Exoplanet Hunting in Deep space

2016112564 정용희

2016112607 이동준

2017112535 온영재

Index

Index

Data

The Search for New Earths

The data describe the change in flux (light intensity) of several thousand stars.
If a star has a planet or multiple planets, there may be a regular 'dimming' of the flux, because planets orbit stars.

Label 1 : Non-Exoplanet star / Label 2 : Exoplanet star

Description

- **Trainset:**
  - 5087 rows or observation
  - 3198 columns or features
  - Column 1 is the label vector. Columns 2 – 3198 are the flux values over time.
  - 37 confirmed exoplanet-stars and 5050 non-exoplanet-stars.

- **Testset:**
  - 570 rows or observation
  - 3198 columns or features
  - Column 1 is the label vector. Columns 2 – 3198 are the flux values over time
  - 5 confirmed exoplanet-stars and 565 non-exoplanet-stars.

Index

Data preprocessing

2.1 Missing Values

Loading Data

Summary

	LABEL
1	5050
2	37

Label 1: Non-exoplanet stars
Label 2: Exoplanet stars

```
Exoplanet stars are 0.73% of total.
```

Checking for Missing Values

```
Total Missing values in train data : 0
```

Choose Column/Row

None

Data preprocessing

2.2 Outliers

With IsolationForest, contamination_rate = 1%

Code

```
from sklearn.ensemble import IsolationForest

clf = IsolationForest(n_estimators=100, max_samples='auto',
                      contamination=float(0.01),
                      max_features=1.0, bootstrap=True,
                      n_jobs=-1, random_state=0, verbose=0)
clf.fit(exo_train.iloc[:, 1:])
exo_train['anomaly'] = clf.predict(exo_train.iloc[:, 1:])
```

	ANOMALY
1	5000
-1	51

(1) : non-Outlier data
(-1) : Outlier data

Removing Outliers

Code

```
exo_train.drop(exo_train.loc[exo_train['anomaly']==-1].index, inplace=True)
exo_train.drop(['anomaly'], axis='columns', inplace=True)
```

	LABEL
1	5000
2	36

Label 1: Non-exoplanet-stars
Label 2: Confirmed-exoplanet-stars

Data preprocessing

2.3 Scaling & Dimension Reduction

Data Scaling

with StandardScaler

Code

```
from sklearn.preprocessing import StandardScaler

X_train = exo_train.iloc[:, 1:]
X_test = exo_test.iloc[:, 1:]
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
y_train = exo_train[['LABEL']]
y_test = exo_test[['LABEL']]
```

Dimension Reduction

with Principal Component Analysis

Code

```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(X_train_scaled)
cumsum = np.cumsum(pca.explained_variance_ratio_)
dimension = np.argmax(cumsum>=0.95)
pca = PCA(n_components=dimension)
pca.fit(X_train_scaled)
pca_X_train = pca.transform(X_train_scaled)
pca_X_test = pca.transform(X_test_scaled)
```

Number of dimensions with 95% variance : 73

Size of the train data : (5036, 73)

Index

Classification

SVM + Logistic Regression, without handling imbalance

SVM with Polynomial Kernel

	0	1
0	565	0
1	5	0

	precision	recall	f1-score	support
1	0.99	1.00	1.00	565
2	0.00	0.00	0.00	5
accuracy			0.99	570
macro avg	0.50	0.50	0.50	570
weighted avg	0.98	0.99	0.99	570

SVM with RBF Kernel

	0	1
0	565	0
1	5	0

	precision	recall	f1-score	support
1	0.99	1.00	1.00	565
2	0.00	0.00	0.00	5
accuracy			0.99	570
macro avg	0.50	0.50	0.50	570
weighted avg	0.98	0.99	0.99	570

SVM with Sigmoid Kernel

	0	1
0	564	1
1	5	0

	precision	recall	f1-score	support
1	0.99	1.00	0.99	565
2	0.00	0.00	0.00	5
accuracy			0.99	570
macro avg	0.50	0.50	0.50	570
weighted avg	0.98	0.99	0.99	570

Logistic Regression

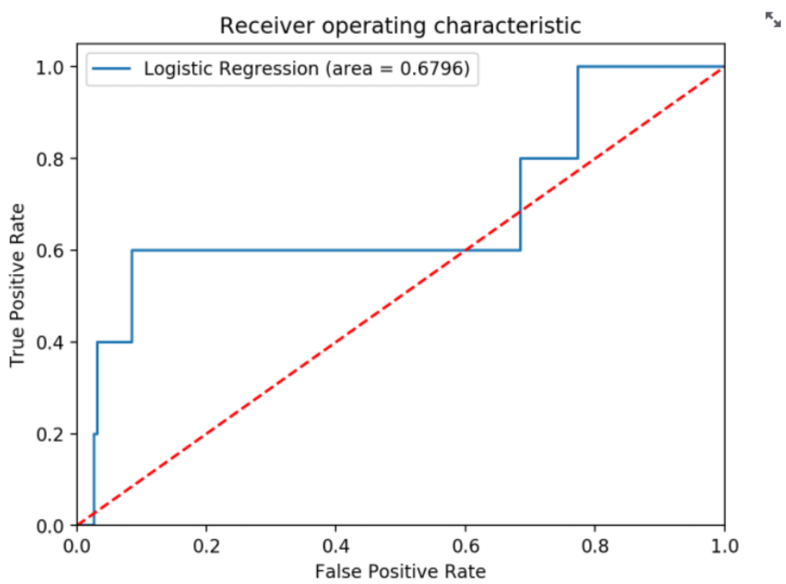
☑ Result

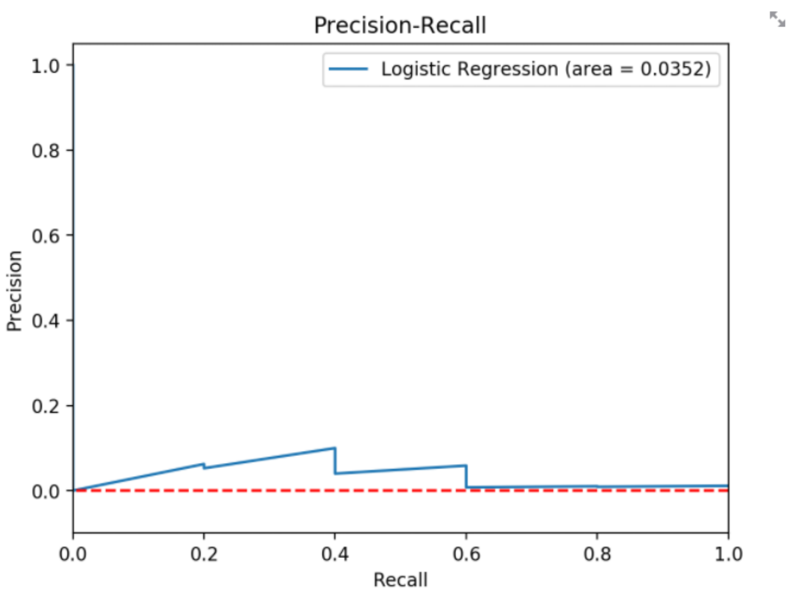
	0	1
0	564	1
1	5	0

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	565
1.0	0.00	0.00	0.00	5
accuracy			0.99	570
macro avg	0.50	0.50	0.50	570
weighted avg	0.98	0.99	0.99	570

Classification

ROC-curve & PR-curve





Because the class of data is highly imbalanced (5000 : 36), we can see that
PR Curve is a much better indicator for performance evaluation, than ROC Curve.

ROC Curve overestimates the model when the data is highly imbalanced.

Classification

SMOTE algorithm

> We use SMOTE algorithm to handle the imbalance of the data. SMOTE algorithm is type of the over-sampling technique.

Code

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=0)
smote_X_train, smote_y_train = smote.fit_sample(X_train, y_train['BINARY'])
smote_y_train = smote_y_train.astype('int')
smote_y_train = smote_y_train.values.tolist()
```

Summary

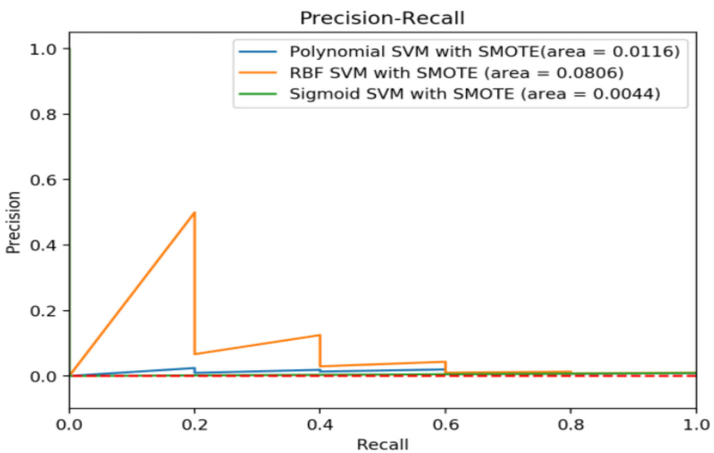
	BINARY
1	5000
0	5000

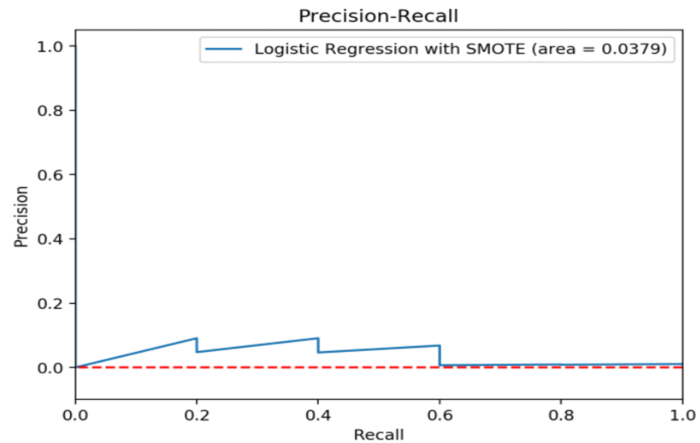
```
Exoplanet stars are 50.00% of total
```

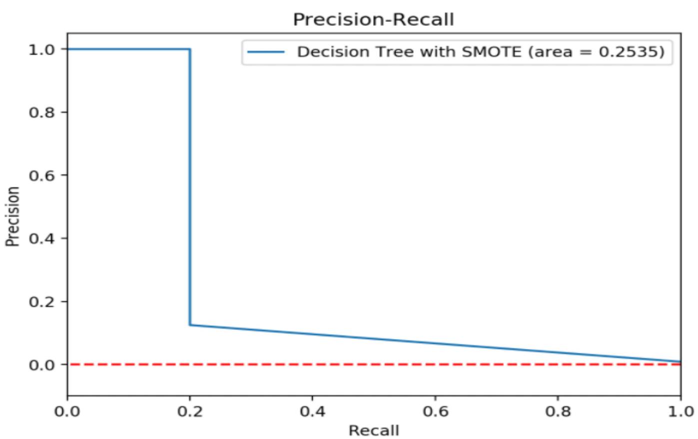
class 0: Non-exoplanet-stars
class 1: Confirmed-exoplanet-stars

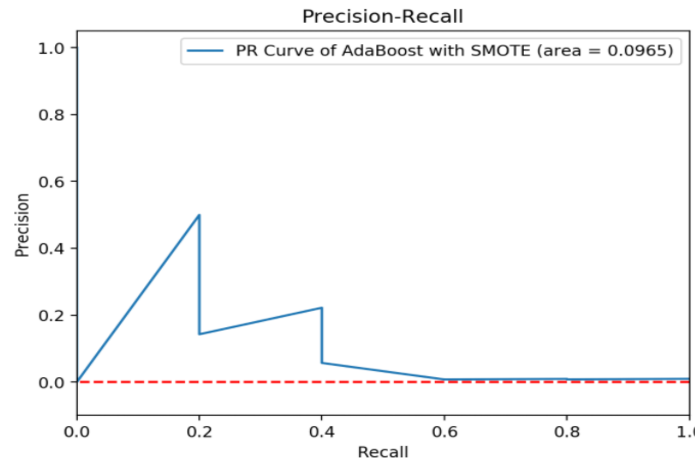
Classification

PR-curve with SMOTE









According to PR-curve, the **Decision Tree** has the best performance.

Based on PR-AUC :
Decision Tree > Adaptive Boosting > RBF SVM

Based on Confusion Matrix :
Adaptive Boosting > RBF SVM > Decision Tree

Classification

ADASYN algorithm

> We also used ADASYN algorithm to handle imbalance of the data. It's an improvement in the SMOTE algorithm.

Code

```
from imblearn.over_sampling import ADASYN

adasyn = ADASYN(random_state=0)
adasyn_X_train, adasyn_y_train = adasyn.fit_resample(X_train, y_train['BINARY'])
adasyn_y_train = adasyn_y_train.astype('int')
adasyn_y_train = adasyn_y_train.values.tolist()
```

Summary

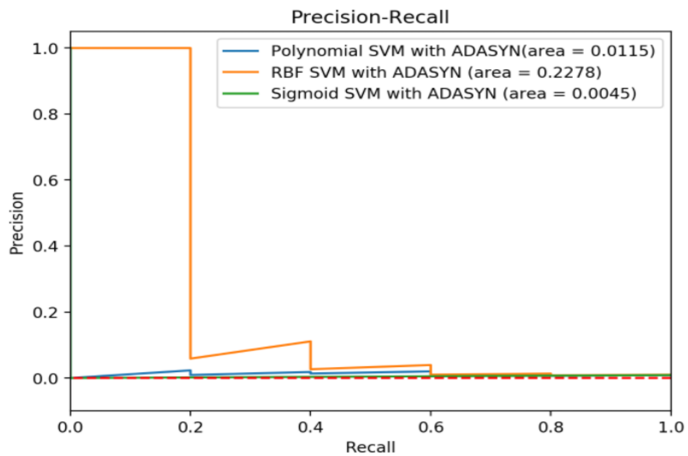
	BINARY
0	5000
1	4989

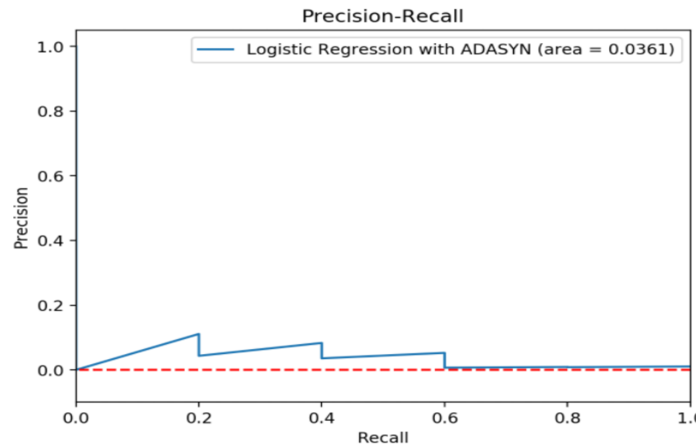
Label 0: Non-Exoplanet-star
Label 1: Exoplanet-star

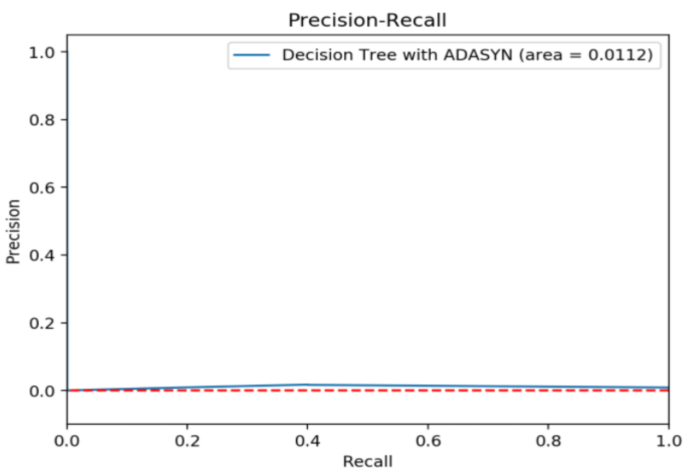
```
Exoplanet stars are 49.94% of total
```

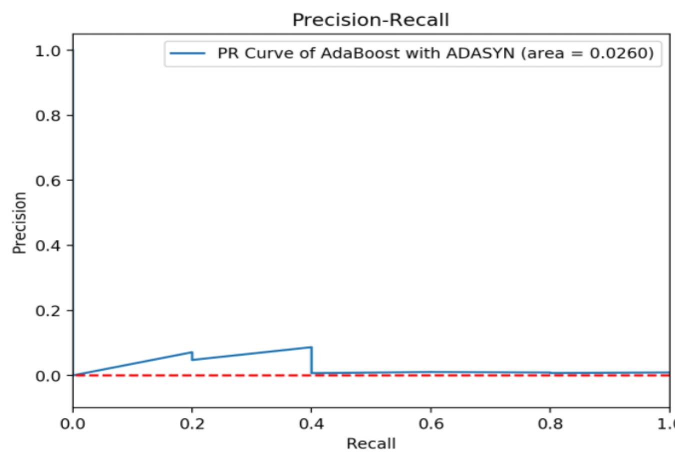
Classification

PR-curve with ADASYN









According to PR-curve, the **RBF SVM** has the best performance.

In conclusion, **SMOTE** showed Better classification result than **ADASYN**.

Based on PR-AUC :
RBF SVM > Logistic Regression > Adaptive Boosting

Based on Confusion Matrix :
Sigmoid SVM > RBF SVM > Decision Tree

Classification

Cost-Sensitive Learning

> We use Cost-Sensitive Learning to handle imbalance of the data. It makes the loss of the majority class greater when building model.

Class Weight

It can be tuned whenever you want. In this web application, I just took simple method. Check the code below.

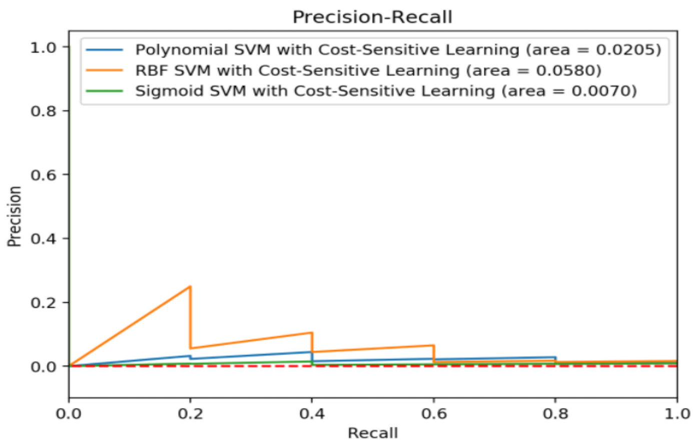
```
Weight for class 0 : 0.20
Weight for class 1 : 27.98
```

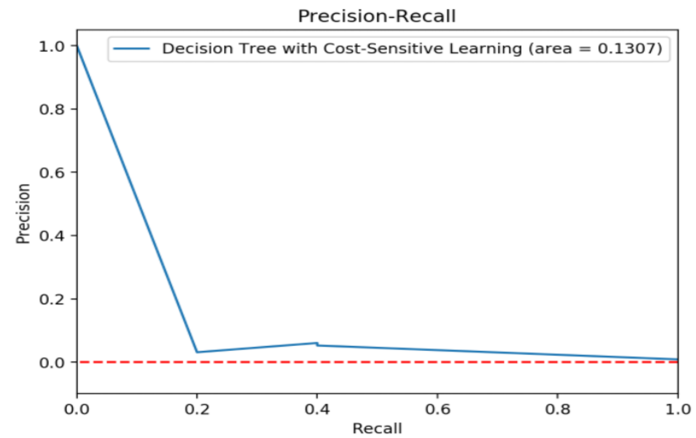
Code

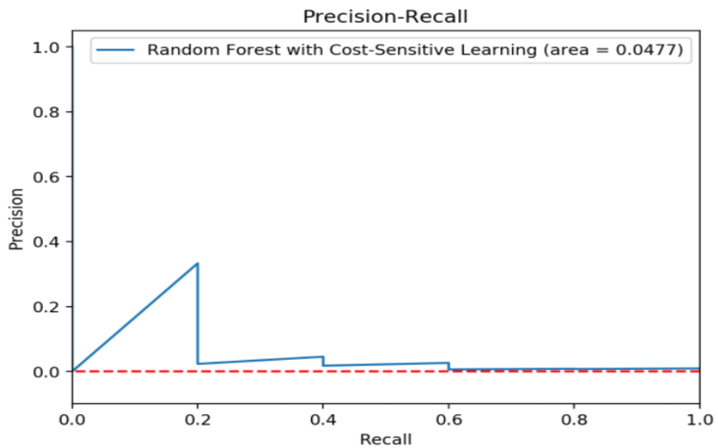
```
neg, pos = np.bincount(y_train['LABEL']-1)
total = neg + pos
weight_0 = (1/neg)*(total)/5.0
weight_1 = (1/pos)*(total)/5.0
class_weight = {0:weight_0, 1:weight_1}
```

Classification

PR-curve with Cost-Sensitive Learning







According to PR-curve, the **Decision Tree** has the best performance.

In conclusion, **SMOTE** showed Better classification result than **Cost-Sensitive Learning**.

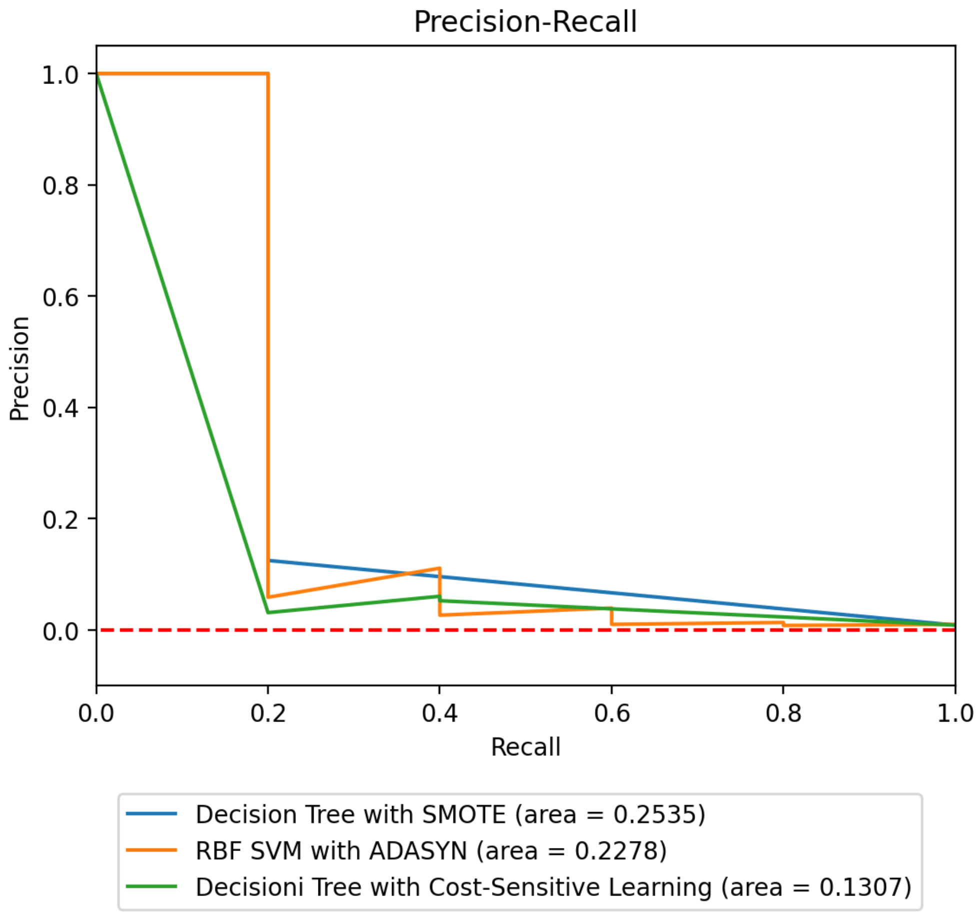
Based on PR-AUC :
Decision Tree > RBF SVM > Random Forest

Based on Confusion Matrix :
Decision Tree > RBF SVM > Random Forest

Index

Conclusion

PR-curve of the 3 Models with the highest PR-AUC



Conclusion

Notable Point

The most ideal PR-AUC is 1, same as AUC. Unfortunately, There was no model with PR-AUC close to 1.

But, We can see Decision Tree has consistently ranked First and Second. This suggests that the performance of the model could be improved if the imbalance in train data could be better treated.

In other words, we can build better models through Hyperparameter tuning when we balance the data.

Through this web application, We hope you could know how to deal with imbalanced data.

Thank you