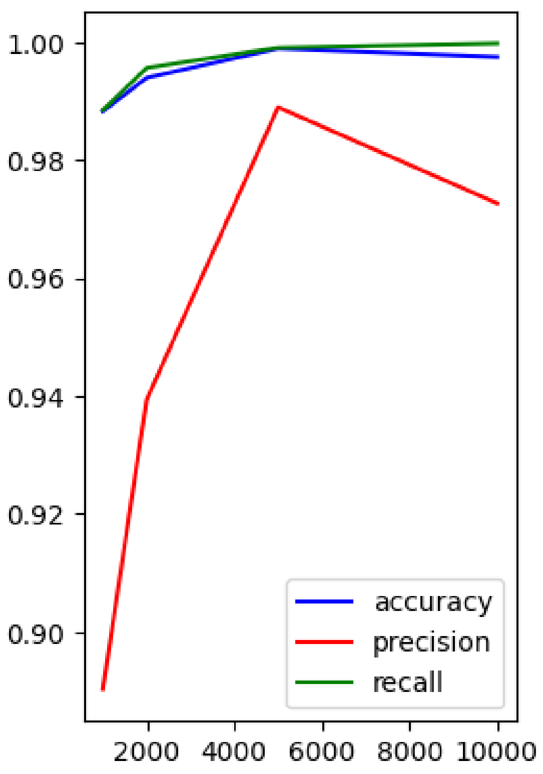
Neural Network To experiment with different settings of neural network, we use grid search + cross-validation to test the accuracy and precision scores on the training dataset. Considering the running time, we only tested a subset of possible settings. We will use 5-folder for CV due to the long processing time of 10-folder.

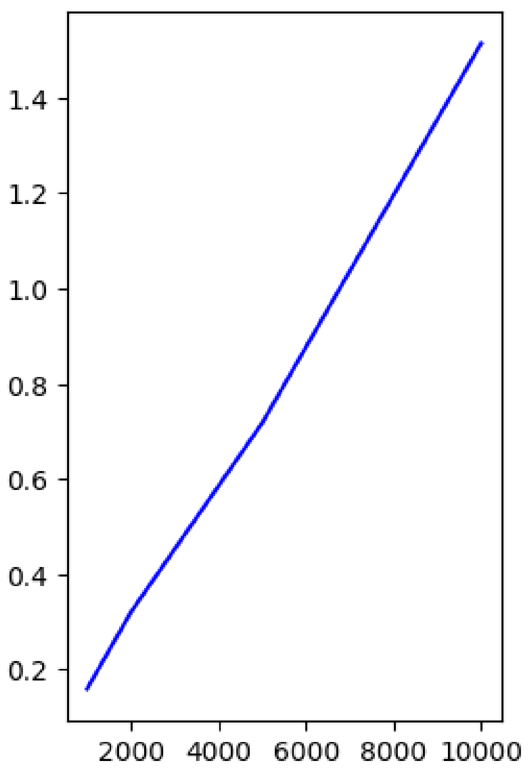


In the first plot, x-axis represents training size, while y axis represents the accuracy/precision/recall, ranged between 0 to 1. The second plot shows the relationship between training size vs training time.

Observations from the plots: 1. Accuracy and recall seem to be “stable”; as training size increases, minor improvements are observed. 2. There is a big gap between precision vs accuracy & recall, meaning the model tends to predict more ‘frauds’ than the true distribution. 3. The precision score increases “slightly” as training size increases, keep on increasing the training size may help improve the performance. 4. Running time linearly (almost) increases as training size increases.

Boosting We will experiment with gradient boosting tree for this work. We expand the max_depth of the tree from 3 (default) to different value as pre-pruning and used post-pruning to cut-off branches. Then we evaluate the model performance on the downsampled training data by accuracy, precision, and recall. As a result, max_depth=7 seems to be the sweet spot for this problem. Lastly, we evaluated the model on the test dataset.

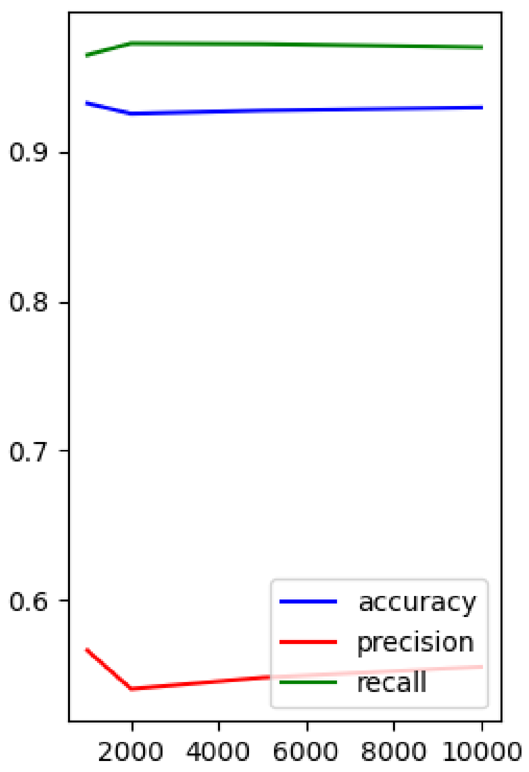


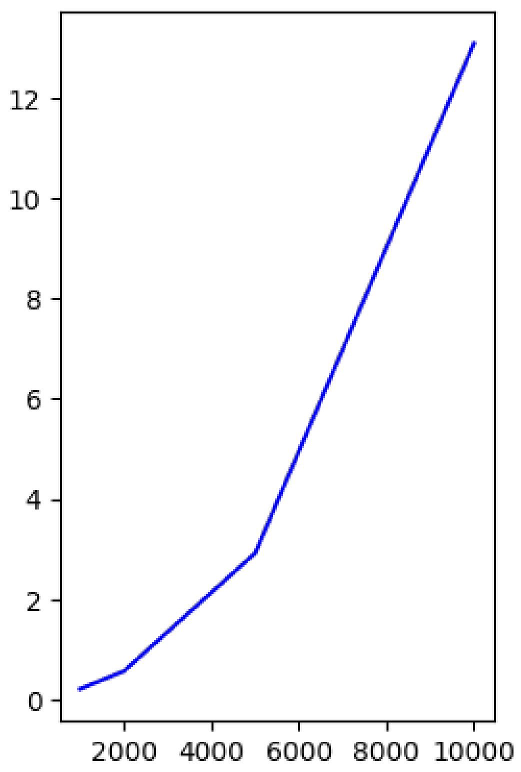


The result shows that the model is well performed on both training (balanced) and testing (imbalanced). In the first plot, x-axis represents training size, while y axis represents the accuracy/precision/recall, ranged between 0 to 1. The second plot shows the relationship between training size vs training time.

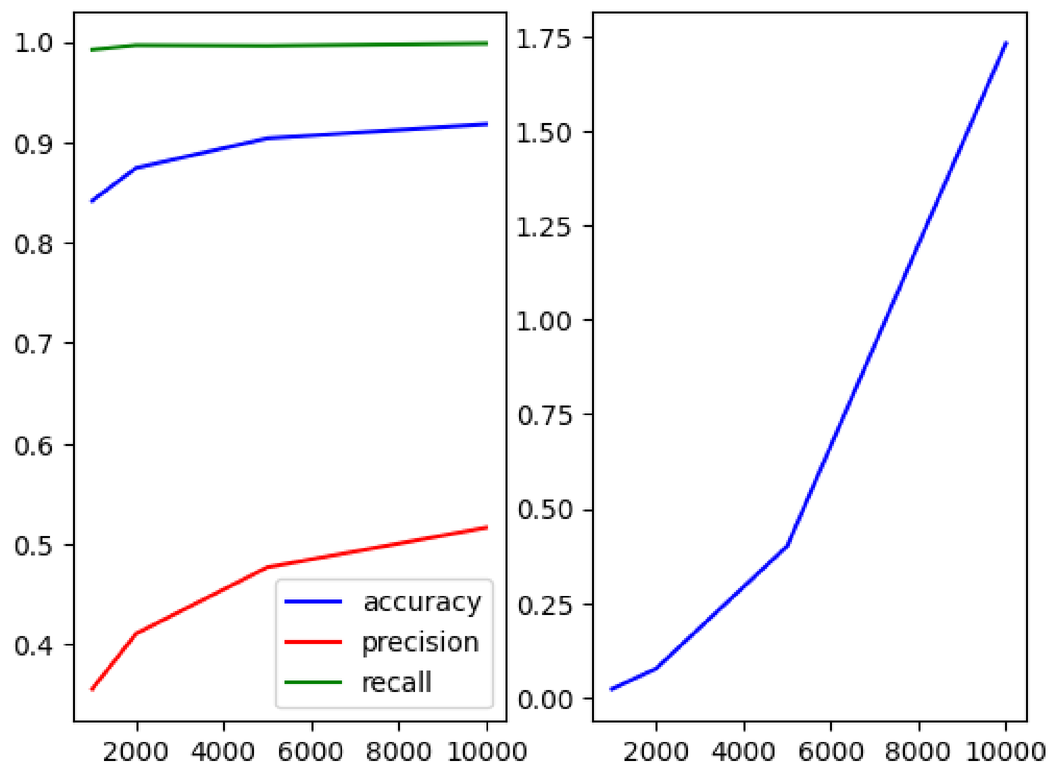
Observations from the plots: 1. Accuracy and recall seem to be "stable"; as training size increases, minor improvements are observed. 2. The best precision score appears at 5000 as training size; it seems to be the sweet spot, we need to keep experimenting with larger trainig size to conduct a conclusion. 3. Running time linearly increases as training size increases.

SVM With SVM, we will experiment with different kernel - linear and rbf.





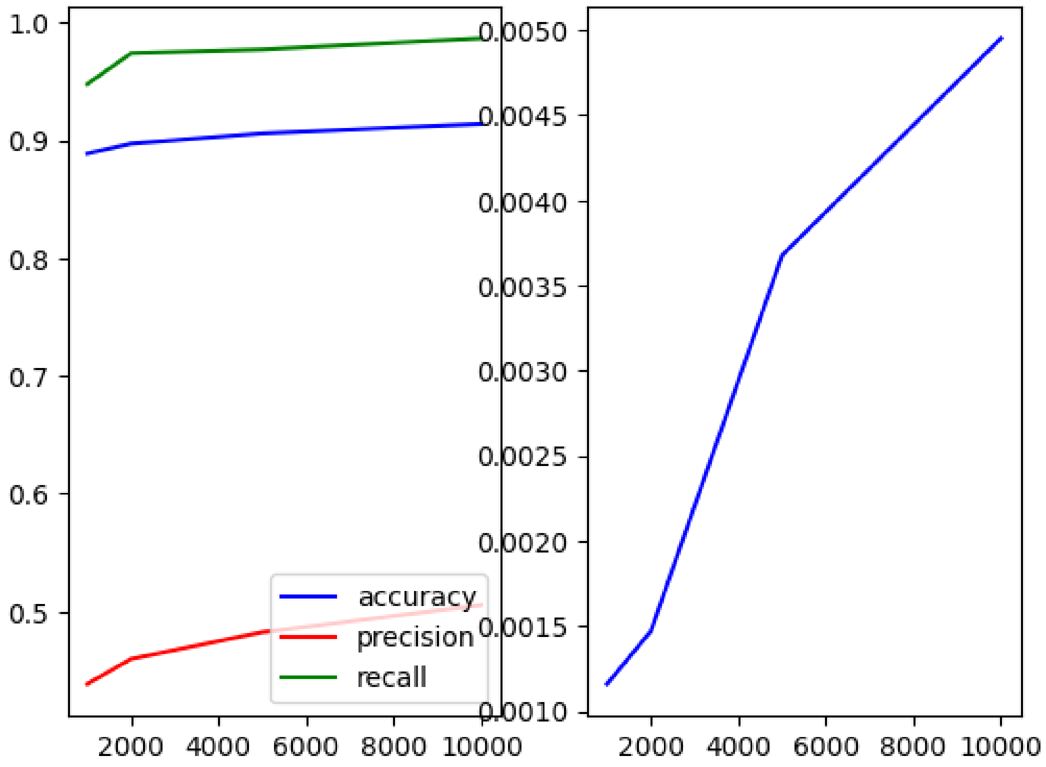
The running time also increases as training size increases, but not in a linear way.

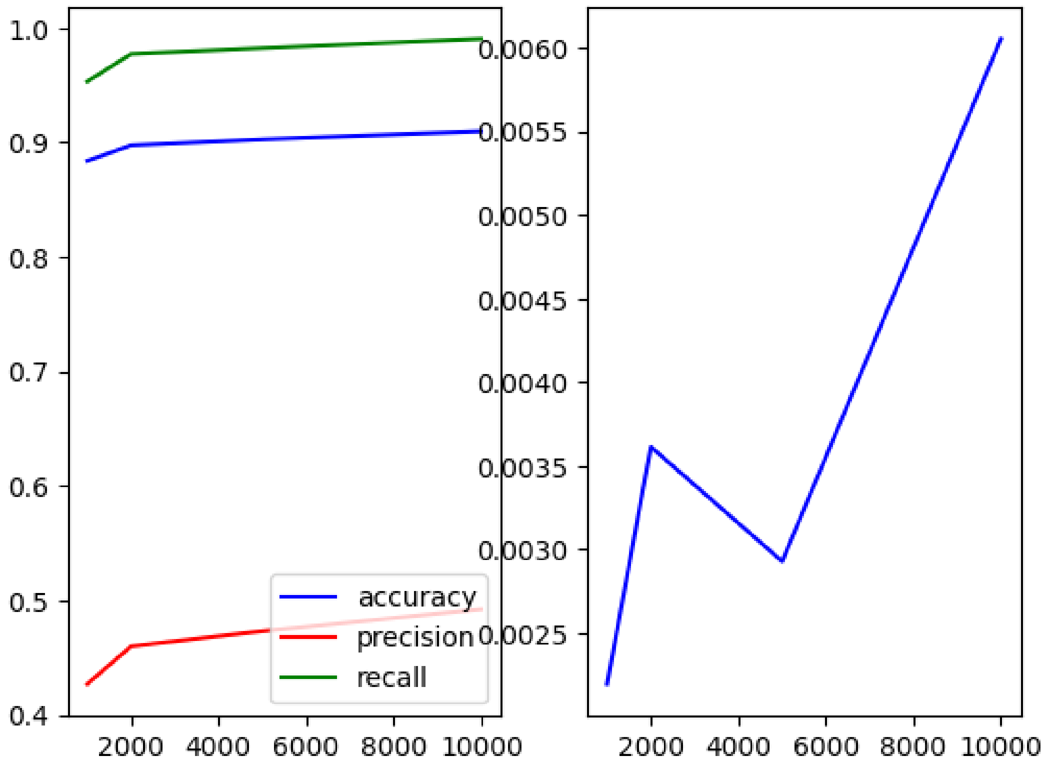


Similar pattern observed as the linear kernel, while we see the precision score drops when switching from training to testing.

The running time increases as training size increases, also in a non-linear way.

KNN Lastly, we will experiment with KNN as k=3 and 5.


1.0
0.9
0.8
0.7
0.6
0.5
accuracy
precision
recall
2000
4000
6000
8000
10000
0.0050
0.0045
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010


1.0
0.9
0.8
0.7
0.6
0.5
0.4
accuracy
precision
recall
2000
4000
6000
8000
10000
0.0060
0.0055
0.0050
0.0045
0.0040
0.0035
0.0030
0.0025

The KNN performs has a better precision score on training (balanced dataset). The running time keep increasing as training size increases.

Conclusion We used accuracy, precision and recall score to define the success of the model. As a result, the Decision Tree and Gradient Boosting Classifier won the game of handling imbalanced data prediction, while Neural Network, SVM and KNN did not do a great job. This is probably because the tree model can handle non-linear relationship better than other algorithms. To improve the model performance, we did some grid search to determine the best hyperparameters for each model and experimented with different training data size. Efficiency wise, the trees and KNN are faster than SVM and Neural Network. The result may change as the hyperparameter changed. We also used cross valdiation to prevent overfitting. For each algorithm, we experimented with at least two settings of parameters.

1.2 Part 2. Data Science Salary

For a second trial, I experienced with the data scientist salary dataset. This is a interesting dataset that is available on Kaggle. As a wokring data scientist… Of course money matters :). I want to know which features are important to decide a data scientist' salary, maybe location? years of experience? or industry.

Link: https://www.kaggle.com/datasets/whenamancodes/data-science-fields-salary-categorization

First of all, let's read in the data first.

We also transfer the amount from Rupee to USD using FX rate ~ 80. The salary mean in USD is about 111k. And so, we created the label: >100k or not, then split the data as 80% training and 20% testing. Since this data set is small, we will not experiment with different training size, but use everything for training.

Now we will rerun all models as part 1 and see the performances.

```
Decision Tree:
Accuracy score on test data: 0.836066

Neural Network:
In sample accuracy score: 0.519588
Accuracy score on test data: 0.434426

Gradient Boosting:
In sample accuracy score: 0.795876
Accuracy score on test data: 0.819672

SVM with linear kernel:
In sample accuracy score: 0.818557
Most important feature (positive): EN -1.9997670858356926
Most important feature (negative): same_loc 1.998656728325571
Accuracy score on test data: 0.852459
```

```
SVM with rbf kernel:
In sample accuracy score: 0.793814
Accuracy score on test data: 0.811475

KNN with k=3:
In sample accuracy score: 0.740206
Accuracy score on test data: 0.762295

KNN with k=5:
In sample accuracy score: 0.779381
Accuracy score on test data: 0.778689
```

Based on the result, the SVM with linear kernel is the winning model as the accuracy is the highest. As the salary is actually a continuous value, the tress might got confused when people with similar profile, but one got over 100k while the other not. The actual situation could be their salary is 99k vs 101k, which is very close. By looking at the model coefficients, the feature same_loc (meaning whether or not the employee is living in the same location as where the company located) is the most important feature, which makes sense as company is willing to pay more to people who work close to the HQ.