Report of Q1
Supervised Outlier Detection

50015627 JIANG Zhuoyang

1. Target: Supervised Outlier Detection

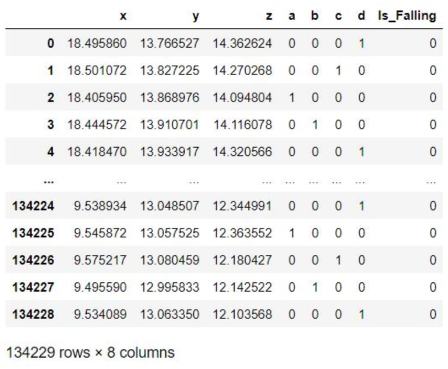
In our class, we've learned that supervised anomaly detection methods face a significant challenge. On one hand, within the training data, the quantity of anomalous data is relatively small compared to normal time series, impacting the detection performance. Hence, these methods see limited practical application in anomaly detection.

This experiment confronts the deficiencies of Supervised Outlier Detection in real-world applications. To address this, we need to clarify the objectives this task aims to achieve in practical scenarios:

- **Primary Goal:** We aim to promptly detect "falls of cats." We prioritize timely detection over false alarms, preferring not to miss any potential falls.
- **Secondary Goal:** Minimizing false alarms for detecting "falls of cats" is also a priority; we seek to reduce such instances as much as possible.

2. Basic: Data Loading

Firstly, by observing the given data format, we found that the training set data is scattered in 20 different tables. We merged them and imported them to construct the training set. The imported training set is roughly as Figure.1. Afterwards, import the test set, which contains 6623 pieces of data.



	x	y	z	a	b	c	d	Is_Falling
0	18.495860	13.766527	14.362624	0	0	0	1	0
1	18.501072	13.827225	14.270268	0	0	1	0	0
2	18.405950	13.868976	14.094804	1	0	0	0	0
3	18.444572	13.910701	14.116078	0	1	0	0	0
4	18.418470	13.933917	14.320566	0	0	0	1	0
...	...	...	...	...	...	...	...	...
134224	9.538934	13.048507	12.344991	0	0	0	1	0
134225	9.545872	13.057525	12.363552	1	0	0	0	0
134226	9.575217	13.080459	12.180427	0	0	1	0	0
134227	9.495590	12.995833	12.142522	0	1	0	0	0
134228	9.534089	13.063350	12.103568	0	0	0	1	0

134229 rows × 8 columns

Figure.1 Training Data

3. Core: Data Augmentation Methods in Imbalanced Data Set

(1) Random Over Sampler ：

The simplest approach for severely imbalanced training samples is to randomly sample minority samples and then replicate them to increase the sample size to a certain extent, in order to achieve the goal of balanced sample distribution.

The advantage of this method is that it is simple and easy to implement, and can effectively improve the performance of the model on imbalanced datasets.

RandomOverSampler is a method used in the imbalanced learn library to randomly replicate and increase the number of minority class samples.

(2) SMOTE:

Other than simple duplication, we can also attempt to generate additional samples using an interpolation-like method, where SMOTE (Synthetic Minority Over-sampling Technique) comes into play. It's a method of oversampling by synthesizing minority class samples, balancing the dataset by creating new samples based on existing minority class samples. Its workings can be described through the following steps and formula:

① Selecting minority class samples: For each minority class sample xi, identify its k nearest neighboring samples.

② Random selection of neighbor sample: Randomly pick one nearest neighbor sample xj of xi and label it as xnew.

③ Generating a new sample: For each feature dimension (each element in the feature vector), compute the feature value of the new sample.

④ For each feature value xnew_k in a feature dimension, where k is a random number between [0, 1].

⑤ Calculate the new sample's feature value:

$$\mathrm{xnew}_k = \mathrm{xi}_k + (\mathrm{xj}_k - \mathrm{xi}_k) * k.$$

⑥ This process constructs a new synthetic sample xnew in the feature space.

(3) SMOTE-ENN

Using oversampling alone may generate some noisy data, so we naturally think of further optimizing data quality by combining undersampling with oversampling. SMOTE-ENN (SMOTE with Edited Near Neighbors) is a method that combines oversampling (SMOTE) and undersampling (ENN). ENN (Edited Near Neighbors): It is an undersampling method that improves the balance of a dataset by identifying and removing noise and uncertain samples from the majority class samples.

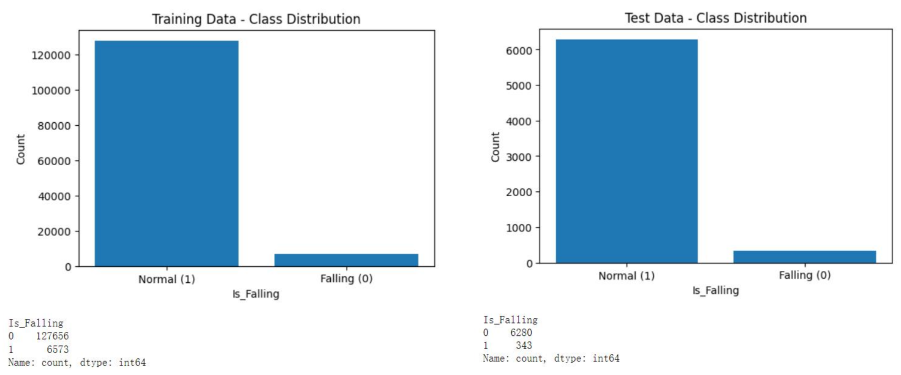
After oversampling the data using SMOTE for minority classes, the ENN algorithm checks each sample and removes samples that are judged by most of its neighboring samples as not belonging to this class. The steps are summarized as follows:

① For each sample x, find its k nearest neighbors.

② If the sample x and most of its nearest neighbors belong to different categories, remove the sample x from the dataset.

4. Experiment: Supervised Outlier Detection after Data Augmentation

(1) Imbalanced Visualization:



```
Is_Falling
0    127656
1      6573
Name: count, dtype: int64
```

```
Is_Falling
0    6280
1     343
Name: count, dtype: int64
```

Figure.2 Imbalanced Train set and Test set

(2) Data Augmentation:

Use imblearn.over_sampling library 's method - RandomOverSampler and SMOTE methods of the sampling library are used to enhance the oversampling data of the training set, and then the SMOTEENN method of the imblarn.combine library is called to perform oversampling (SMOTE) and undersampling (ENN) on the training set. The training set effects of using three data augmentation schemes are shown in the following figure:

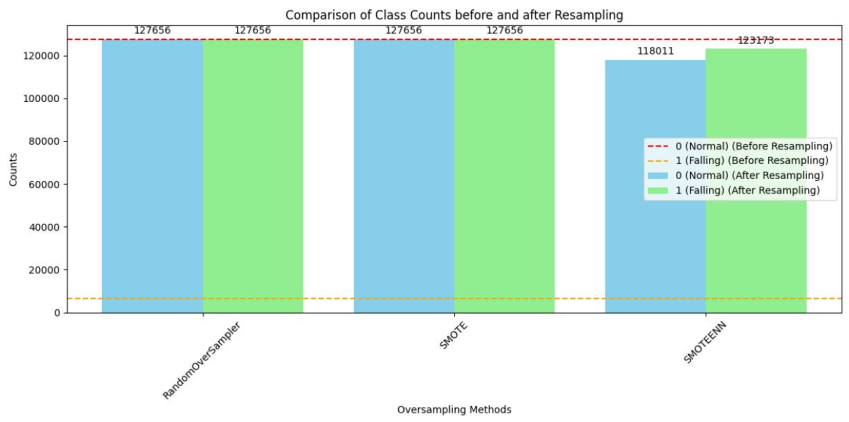


Figure.3 Data Augmentation Methods' Result

(3) Supervised Model Training and Evaluation

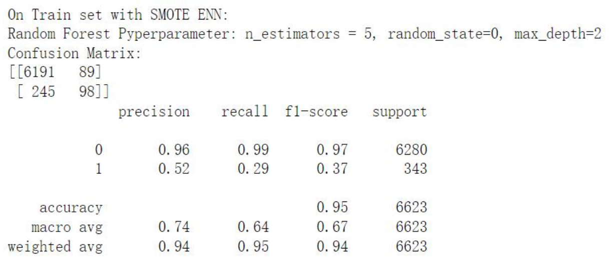
Three different data augmentation methods were used to supplement the imbalanced training set with minority class samples, resulting in three resampled training sets. These three training sets were used in sequence to train different Random Forest Classifiers, and the best model was selected as the final model to be used.

5. Result:

(1) Bad Model Example:

① Bad Example 1: Great Accuracy but Bad Recall

The model below has a bad result. For samples labeled as 1, recall is more important because we want to detect "cats falling" in a timely manner. Compared to the fact that the number of reminders for "cats falling" is more than the actual situation, the worse result is that the number of reminders for "cats falling" is less than the actual situation

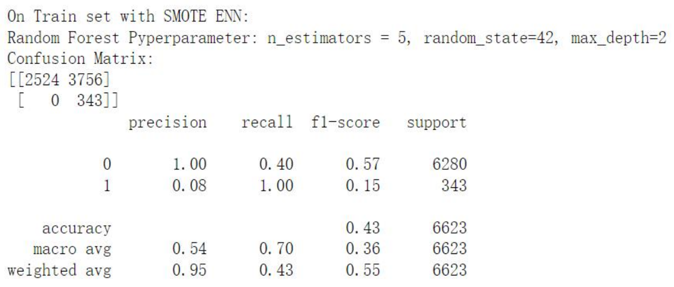


```
On Train set with SMOTE ENN:
Random Forest Pyperparameter: n_estimators = 5, random_state=0, max_depth=2
Confusion Matrix:
[[6191   89]
 [ 245   98]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      6280
           1       0.52      0.29      0.37       343

    accuracy                           0.95      6623
   macro avg       0.74      0.64      0.67      6623
weighted avg       0.94      0.95      0.94      6623
```

Figure.4 Great Accuracy but Bad Recall

② Bad Example 2: Best Recall with Unbearable Accuracy

The model below also has a bad result. For samples labeled as 1, recall is more important, but if there are too many false positives (low accuracy), the alarm loses its meaning.

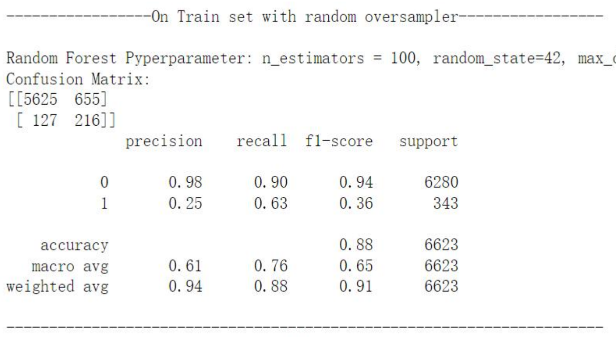


```
On Train set with SMOTE ENN:
Random Forest Pyperparameter: n_estimators = 5, random_state=42, max_depth=2
Confusion Matrix:
[[2524 3756]
 [   0  343]]
              precision    recall  f1-score   support

           0       1.00      0.40      0.57      6280
           1       0.08      1.00      0.15       343

    accuracy                           0.43      6623
   macro avg       0.54      0.70      0.36      6623
weighted avg       0.95      0.43      0.55      6623
```

Figure.5 Best Recall with Unbearable Accuracy

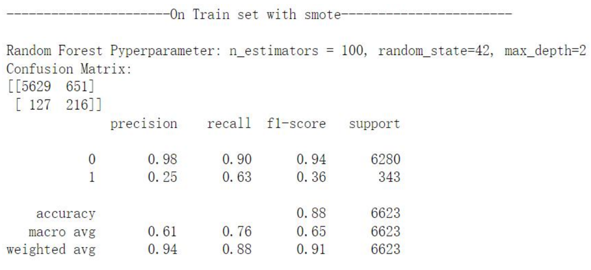
(2) Comparison of model depending on different Oversampling Methods:

I adjusted the hyperparameters and obtained a model that can meet the two objectives as much as possible. The hyperparameters and model effects are as follows:



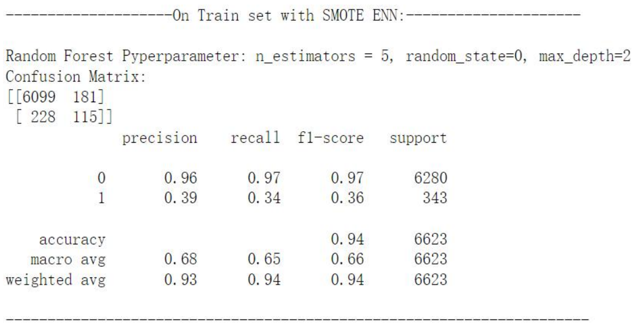
```
-----------------On Train set with random oversampler-----------------

Random Forest Pyperparameter: n_estimators = 100, random_state=42, max_depth=2
Confusion Matrix:
[[5625  655]
 [ 127  216]]
              precision    recall  f1-score   support

           0       0.98      0.90      0.94      6280
           1       0.25      0.63      0.36       343

    accuracy                           0.88      6623
   macro avg       0.61      0.76      0.65      6623
weighted avg       0.94      0.88      0.91      6623

----------------------------------------------------------------------
```

Figure.6 Model on Train set with random oversampler



```
---------------------On Train set with smote----------------------

Random Forest Pyperparameter: n_estimators = 100, random_state=42, max_depth=2
Confusion Matrix:
[[5629  651]
 [ 127  216]]
              precision    recall  f1-score   support

           0       0.98      0.90      0.94      6280
           1       0.25      0.63      0.36       343

    accuracy                           0.88      6623
   macro avg       0.61      0.76      0.65      6623
weighted avg       0.94      0.88      0.91      6623

----------------------------------------------------------------------
```

Figure.7 Model on Train set with Smote



```
--------------------On Train set with SMOTE ENN:---------------------

Random Forest Pyperparameter: n_estimators = 5, random_state=0, max_depth=2
Confusion Matrix:
[[6099  181]
 [ 228  115]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      6280
           1       0.39      0.34      0.36       343

    accuracy                           0.94      6623
   macro avg       0.68      0.65      0.66      6623
weighted avg       0.93      0.94      0.94      6623

----------------------------------------------------------------------
```

Figure.8 Model on Train set with Smote-enn

From the above model results, it can be seen that **“Model on Train set with Smote”** best satisfies our two objectives.

It prioritizes meeting the first objective, although sacrificing a little accuracy, the recall rate is already quite good compared to other models.

Therefore, although this result is somewhat average, I still attribute it to the limitations of supervised anomaly detection.

(3) Obtaining the Output

So, I ultimately chose to apply "Model on Train set with Smote" to the test data. And obtained the Q1_output.csv.