Module title and code:
Machine Learning- COIY065H7

Name:
Sawsan Shakir

Emails:

sshaki02@dcs.bbk.ac.uk

sunahussam@yahoo.com

Academic Declaration

Introduction

There is a significant interest in studying the classification localization sites of protein into known classes. An expert system developed by Nakai and Kanehisa was the first study for predicting the localization sites of protein. Later, there was number of investigations to improve the prediction accuracy using different classifier systems (Aristoklis D. et al., 2003).

In this report we implemented random forest classifier to investigate the performance of predict the Localization site of protein. Random forest is an ensemble-based schemes which make it suitable for learning from imbalanced data (Chen et al., 2004).

We used three approaches to deal with imbalanced data. The first one, implement the random forest algorithm and predict the 30% of dataset. The second trial was using class weights or cost sensitive learning. That what make random forest more suitable for this problem. (Chen et al., 2004).

The last experiment was resampling the training dataset with oversampling algorithm SMOTE and then training the random forest algorithm from balanced data. Class weighted random forest has shown an improvement better than the oversampling algorithm.

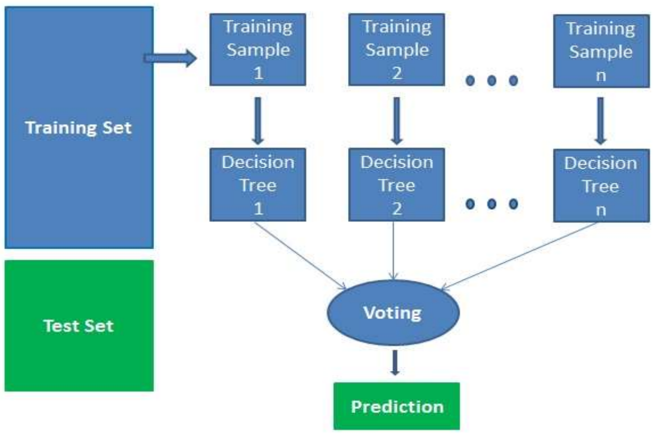
1- Methodology

Most classification problems in the real world are imbalanced dataset. There are two approaches to deal with these cases. One method is weighting the patterns and the other one is sampling either under sampling the majority or oversampling the minority.

In our research we used the weighted approach and SMOTE oversampling to balance the data and avoided using under sampling approach because this approach discards some of the data from the majority which can lead to loss of information from the training instances (Chen et al., 2004).

1.1 Random Forest Algorithm (RF)

It is one kind of ensemble techniques that are based on generating multiple decision trees from subsets of the dataset are randomly drawn with replacement (Liu, Y., 2017) and collecting the prediction from these trees and selecting the best result based on majority voting (Navlani A. 2018).



This chart is taken from (Navlani A. 2018).

The random forest algorithm has two parts (Polamuri S., 2017):

- Generate a number of decision tress

1- In each decision tree model, select randomly "k" number of features from the total "m" (k << m)
2- With the k features, use the best split to split the node and generate children nodes.
3- Iterate the steps 1 to 3 until reaching the required number of nodes.

These steps done for "n" times to create "n" number of decision trees.

- Random forest prediction

1- Using test features and the rules of each tree to predict the targets for test set
2- Select the best predict from the tree's predictions based on the majority voting.
   The majority voting is simply when we have three classifiers results prediction

   classifier 1 -> class 0   classifier 2 -> class 0   classifier 3 -> class 1
   with majority vote, the sample is classified as "class 0."

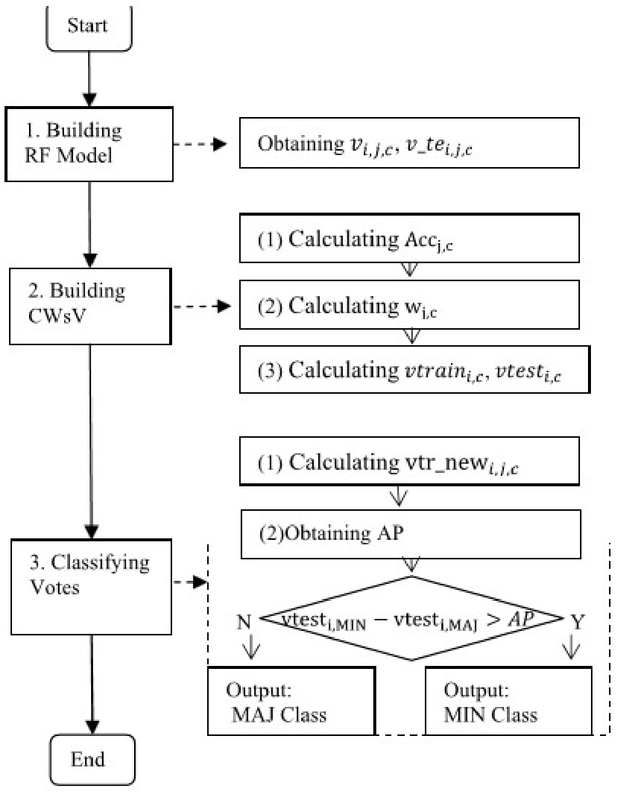
1.2 Class weights random forest (CWsRF)

In random forest different classes have the same weight and tends to be biased towards the majority class that leads to reducing the performance for predicting the minority classes (Zhu M., Xia J., 2018). Therefore, we used class weights method which assign different weights to the patterns with high cost for the minority classes and low cost for majority classes. (Chen et al., 2004)

The flow chart shows the steps of building CWsRF. The algorithm of this method has three main procedures (Zhu M., Xia J., 2018).

- Firstly, building the RF model to get the vote classifiers. v for sample I in classifier j
- Secondly, building the class weights vote which is done in two steps:

The first step is calculating different weights for each class by finding out the scores for each classifier for each sample which be 0 or 1. And then find the accuracy Acc $_{j,c}$ for the class 'c' per classifier 'j'. each classifier has two accuracies for minority and majority. Calculating the weight w $_{j,c}$ for each classifier ' j' per class 'c'.



Start
1. Building RF Model
Obtaining $v_{i,j,c}$, $v\_te_{i,j,c}$
(1) Calculating Acc$_{j,c}$
2. Building CWsV
(2) Calculating w$_{i,c}$
(3) Calculating $vtrain_{i,c}$, $vtest_{i,c}$
(1) Calculating vtr_new$_{i,j,c}$
(2)Obtaining AP
3. Classifying Votes
N
$vtest_{i,MIN} - vtest_{i,MAJ} > AP$
Y
Output: MAJ Class
Output: MIN Class
End

This chart is taken from (Zhu M., Xia J., 2018).

w $_{j,\ minority}$ = Acc $_{j,\ minority}$

w $_{j,}$ = Acc $_{j,\ majority}$

the second step compute the vote train and test for each class

vote-train $_{i,c}$ = $\sum_{j=1}^{H} V\_tr\ i,j,c\ x\ Wj,c$

vote-test $_{i,c}$ = $\sum_{j=1}^{H} V\_te\ i,j,c\ x\ Wj,c$

- Thirdly, to classify the votes for new sample, threshold voting technique is used. The vote-test calculated for the sample in each class and the decision is based on the aggregating probability AP as shown in the chart above. If the difference of vote minority and vote majority greater than AP then the sample voting is classified as minority. Other wise it is classified as majority.

1.3 Oversampling random forest

Another way to tackle the imbalanced data for classification is resampling the dataset approach.

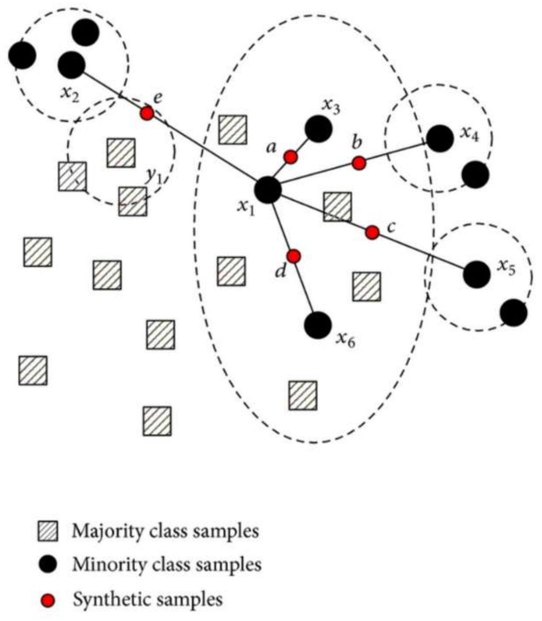
We focused on SMOTE (Synthetic Majority Oversampling Technique) as one of techniques used to generate synthetic examples of existing minority points that sit along the line segments of the different classes.

This is to improve the performance as having these additional points helps in creating large and less specific decision boundaries between the classes which helps the classifiers in improving their generalisation ability. (Santos, 2018)

The algorithm of SMOTE is this; one of the minority data points is chosen randomly for example x1 as show in the figure below and based on k of nearest neighbours which is specified by the user. In the figure four points x2, x3, x4 and x5 are taken and new four points are generated (a, b,c,d) and interpolating every two samples on the line between them (Last F., el at.,2017).

For instance, the new sample a is created according to the formula $a = x1 + \omega (x1 - x3)$

where ω is a random weight between 0 and 1.



The figure is taken from:

https://www.researchgate.net/publication/287601878/figure/fig1/AS:316826589384744@1452548753581/The-schematic-of-NRSBoundary-SMOTE-algorithm.png

2. Tuning the classifier's parameters

In our experiments we used grid search technique to hyperparameter tuning which builds, fits and evaluates the model with all possible combination of the parameter’s values specified in the grid function (Liu, Y., 2017).

We used four main parameters with range of values that could tune to improve the performance of the model recommended (Liu, Y., 2017)

n_estimators: [100,300,500,700,900]

max_features: ["sqrt", "log2", None]

max_depth': [10, 20, None]

min_samples_split: [2,5,10]

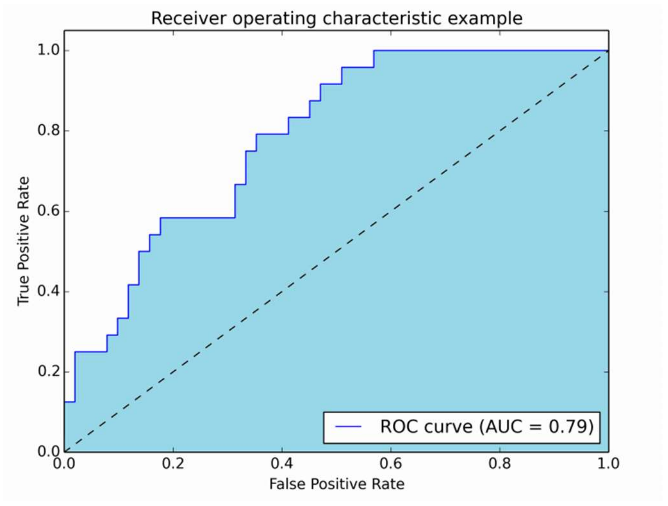
The method takes long computation time as it builds the model many times. In our experiments it generates 135 possible combinations with 5-fold validation.

3. Evaluation methods

To get a better understanding for classification errors, we considered confusion matrix and other evaluation methods to enhance the performance of classification model. For instance:

- Precision represents a measure of the exactness of a classifier. The larger the number of false positives the lower the precision (Brownlee J. 2014)
  Precision = TP / TP+FP
- Recall indicates a measure of a classifier's completeness. The larger the number of false negatives the lower the recall (Brownlee J. 2014)
  Recall = TP / TP+FN
- AUC_ROC:
  AUC stands for area under curve. The curve is the receiver operating characteristic curve (ROC). The curve drawn by plotting the true positive rate = TP / TP+FN with the false positive rate =FP / FP+TN at different probability between [0 -1]. (Liu, Y., 2017)

The area under curve represents the degree of separability and measures the ability of the model to distinguish between the classes.



This figure is taken from https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it

When the AUC is near to 0 this indicates the worse measure of separability for the model. Whereas, the degree of separability is good when the AUC near to 1 (Narkhede S., 2018). The dash line in the figure represent random guessing and the AUC =0.5 (Liu, Y., 2017).

For multi-classes, there is an AUC measure for each class. Each class is compared against the other classes to find AUC for this class. Then the average of the AUC is calculated which is a measure of the separability of the model (Narkhede S., 2018).

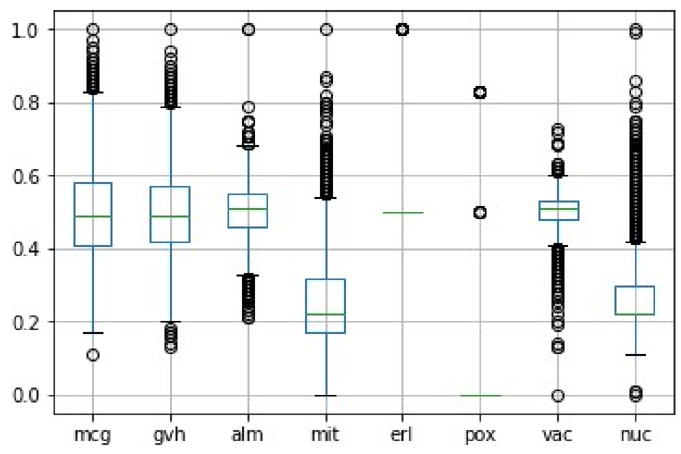
4. Dataset and pre-processing

- The dataset used was Yeast proteins which has 1484 instances and 9 features (8 predictive, 1 target) with no missing values. (Uci, Yeast Data Set)

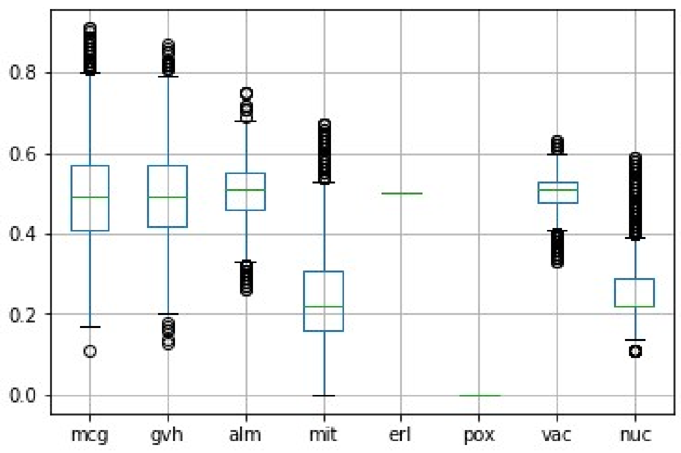
There are 10 classes:

CYT (cytosolic or cytoskeletal)	463
NUC (nuclear)	429
MIT (mitochondrial)	244
ME3 (membrane protein, no N-terminal signal)	163
ME2 (membrane protein, uncleaved signal)	51
ME1 (membrane protein, cleaved signal)	44
EXC (extracellular)	37
VAC (vacuolar)	30
POX (peroxisomal)	20
ERL (endoplasmic reticulum lumen)	5

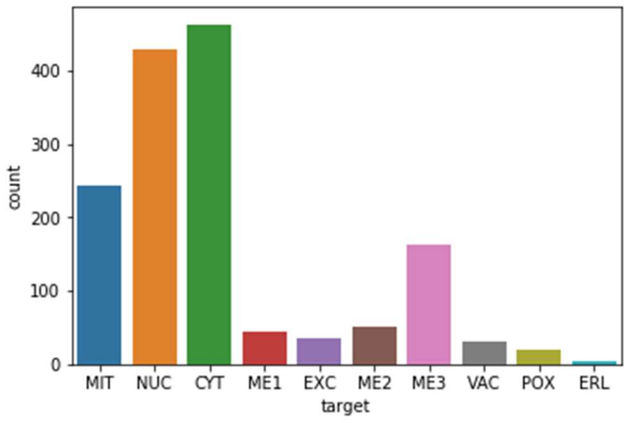
More details available in Appendix A



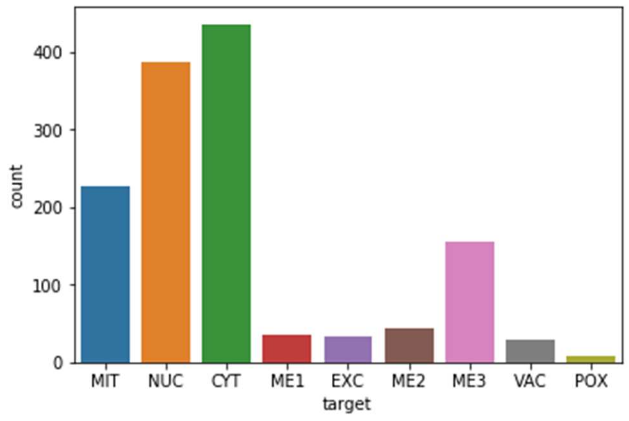
Yeast data box plot with outliers



yeast data box plot after reduce outliers



with outliers



after reduced the outliers

Bar graph data point numbers per class

It is clear from the box plot above that there are many numbers of outliers. We used z-score method to find out the outliers and reduce them. This method assumes the outliers' data are the points far from the mean of Gaussian distribution and normalizes any data point (xi) that has a zi value larger than a threshold z_{th}. (Widmann M. et al., 2018)

Where (zi = xi – mean / sd) and z_{th} = +3, -3

However, when 126 outlier's data point were removed, all data points for ERL class were removed causing a new problem. Hence this approach was abandoned and the experiments were conducted on all of the original dataset.

See appendix B and C-1

5. Experiments and results

We have implemented three random forest experiments using different techniques to tackle the imbalanced data problem, and in all of these experiments the parameters of the random forest classifier function are assigned as baseline:

n_estimators = 200,   oob_score = True,   random_state = 10

5.1 Experiment 1

Random Forest (RF)

We classified yeast Protein Localization by implementing random forest using 70% training set. The random forest was used to train 200 trees as baseline experiment. To tuning the parameters, we used grid search hyper parameter using range of values for the parameters which were

n_estimators: [100,300,500,700,900],   max_features: ["sqrt", "log2", None],

max_depth   : [10, 20, None],   min_samples_split: [2,5,10]

We found the best training accuracy result of 61.17 % after different combinations with cross validation of 5. We specified the cv = 5 as the minority class has 5 data points and to ensure that each class can be in each partition for the training.

The best training accuracy was fund with the following tuning parameter values:

max_depth = 10,   max_features = sqrt,   min_samples_split = 2 ,   n_estimators = 300

Afterwards we fitted the model and predicted the classification using 30% (446 dataset) test set and got test accuracy of 63.90 %. The table below shows the performance and the misclassification in a confusion matrix. See appendix B and C-2

Class label	precision	recall	f1-score	Support (no. of patterns)	CYT	ERL	EXC	ME1	ME2	ME3	MIT	NUC	POX	VAC
CYT	0.57	0.72	0.64	140	101	0	0	0	2	3	6	26	1	1
ERL	1.00	1.00	1.00	1	0	1	0	0	0	0	0	0	0	0
EXC	0.64	0.64	0.64	11	3	0	7	0	1	0	0	0	0	0
ME1	0.88	0.74	0.80	19	0	0	2	14	1	1	0	0	1	0
ME2	0.33	0.33	0.33	12	3	0	1	2	4	0	2	0	0	0
ME3	0.71	0.89	0.79	45	2	0	0	0	0	40	0	3	0	0
MIT	0.73	0.60	0.66	77	19	0	0	0	2	4	46	6	0	0
NUC	0.66	0.54	0.59	128	42	0	0	0	2	6	9	69	0	0
POX	0.60	0.60	0.60	5	2	0	0	0	0	0	0	0	3	0
VAC	0.00	0.00	0.00	8	5	0	1	0	0	2	0	0	0	0
micro avg.	0.64	0.64	0.64	446										
macro avg.	0.61	0.61	0.61	446										
weighted avg.	0.64	0.64	0.63	446										

5.2 Experiment 2

Class weights Random Forest (CWsRF)

Random forest with class weights was conducted on training 70% of the data set. The minority class got high weight while the majority got the low weight using the function class_weight.compute_class_weight.

'CYT': 0.32136223, 'ERL':25.95, 'EXC': 4.325 , 'ME1': 4.152 , 'ME2': 2.66153846, 'ME3': 0.87966102,

'MIT':0.62155689, 'NUC':0.3448505, 'POX': 6.92, 'VAC':4.71818182

We fitted the random forest classifier on the grid search function with the same range of parameters' values in the previous experiment with 5-fold validation. The best training accuracy found was 61.07% by assigning the parameters values as below:

max_features = 'sqrt', max_depth = 20, min_samples_split = 10, n_estimators = 500

Using the model to predict the 30% testing data set. The test accuracy was 62.33% and the Table below illustrates the performance and confusion matrix. See appendix B and C-3

Class label	precision	recall	f1-score	Support (no. of patterns)	CYT	ERL	EXC	ME1	ME2	ME3	MIT	NUC	POX	VAC
CYT	0.60	0.62	0.61	140	87	0	0	0	2	4	13	30	1	3
ERL	0.50	1.00	0.67	1	0	1	0	0	0	0	0	0	0	0
EXC	0.64	0.64	0.64	11	1	0	7	0	1	0	1	0	0	1
ME1	0.81	0.89	0.85	19	0	0	1	17	0	1	0	0	0	0
ME2	0.33	0.33	0.33	12	0	1	1	4	4	0	0	1	0	1
ME3	0.68	0.91	0.78	45	2	0	0	0	0	41	0	2	0	0
MIT	0.65	0.58	0.62	77	14	0	1	0	2	5	45	9	1	0
NUC	0.63	0.57	0.60	128	35	0	0	0	3	7	10	73	0	0
POX	0.60	0.60	0.60	5	2	0	0	0	0	0	0	0	3	0
VAC	0.00	0.00	0.00	8	5	0	1	0	0	2	0	0	0	0
micro avg.	0.62	0.62	0.62	446										
macro avg.	0.54	0.62	0.57	446										
weighted avg.	0.62	0.62	0.62	446										

5.3 Experiment 3

Oversampling Random Forest (OSRF)

We implemented the same previous experiment, but instead of the weighted classes we used one of the resampling algorithms to solve the imbalanced dataset.

We split the data to 70% training set and applied the SMOTE algorithm to get (323) data points for each class. Following that, we trained the random forest algorithm using grid search with the same parameters' values range as was done in the previous experiments and a 5-fold validation. We got the best performance with this combination of parameters

max_features= 'sqrt', max_depth = 20, min_samples_split= 10, n_estimators= 900

And training accuracy of 87.08% which was higher than the accuracy of previous experiments. When applied the 30% testing set on the model the testing accuracy was 60.76%. Below are the performance results and confusion matrix for the testing data set. Appendix B and C-4

Class label	precision	recall	f1-score	Support (no. of patterns)	CYT C	ERL	EXC	ME1	ME2	ME3	MIT	NUC	POX	VA
CYT	0.60	0.58	0.59	140	81	0	0	0	3	3	13	33	2	5
ERL	1.00	1.00	1.00	1	0	1	0	0	0	0	0	0	0	0
EXC	0.64	0.64	0.64	11	1	0	7	0	1	0	1	0	0	1
ME1	0.83	0.79	0.81	19	0	0	0	15	1	1	0	0	1	1
ME2	0.31	0.42	0.36	12	2	0	0	3	5	1	0	0	0	1
ME3	0.67	0.87	0.76	45	0	0	1	0	1	39	0	3	1	0
MIT	0.67	0.61	0.64	77	11	0	1	0	2	5	47	9	1	1
NUC	0.61	0.57	0.59	128	35	0	1	0	3	7	9	73	0	0
POX	0.38	0.60	0.46	5	1	0	0	0	0	0	0	1	3	0
VAC	0.00	0.00	0.00	8	5	0	1	0	0	2	0	0	0	0
micro avg.	0.61	0.61	0.61	446										
macro avg.	0.57	0.61	0.58	446										
weighted avg.	0.61	0.61	0.61	446										

6. Results:

Table 1 below shows the training accuracy, test accuracy and auc_roc for the three algorithms. Overall, what stands out from the table is that the best performance was the RF experiment with test accuracy of 63.9% and training accuracy of 61.1%.

In details, it can be seen that the CWsRF method had no clear improvement over the RF with 62.3% and 61% test and train accuracies respectively. Another interesting point is that the oversampling method achieved improvement in the training accuracy by 87%. However, the test result was significantly lower than the training result of 60.7%. This can be explained by the model being overfitted with the generated samples to balance the data.

We can notice that the areas under curve were similar for the three algorithms. The class weights random forest experiment seems to be slightly better than the others with 78.2%.

Algorithms	Training accuracy%	Test accuracy%	Auc_roc%
RF	61.175	63.901	77.84
CWsRF	61.078	62.331	78.29
OSRF	87.089	60.762	77.80

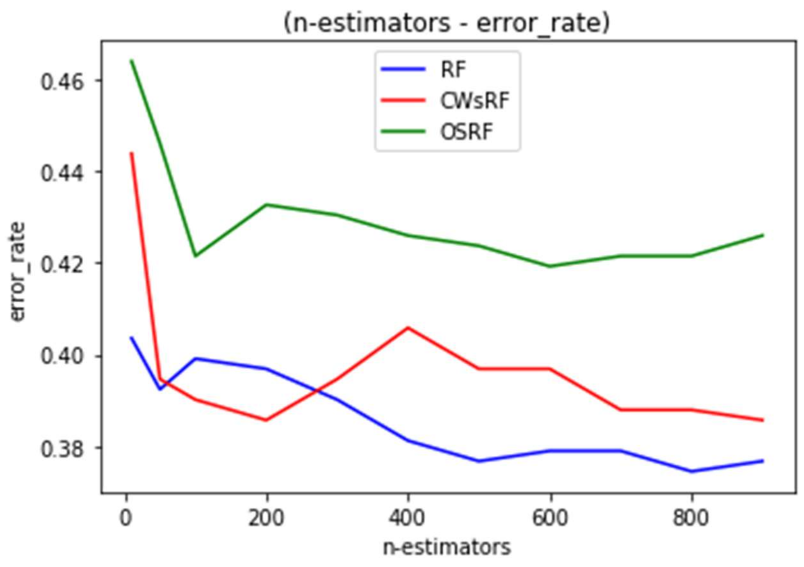
Table 1

It is important when the data was imbalanced to focus on how correct the model predicts the minority classes (Chen et al., 2004). The table 2 below presents the accuracy result for each class using 30% of the dataset (446 data points). It can be seen that the VAC class with 8 data points for testing was misclassified in all three methods. By contrast, the ERL has 1 data point which was predicted correctly in all algorithms.

patterns	Class	RF %	CWsRF %	OSRF %
140	CYT	72.14	62.14	57.85
1	ERL	100	100	100
11	EXC	63.63	63.63	63.63
19	ME1	73.68	89.47	78.94
12	ME2	33.33	33.33	41.66
45	ME3	88.88	91.11	86.66
77	MIT	59.74	58.44	61.03
128	NUC	53.90	57.03	57.03
5	POX	60	60	60
8	VAC	0	0	0
Mean		60.53	61.51	60.68

Table 2

With respect to POX class, we found the same accuracy in all algorithms with 60%. It is important to highlight that the class weights random forest algorithms made significant improvement in model performance compared with oversampling random forest classifier with 61.5% and 60.6% respectively as methods to deal with imbalanced data set.



The line graph above compares the test error rates for the three experiments RF, CWsRF and OSRF in range of number of trees between 10 to 900 and the other parameters were set to default.

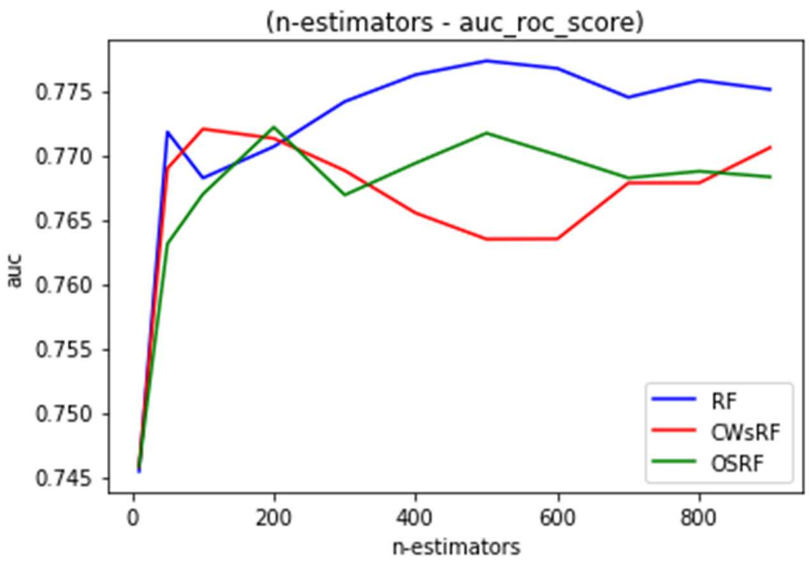
Overall, we can see that the greater the number of trees generated to fit the model the lower the error rate that model encounters.

What stands out from the graph is that the RF algorithm had the best performance when the error tends to be minimized to nearly 33% at number of trees around 500. Whereas CWsRF predicted the test data with 39% error rate when there were 200 trees. Regarding the OSRF when number of trees around 600, the error rate decreased to approximately 42% which was lowest performance compared with the other algorithms.

Comparing the algorithms with different number of trees by the values of area under curve ROC. It is clear that the more trees contribute to building the model the higher auc_roc got.

Another interesting point is that at number of trees of 500, the RF experiment and OSRF predicted the testing data with auc_roc to around 77.6% and 77% respectively. However, CWsRF reached the highest auc_roc value at 77.0% when the number of trees was 900.

The line graph results is based on 30% of the dataset after fitting the three algorithms with range of n-estimators between 10 and 900 with default values for the other parameters.



Conclusion

In this research we have explored the use of random forest classifier to classify the yeast dataset.

We first used random forest algorithm then implemented class weights to reduce the effect of imbalanced data and the third trial, SMOTE oversampling method was implemented to generate samples of minority classes data points. Generally, the best result we achieved with the random forest experiment and class weights random forest algorithm performed considerably better than oversampling random forest technique.

Given more time we would train and build the model using k-fold cross validation while using oversampling algorithm. We expect to avoid the model overfitting by generating new samples for each training partition.

Bibliography

- Aggiwal R. 2017, Introduction to Random forest, available online at https://dimensionless.in/introduction-to-random-forest/. Last accessed 22/04/2019.

- Aristoklis D. et al., 2003. Classification of Protein Localisation Patterns via Supervised Neural Network Learning, Department of Information Systems and Computing, Brunel University.

- Brownlee J. 2014, Classification Accuracy is Not Enough: More Performance Measures You Can Use, available online https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/. Last accessed 17/04/2019

- Chen C., Liaw A., Breiman L., 2004. Using random forest to learn imbalanced data. UC Berkeley.

- Fawcett T., 2016, Learning from Imbalanced Classes, available online at https://www.svds.com/learning-imbalanced-classes/. Last accessed 20/04/2019

- Last F., el at.,2017, Oversampling for Imbalanced Learning Based on K-Means and SMOTE. NOVA Information Management School.

- Liu, Y., 2017. Python machine learning by example. Birmingham-Mumbai: Packt.

- Navlani A., 2018, Understanding Random Forests Classifiers in Python, available online at https://www.datacamp.com/community/tutorials/random-forests-classifier-python. Last accessed 20/04/2019.

- Narkhede S., 2018, Understanding AUC - ROC Curve, available online https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. Last accessed 17/04/2019.

- Polamuri S., 2017, How the random forest algorithm works in machine learning, available online at http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/. Last accessed 19/04/2019

- Santos M. Seoane, 2018. Cross-validation for imbalanced datasets: avoiding overoptimistic and overfitting approaches. IEEE Computational Intelligence Magazine, 10.1109.

- Uci, Yeast Data Set, available online at https://archive.ics.uci.edu/ml/datasets/Yeast. Last accessed 20/04/2019

- Widmann M. et al., 2018, Four Techniques for Outlier Detection, available online https://www.kdnuggets.com/2018/12/four-techniques-outlier-detection.html. Last accessed 15/04/2019.

- Zhu M., Xia J., 2018. Class Weights Random Forest Algorithm for Processing Class Imbalanced Medical Data. IEEE access

Appendix

Appendix A

Data File

Filename: SHAKIR - data.zip

this zip file contains file: yeast_data.txt

Description:

This file contains dataset for Protein Localization Sites. It is created by Kenta Nakai in institute of Molecular and Cellular Biology Osaka University. the Number of Instances: 1484 for the Yeast dataset and the Number of Attributes for Yeast dataset: 9 (8 predictive, 1 name). there are no missing values in the dataset. This dataset available in website: http://archive.ics.uci.edu/ml/datasets/Yeast.

Attribute Information.

1. Sequence Name : Accession number for the SWISS-PROT database
2. mcg: McGeoch's method for signal sequence recognition.
3. gvh: von Heijne's method for signal sequence recognition.
4. alm: Score of the ALOM membrane spanning region prediction program.
5. mit: Score of discriminant analysis of the amino acid content of

   the N-terminal region (20 residues long) of mitochondrial and

   non-mitochondrial proteins.
6. erl: Presence of "HDEL" substring (thought to act as a signal for

   retention in the endoplasmic reticulum lumen). Binary attribute.
7. pox: Peroxisomal targeting signal in the C-terminus.
8. vac: Score of discriminant analysis of the amino acid content of

   vacuolar and extracellular proteins.
9. nuc: Score of discriminant analysis of nuclear localization signals

   of nuclear and non-nuclear proteins.

Class Distribution. The class is the localization site.

CYT (cytosolic or cytoskeletal)	463
NUC (nuclear)	429
MIT (mitochondrial)	244
ME3 (membrane protein, no N-terminal signal)	163
ME2 (membrane protein, uncleaved signal)	51
ME1 (membrane protein, cleaved signal)	44
EXC (extracellular)	37
VAC (vacuolar)	30
POX (peroxisomal)	20
ERL (endoplasmic reticulum lumen)	5

Appendix B

Result file

Filename: SHAKIR - results.zip

this zip file contains 5 files

File name	Type	Description
SHAKIR - preproccesing.pdf	pdf	The file contains statistical analysis table, box graphs and bar graphs before and after removed the outliers. Ai addition to matrix pairwise graphs.
SHAKIR - RF experiment 1.pdf	pdf	This file contains the result of implement random forest algorithm which are the best training accuracy with best parameters combination and then displays the performance, confusion matrix of testing result, the accuracy for each class and the average auc_roc value
SHAKIR - CWsRF experiment 2.pdf	pdf	This file contains the result of implement class weights random forest algorithm using the result of weights for each class. Then got the best training accuracy with best parameters combination and then displays the performance and confusion matrix of testing result, the accuracy for each class and the average auc_roc value
SHAKIR - OSRF experiment 3.pdf	pdf	This file contains the result of the list of classes after resampling the data and the build the model to get the best training accuracy with best parameters combination and then displays the performance and confusion matrix of testing result, the accuracy for each class and the average auc_roc value
SHAKIR - graphes.pdf	pdf	The file contains the result of two line graphs to compare the algorithms by error rate and auc_roc with range of number of trees.

Appendix C

Code file

Filename: SHAKIR - experiments python code.zip

this zip file contains 5 files:

Appendix C-1.

Filename: SHAKIR - preproccesing.py

pre-processing code

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt



yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

print(yeast_data.shape)

yeast_data.head(10)

yeast_data.describe()

#histograms
yeast_data.hist()
plt.show()

## dataset before outliers removed
yeast_data.boxplot()


### find out the outliers and remove it using z-score method

yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

from scipy import stats
import numpy as np
z = np.abs(stats.zscore(yeast_data.iloc[:,:8]))
print(z)

threshold = 3

#print(np.where(z > 3))
#z[5][5]
yeast_outlier_removed = yeast_data.iloc[:,:8][(z < 3).all(axis=1)]

yeast_data.shape


## ## dataset after outliers removed
yeast_outlier_removed.boxplot()
yeast_outlier_removed.shape
#export_excel = yeast_outlier_removed.to_excel (r'C:\Users\Soona\export_dataframe.x
lsx', index = None, header=True)





## before remove outliers
print(yeast_data.groupby('target').size())
sns.countplot(yeast_data.target)
plt.show()


##after remove outliers

yeast_data_no_outliers = pd.read_csv('yeast_no_outliers.csv', names= ['mcg','gvh','
alm','mit','erl', 'pox','vac','nuc','target'])
print(yeast_data_no_outliers.groupby('target').size())
sns.countplot(yeast_data_no_outliers.target)
plt.show()


## Seaborn, seaborn.pairplot,available online https://seaborn.pydata.org/generated/
seaborn.pairplot.html,Last accessed 12/04/2019
sns.pairplot(yeast_data, hue="target")
plt.show()
```

Appendix C-2

Filename: SHAKIR _ RF experiment 1.py

Experiment 1 code

```
## Experiment 1 Random Forest (RF)

## the code references

## Sullivan W.,2017. Python machine learning illustrated guide for beginners. Healt
hy pragmatic solutions Inc.
##
## Liu Y.(Hayden), 2017. Python machine learning by example. Birmingham-Mumbai:
Packt.
##
##Stackoverflow, Scikit-learn, get accuracy scores for each class,
##available online https://stackoverflow.com/questions/39770376/scikit-learn-get-ac
curacy-scores-for-each-class.
##Last accessed 22/04/2019
##
##Medium, AUC ROC Curve Scoring Function for Multi-class Classification,
##available online https://medium.com/@plog397/auc-roc-curve-scoring-function-for-m
ulti-class-classification-9822871a6659.
##Last accessed 22/04/2019
##



## This program first implements Random Forest with imbalanced data using grid sear
ch to tuning the parameters and predict the accuracy
## split the data to 70% training set and 30% test set

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from imblearn.over_sampling import RandomOverSampler
from sklearn.ensemble import RandomForestClassifier


## split the data to training and testing datasets (Sullivan,2017)

yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

features = yeast_data.iloc[:,0:8].values
labels = yeast_data.iloc[:,8].values
train_features, test_features, train_labels, test_labels = train_test_split(feature
s, labels, test_size = 0.3, random_state= 0)

###################################################################################
########

rf_clf = RandomForestClassifier(n_estimators=200, oob_score=True ,random_state=10)

## Hyper parameter using grid search(Liu, 2017)

parameters = {'n_estimators' : [100,300,500,700,900],
              'max_features' : ["sqrt", "log2", None],
              'max_depth'    : [10, 20, None],
              'min_samples_split': [2,5,10] }

from sklearn.model_selection import GridSearchCV

gd_sr = GridSearchCV(estimator= rf_clf,
                     param_grid=parameters,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1)


gd_sr.fit( train_features, train_labels)
best_parameters = gd_sr.best_params_
print(best_parameters)


best_result = gd_sr.best_score_
print(best_result)

###########################################################################

## predict the test set using best result

predictions = gd_sr.best_estimator_.predict(test_features)

#comparison = pd.DataFrame({'Real':test_labels, 'Predictions': predictions})
#print(comparison)


## find accuracy and confusion matrix (Sullivan,2017)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(confusion_matrix(test_labels, predictions))
print(classification_report(test_labels, predictions))
print("accuracy_score" , accuracy_score(test_labels, predictions))
print( )


### find the accuracy of each class (Stackoverflow)

cm = confusion_matrix(test_labels, predictions)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print ("accuracy of each class")
print()
print(cm.diagonal())


### calculate the avarage auc_roc for the classes(Medium)

from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_roc_auc_score(truth, pred, average="macro"):

    lb = LabelBinarizer()
    lb.fit(truth)

    truth = lb.transform(truth)
    pred = lb.transform(pred)

    return roc_auc_score(truth, pred, average=average)

print("the avarage Area under curve ROC is:")
multiclass_roc_auc_score(test_labels, predictions)
```

Appendix C-3

Filename: SHAKIR - CWsRF experiment 2.py

Experiment 2 code

```
### Experiment 2 Class Weights random forest (CWsRF)


## the code references

## Sullivan W.,2017. Python machine learning illustrated guide for beginners. Healt
hy pragmatic solutions Inc.
##
## Liu Y.(Hayden), 2017. Python machine learning by example. Birmingham-Mumbai: Pac
kt.
##
## Scikit-learn, sklearn.utils.class_weight, available online
##https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.comp
ute_class_weight.html. Last accessed 22/04/2019
##
##Stackoverflow, Scikit-learn, get accuracy scores for each class,
##available online https://stackoverflow.com/questions/39770376/scikit-learn-get-ac
curacy-scores-for-each-class.
##Last accessed 22/04/2019
##
##Medium, AUC ROC Curve Scoring Function for Multi-class Classification,
##available online https://medium.com/@plog397/auc-roc-curve-scoring-function-for-m
ulti-class-classification-9822871a6659.
##Last accessed 22/04/2019
##


## This program implement Random Forest with class weights  and split the data to 7
0% for training and 30% testing


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from imblearn.over_sampling import RandomOverSampler


yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

print(yeast_data.groupby('target').size())

## split the data to training and testing datasets (Sullivan,2017)
features = yeast_data.iloc[:,0:8].values
labels = yeast_data.iloc[:,8].values
train_features, test_features, train_labels, test_labels = train_test_split(feature
s, labels, test_size = 0.3, random_state= 0)


## weights of the classes(Scikit-learn)
class_weight =class_weight.compute_class_weight('balanced', np.unique(train_labels)
,train_labels)
```

```python
print(class_weight)

class_weight = dict({'CYT':0.32136223, 'ERL':25.95    ,'EXC': 4.325 ,'ME1': 4.152
,'ME2': 2.66153846  ,'ME3': 0.87966102,
 'MIT':0.62155689  , 'NUC':0.3448505, 'POX': 6.92   , 'VAC':4.71818182})


rf_clf1 = RandomForestClassifier(n_estimators=200, oob_score=True, random_state=10,
class_weight = class_weight)

## Hyper parameters using grid search(Liu Y., 2017)
parameters = {'n_estimators' : [100,300,500,700,900],
              'max_features' : ["sqrt", "log2", None],
              'max_depth'    : [10, 20, None],
              'min_samples_split': [10,30,50]}


from sklearn.model_selection import GridSearchCV

gd_sr = GridSearchCV(estimator= rf_clf1,
                     param_grid=parameters,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1)


gd_sr.fit( train_features, train_labels)
best_parameters = gd_sr.best_params_
print(best_parameters)


best_result = gd_sr.best_score_
print(best_result)

## predict the test set using best result
predictions = gd_sr.best_estimator_.predict(test_features)

#comparison = pd.DataFrame({'Real':test_labels, 'Predictions': predictions})
#print(comparison)

## find accuracy and confusion matrix (Sullivan,2017)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(confusion_matrix(test_labels, predictions))
print(classification_report(test_labels, predictions))
print("accuracy_score" , accuracy_score(test_labels, predictions))
print( )

### find the accuracy of each class (Stackoverflow)
cm = confusion_matrix(test_labels, predictions)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print ("accuracy of each class")
print()
print(cm.diagonal())

### calculate the avarage auc_roc for the classes(Medium)
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelBinarizer
def multiclass_roc_auc_score(truth, pred, average="macro"):

    lb = LabelBinarizer()
    lb.fit(truth)

    truth = lb.transform(truth)
    pred = lb.transform(pred)

    return roc_auc_score(truth, pred, average=average)
print("Area under curve ROC is:")
multiclass_roc_auc_score(test_labels, predictions)
```

Appendix C- 4

Filename: SHAKIR - OSRF experiment 3.py

Experiment 3 code

```
### Experiment 3 Over Sampling random forest (OSRF)


## the code references

## Sullivan W.,2017. Python machine learning illustrated guide for beginners. Healt
hy pragmatic solutions Inc.
##
## Liu Y.(Hayden), 2017. Python machine learning by example. Birmingham-Mumbai: Pac
kt.
##
##Imbalanced-learn, imblearn.over_sampling.SMOTE ,available online
##https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_samplin
g.SMOTE.html, Last accessed 22/04/2019
##
##Stackoverflow, Scikit-learn, get accuracy scores for each class,
##available online https://stackoverflow.com/questions/39770376/scikit-learn-get-ac
curacy-scores-for-each-class.
##Last accessed 22/04/2019
##
##Medium, AUC ROC Curve Scoring Function for Multi-class Classification,
##available online https://medium.com/@plog397/auc-roc-curve-scoring-function-for-m
ulti-class-classification-9822871a6659.
##Last accessed 22/04/2019
##



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold # import KFold
from sklearn.ensemble import RandomForestClassifier
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE


yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

## split the data to training and testing datasets (Sullivan,2017)
print(yeast_data.groupby('target').size())
features = yeast_data.iloc[:,0:8].values
labels = yeast_data.iloc[:,8].values
train_features, test_features, train_labels, test_labels = train_test_split(feature
s, labels, test_size = 0.3, random_state= 0)

#########################################################################

## oversampling by SMOTE algorithm to balance the training data set(Imbalanced-lear
n)
```

```
from collections import Counter
from imblearn.over_sampling import SMOTE

smote= SMOTE(sampling_strategy = 'not majority', random_state= 2, k_neighbors = 2)
new_train_features, new_train_labels = smote.fit_sample(train_features, train_label
s)
print ("the balance classes")
print()
print(sorted(Counter(new_train_labels).items()))

####################################################################################

## fit the balance training data to random forest and tuning the parameters by grid
search(Liu Y., 2017)

rf_clf = RandomForestClassifier(n_estimators=200, oob_score=True, random_state=10)


parameters = {'n_estimators' : [100,300,500,700,900],
              'max_features' : ["sqrt", "log2", None],
              'max_depth'    : [10, 20, None],
              'min_samples_split': [10,30,50] }

from sklearn.model_selection import GridSearchCV

gd_sr = GridSearchCV(estimator= rf_clf,
                     param_grid=parameters,
                     scoring='accuracy',
                     cv=5,
                     n_jobs= -1)

gd_sr.fit( new_train_features, new_train_labels)
best_parameters = gd_sr.best_params_
print(best_parameters)

best_result = gd_sr.best_score_
print (" the best result is")
print(best_result)

####################################################################################
#############

## predict the imbalanced test set using best result from grid search
predictions = gd_sr.best_estimator_.predict(test_features)


## find accuracy and confusion matrix (Sullivan,2017)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(confusion_matrix(test_labels, predictions))
print(classification_report(test_labels, predictions))
print(accuracy_score(test_labels, predictions))


### find the accuracy of each class (Stackoverflow)
cm = confusion_matrix(test_labels, predictions)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print ("accuracy of each class")
print()
print(cm.diagonal())


### calculate the avarage auc_roc for the classes(Medium)
from sklearn.metrics import roc_auc_score
```

```python
from sklearn.preprocessing import LabelBinarizer
def multiclass_roc_auc_score(truth, pred, average="macro"):

    lb = LabelBinarizer()
    lb.fit(truth)

    truth = lb.transform(truth)
    pred = lb.transform(pred)

    return roc_auc_score(truth, pred, average=average)

print("Area under curve ROC is:")
multiclass_roc_auc_score(test_labels, predictions)
```

Appendix C-5

Filename: SHAKIR - graphes.py

Graphs code

```
## program to find lists of error_rate and auc_roc with range of n_estimator for RF
experiment

## the code references

## Sullivan W.,2017. Python machine learning illustrated guide for beginners. Healt
hy pragmatic solutions Inc.
##
## Liu Y.(Hayden), 2017. Python machine learning by example. Birmingham-Mumbai: Pac
kt.
##
##Stackoverflow, Scikit-learn, get accuracy scores for each class,
##available online https://stackoverflow.com/questions/39770376/scikit-learn-get-ac
curacy-scores-for-each-class.
##Last accessed 22/04/2019
##
##Medium, AUC ROC Curve Scoring Function for Multi-class Classification,
##available online https://medium.com/@plog397/auc-roc-curve-scoring-function-for-m
ulti-class-classification-9822871a6659.
##Last accessed 22/04/2019
##

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold # import KFold
from sklearn.ensemble import RandomForestClassifier
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from imblearn.over_sampling import RandomOverSampler


yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

 split the data to training and testing datasets (Sullivan,2017)
#print(yeast_data.groupby('target').size())
features = yeast_data.iloc[:,0:8].values
labels = yeast_data.iloc[:,8].values
train_features, test_features, train_labels, test_labels = train_test_split(feature
s, labels, test_size = 0.3, random_state= 0)

rocauc_score=[]
error_rate =[]

n_estimators = [10,50,100,200,300,400,500,600,700,800,900]
#max_features = ["sqrt", "log2", None]

for n_estimator in n_estimators:


    rf_clf1 = RandomForestClassifier(n_estimators = n_estimator, oob_score=True ,ra
ndom_state=10)

    rf_clf1.fit( train_features, train_labels)
    predictions = rf_clf1.predict(test_features)

## find accuracy and confusion matrix (Sullivan,2017
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_s
core

    #print(confusion_matrix(test_labels, predictions))
    #print(classification_report(test_labels, predictions))
    error= 1-accuracy_score(test_labels, predictions)
    error_rate.append(error)

### calculate the avarage auc_roc for the classes(Medium)
    from sklearn.metrics import roc_auc_score
    from sklearn.preprocessing import LabelBinarizer
    def multiclass_roc_auc_score(truth, pred, average="macro"):

        lb = LabelBinarizer()
        lb.fit(truth)

        truth = lb.transform(truth)
        pred = lb.transform(pred)

        return roc_auc_score(truth, pred, average=average)


    auc = multiclass_roc_auc_score(test_labels, predictions)
    rocauc_score.append(auc)
print(error_rate)
print(rocauc_score)




## program to find lists of error_rate and auc_roc with range of n_estimator for CW
sRF experiment

## the code references

## Sullivan W.,2017. Python machine learning illustrated guide for beginners. Healt
hy pragmatic solutions Inc.
##
## Liu Y.(Hayden), 2017. Python machine learning by example. Birmingham-Mumbai: Pac
kt.
##
## Scikit-learn, sklearn.utils.class_weight, available online
##https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.comp
ute_class_weight.html. Last accessed 22/04/2019
##
##Stackoverflow, Scikit-learn, get accuracy scores for each class,
##available online https://stackoverflow.com/questions/39770376/scikit-learn-get-ac
curacy-scores-for-each-class.
##Last accessed 22/04/2019
##
##Medium, AUC ROC Curve Scoring Function for Multi-class Classification,
##available online https://medium.com/@plog397/auc-roc-curve-scoring-function-for-m
ulti-class-classification-9822871a6659.
##Last accessed 22/04/2019
##
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold # import KFold
from sklearn.ensemble import RandomForestClassifier
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from imblearn.over_sampling import RandomOverSampler


yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

## split the data to training and testing datasets (Sullivan,2017)
features = yeast_data.iloc[:,0:8].values
labels = yeast_data.iloc[:,8].values
train_features, test_features, train_labels, test_labels = train_test_split(feature
s, labels, test_size = 0.3, random_state= 0)

## weights of the classes(Scikit-learn)
class_weight =class_weight.compute_class_weight('balanced', np.unique(train_labels)
,train_labels)

class_weight = dict({'CYT':0.32136223, 'ERL':25.95    ,'EXC': 4.325 ,'ME1': 4.152
,'ME2': 2.66153846  ,'ME3': 0.87966102,
 'MIT':0.62155689  , 'NUC':0.3448505, 'POX': 6.92   , 'VAC':4.71818182})

rocauc_score=[]
error_rate = []
n_estimators = [10,50,100,200,300,400,500,600,700,800,900]
#max_features = ["sqrt", "log2", None]

for n_estimator in n_estimators:


    rf_clf1 = RandomForestClassifier(n_estimators = n_estimator, class_weight = cla
ss_weight, oob_score=True ,random_state=10)

    rf_clf1.fit( train_features, train_labels)


    predictions = rf_clf1.predict(test_features)

## find accuracy and confusion matrix (Sullivan,2017)
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_s
core

    #print(confusion_matrix(test_labels, predictions))
    #print(classification_report(test_labels, predictions))
    error= 1-accuracy_score(test_labels, predictions)
    error_rate.append(error)

### calculate the avarage auc_roc for the classes(Medium)
    from sklearn.metrics import roc_auc_score
    from sklearn.preprocessing import LabelBinarizer
    def multiclass_roc_auc_score(truth, pred, average="macro"):

        lb = LabelBinarizer()
        lb.fit(truth)

        truth = lb.transform(truth)
        pred = lb.transform(pred)

        return roc_auc_score(truth, pred, average=average)


    auc = multiclass_roc_auc_score(test_labels, predictions)
    rocauc_score.append(auc)
```

```
print(error_rate)
print(rocauc_score)



## program to find lists of error_rate and auc_roc with range of n_estimator for OS
RF experiment


## the code references

## Sullivan W.,2017. Python machine learning illustrated guide for beginners. Healt
hy pragmatic solutions Inc.
##
## Liu Y.(Hayden), 2017. Python machine learning by example. Birmingham-Mumbai: Pac
kt.
##
##Imbalanced-learn, imblearn.over_sampling.SMOTE ,available online
##https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_samplin
g.SMOTE.html, Last accessed 22/04/2019
##
##Stackoverflow, Scikit-learn, get accuracy scores for each class,
##available online https://stackoverflow.com/questions/39770376/scikit-learn-get-ac
curacy-scores-for-each-class.
##Last accessed 22/04/2019
##
##Medium, AUC ROC Curve Scoring Function for Multi-class Classification,
##available online https://medium.com/@plog397/auc-roc-curve-scoring-function-for-m
ulti-class-classification-9822871a6659.
##Last accessed 22/04/2019
##

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold # import KFold
from sklearn.ensemble import RandomForestClassifier
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from imblearn.over_sampling import RandomOverSampler


yeast_data = pd.read_csv('yeast_data.txt', names= ['mcg','gvh','alm','mit','erl', '
pox','vac','nuc','target'])

## split the data to training and testing datasets (Sullivan,2017)
#print(yeast_data.groupby('target').size())
features = yeast_data.iloc[:,0:8].values
labels = yeast_data.iloc[:,8].values
train_features, test_features, train_labels, test_labels = train_test_split(feature
s, labels, test_size = 0.3, random_state= 0)

## oversampling by SMOTE algorithm to balance the training data set(Imbalanced-lear
n)
from collections import Counter
from imblearn.over_sampling import SMOTE

smote= SMOTE(sampling_strategy = 'not majority', random_state= 2, k_neighbors = 2)
new_train_features, new_train_labels = smote.fit_sample(train_features, train_label
s)


n_estimators = [10,50,100,200,300,400,500,600,700,800,900]
```

```
rocauc_score=[]
error_rate = []
for n_estimator in n_estimators:


    rf_clf1 = RandomForestClassifier(n_estimators = n_estimator, oob_score=True ,ra
ndom_state=10)

    rf_clf1.fit( new_train_features, new_train_labels)


    predictions = rf_clf1.predict(test_features)

## find accuracy and confusion matrix (Sullivan,2017)
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_s
core

    #print(confusion_matrix(test_labels, predictions))
    #print(classification_report(test_labels, predictions))
    error= 1-accuracy_score(test_labels, predictions)
    error_rate.append(error)



### calculate the avarage auc_roc for the classes(Medium)
    from sklearn.metrics import roc_auc_score
    from sklearn.preprocessing import LabelBinarizer
    def multiclass_roc_auc_score(truth, pred, average="macro"):

        lb = LabelBinarizer()
        lb.fit(truth)

        truth = lb.transform(truth)
        pred = lb.transform(pred)

        return roc_auc_score(truth, pred, average=average)


    auc = multiclass_roc_auc_score(test_labels, predictions)
    rocauc_score.append(auc)
print(error_rate)
print(rocauc_score)









## Grus J.,2015, Data science from scratch, O'Reilly


## plot the groph of n_estimators and error_rate for three experiments(Grus,2015)

n_estimators = [10,50,100,200,300,400,500,600,700,800,900]
error_rate_RF = [0.4035874439461884, 0.3923766816143498, 0.39910313901345296, 0.396
86098654708524, 0.3901345291479821, 0.3811659192825112, 0.37668161434977576, 0.3789
237668161435, 0.3789237668161435, 0.37443946188340804, 0.37668161434977576]
```

```
error_rate_CWsRF = [0.4439461883408071, 0.3946188340807175, 0.3901345291479821, 0.3
8565022421524664, 0.3946188340807175, 0.405829596412556, 0.39686098654708524, 0.396
86098654708524, 0.38789237668161436, 0.38789237668161436, 0.38565022421524664]
error_rate_OSRF = [0.4641255605381166, 0.44618834080717484, 0.42152466367713004, 0.
43273542600896864, 0.4304932735426009, 0.4260089686098655, 0.42376681614349776, 0.4
192825112107623, 0.42152466367713004, 0.42152466367713004, 0.4260089686098655]

plt.plot(n_estimators, error_rate_RF, 'b', label='RF' )
plt.plot(n_estimators, error_rate_CWsRF, 'r', label='CWsRF' )
plt.plot(n_estimators, error_rate_OSRF, 'g', label='OSRF' )
plt.legend(loc=9)
plt.title("(n-estimators - error_rate)")
plt.xlabel("n-estimators")
plt.ylabel("error_rate")
plt.show






## plot the groph of n_estimators and auc_roc for three experiments(Grus,2015)

n_estimators = [10,50,100,200,300,400,500,600,700,800,900]
auc_score_RF = [0.745418853315478, 0.7717805808178302, 0.7682019761348619, 0.770646
3092628373, 0.7741440317522306, 0.7762270541363433, 0.7773030567467545, 0.776719247
0706836, 0.7744734244406073, 0.7757917413115042, 0.7750888276140719]
auc_score_CwsRF = [0.7457758028975399, 0.7689511622813614, 0.7720145202660185, 0.77
12812005223472, 0.7687861676534699, 0.765486847097248, 0.7634552119893373, 0.763476
9433387263, 0.7678244212571438, 0.7678244212571438, 0.7705712075777842]
auc_score_OSRF = [0.7457936394338288, 0.7630941745689952, 0.7669542017594995, 0.772
1572298318675, 0.7668845500456993, 0.7693633364674624, 0.7716890317787058, 0.769983
8832470134, 0.7682077727677592, 0.7687287555166863, 0.7682922973009411]


plt.plot(n_estimators, auc_score_RF, 'b', label='RF' )
plt.plot(n_estimators, auc_score_CwsRF, 'r', label='CWsRF' )
plt.plot(n_estimators, auc_score_OSRF, 'g', label='OSRF' )
plt.legend(loc=4)
plt.title("(n-estimators - auc_roc_score)")
plt.xlabel("n-estimators")
plt.ylabel("auc")
plt.show
```