Birkbeck, University of London

Machine Learning - COIY065H7 Coursework Report

Baran Buluttekin

13153116

Abstract

This is placeholder text. To add more information type it after this line.

Contents

1 Introduction

This report is part of the coursework for *Machine Learning* (COIY065H7) module. From the coursework instructs Yeast data [7] has chosen as the dataset.

1.1 Dataset

Yeast dataset is part of the UCI machine learning repository where variety of machine learning related datasets curated. This dataset includes properties of yeast proteins to predict localization site of the protein. Number of data points in the dataset is 1484 and have 9 features in which only 8 of them are predictive feature. First feature is only index indicating database of that data acquired and have no predictive purpose. Final column of the data set provides label for protein class that we are aiming to predict. This dataset have no missing values. Given the labelled nature of the data this classification problem can be solved using *supervised learning.*

1.1.1 Problem Statement

There are some additional properties of this data makes classification task challenging. Firstly this is a multi class classification problem, because our dataset have 10 distinct labels. Labels extracted from dataset are, 'MIT', 'NUC', 'CYT', 'ME1', 'EXC', 'ME2', 'ME3', 'VAC', 'POX' and 'ERL'. Secondary challenge is that this dataset have class imbalance. Number of instances varies form 463 observation of 'CYT' to 5 instances of 'ERL'.

2 Algorithms and Techniques

There are number of techniques and algorithms introduced to address the challenges mentioned in the subsection 1.1.1. I will list some of this techniques below and explain more detailed in the fallowing sections.

- **Over-sampling data:** This method oversample the minority class in the data to create balanced classes.
- **Under-sampling data:** Opposite of the item above this method under sample the majority class to balance the data.
- **Balanced sampling for ensemble:** Sampling data in a balanced way while building ensemble models.
- **Choosing performance matrix sensitive to imbalance:** Using evaluation metric such as f1 score instead of accuracy will better capture mis-classification errors.

Most of the techniques discussed above are implemented as a code package in imbalance-learn [5]. In the case of over-sampling, only random sampling method available for multi class datasets. Choice of algorithm for under-sampling is substantially more than over-sampling. But due to the very large gap in the number of instances in the different classes, under-sampling can effect performance of the classification model significantly. For example if we were to under-sample until we reached to balance dataset because of the only 5 instance in the class 'ERL' we would get a dataset of 50 instances (10 class times 5 observation each). Regardless I will be using alternative under-sampling algorithm design to experiment their performance.

2.1 Under-sampling methods

In addition to random under-sampling method special algorithms to under-sample selectively will be used in experiments. One such algorithm is *Condensed nearest neighbour* (CNN) [3], utilizes one nearest neighbor rule to decide for each instance to be removed from the dataset or not. Algorithm steps from the article are [3]:

> "Get all minority samples in a set C.
>
> Add a sample from the targeted class (class to be under-sampled) in C and all other samples of this class in a set S.
>
> Go through the set S, sample by sample, and classify each sample using a 1 nearest neighbor rule.
>
> If the sample is misclassified, add it to C, otherwise do nothing.
>
> Reiterate on S until there is no samples to be added."

Neighbor cleaning rule (NCL) [4] is another algorithm that derived from CNN where more emphasis give to the data cleaning.

2.2 Sampling for ensemble

Ensemble models uses subset of the data with replacement while training different classifier but this sapling usually does not take class imbalance into account. Classifiers in the imbalance-learn library build in a way that sampling is done while maintaining the class balance. Below is the short description of these classifiers.

Balanced random forest [2] classifier is design such a way that each bootstrap sample to train tree in the forest is supplied with balanced class subset. This algorithm uses classic random forest [1] implementation and only differs in the sampling methodology.

RUSBoost [9] also randomly sample data in balanced way before the boosting implemented.

Easy Ensemble [6] is a AdaBoost [8] implementation trained on balanced bootstrap samples.

3 Methodology

As an initial step, I have loaded the dataset in a data frame which then allow for exploratory examination of the data. For that reason I have created a pairplot to observe if there is any linear separation is possible. With fallowing steps plot is created. All variable in the data is paired to each other except the ”pox” and ”erl” which behaves more like categorical data then continuous data.

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

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/yeast/
                                                yeast.data"
columns = ["mcg", "gvh", "alm", "mit", "erl", "pox", "vac", "nuc", "class"
                                                ]
df = pd.read_csv(url, header=None, sep=r"\s+", names=columns, usecols=list
                                                (range(1,10)))

feature_list = [x for x in df.columns if x not in ['pox', 'erl']]
# Plotting
sns.pairplot(df[feature_list], hue='class')
```

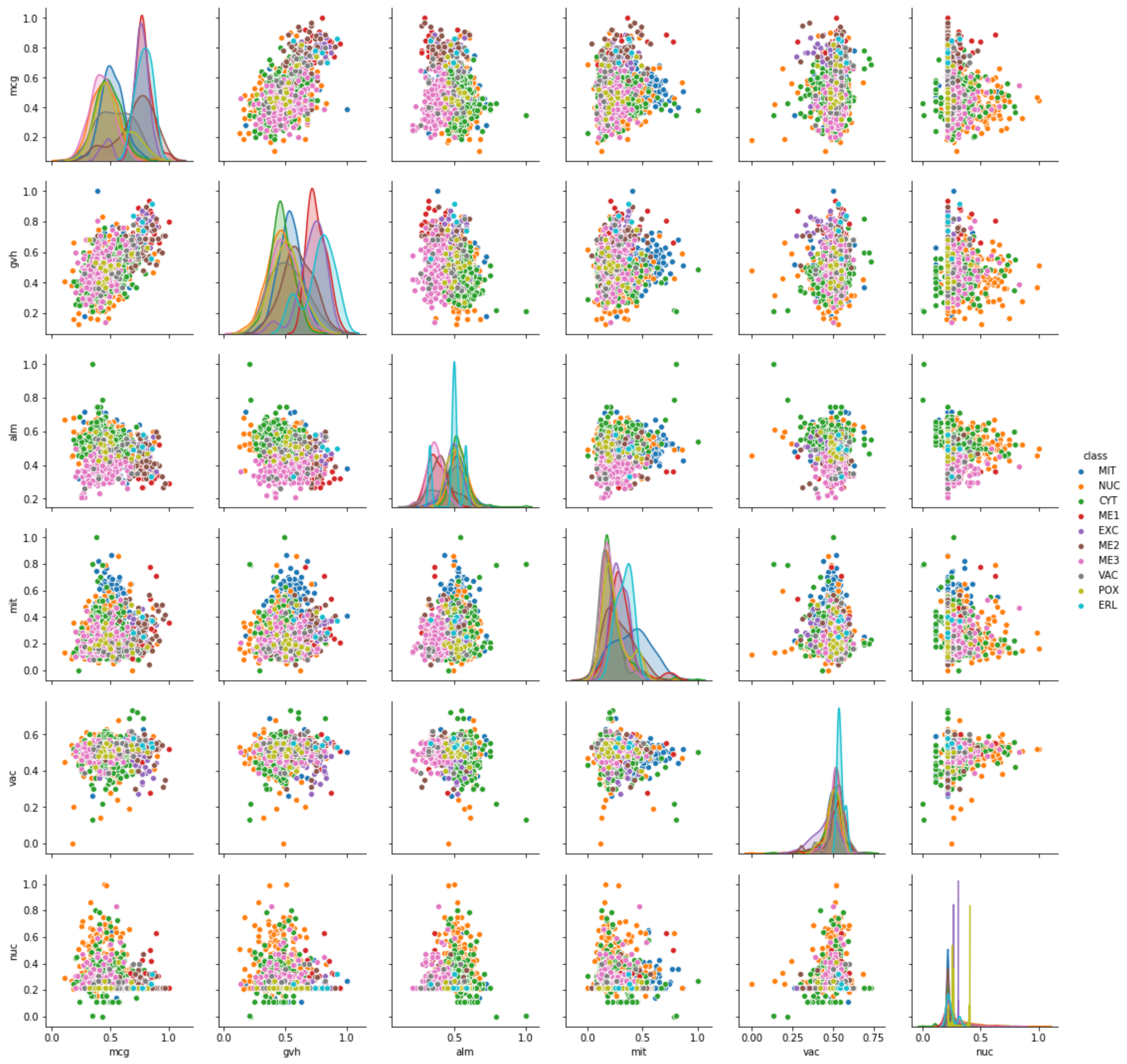

Figure 1: Pairplot of the data.

3.1 Data preprocessing

Data preprocessing is relatively simple for this dataset. There are no missing values and all data points are numeric which only leaves us to encode class variable to be able to feed into machine learning model. Judging from pairplot above classes have different range in some variable and can benefit min max scaling of the data.

This preparation can be streamlined with fallowing code.

```
def process_data(df):
    encoder = LabelEncoder()
```

```
        y = encoder.fit_transform(df['class'])
        X = df[df.columns[:-1]].values
        return X, y

    def process_scaled_data(df):
        l_encoder = LabelEncoder()
        scaler = MinMaxScaler()
        y = l_encoder.fit_transform(df['class'])
        X = df[df.columns[:-1]].values
        X = scaler.fit_transform(X)
        return X, y
```

3.2 Over-sampling and training model

Defined over-sampling method will be applied and compare the result with the same classifier trained on the data to observe if over-sampling help the classification task. Similarly utility function is created for the ease of transformation.

```
    def make_over_sample(X, y, random_state=0):
        ros = RandomOverSampler(random_state=random_state)
        return ros.fit_resample(X, y)
```

3.3 Under-sample and train model

I previously discuss that we have different algorithm for under-sampling and details for these algorithms given in the subsection 2.1. These algorithms implemented in imbalance-learn package as *CondensedNearestNeighbour*, *NeighbourhoodCleaningRule* and *RandomUnderSampler*. Similarly these will be compared to base models trained on the data to evaluate if under-sampling helps the classification. Also implemented with utility function.

```
    def make_under_sample(X, y, method=NeighbourhoodCleaningRule,
                                    random_state=0):
        clf = method(random_state=random_state)
        return clf.fit_resample(X, y)
```

3.4 Comparison of models

All of the experimentations will be carried out by firstly splitting data to training and testing data. For the splitting *StratifiedKFold* algorithm is chosen because of the minority class in the dataset. If the random splitting algorithm is used minority class can either not represented in test data or training data. StratifiedKFold will ensure each class is represented in the training and test dataset. All machine learning algorithm and sampling techniques will be applied to training data after the split so the effect of each process can be examined on unseen data when it tested with test data. This rule also applies to any scaling on the data. Applying scaling separately to test and training data especially important because applying scaling before splitting will cause information leakage and will cause to misleading results in testing.

Main choice of evaluation metric for this classification is f1 score because of its nature of combining precision and recall. F1 score calculates as harmonic mean of precision and recall, calculated with formula:

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{precision \times recall}{precision + recall}$$

Harmonic mean differ from arithmetic mean in a way that it will give more weight to low value, hence f1 score will increase only if both precision and recall increases.

In general first over-sampling and under-sampling method will be compared to classification models build on original data. Then another comparison will be drawn on balanced ensemble models versus general ensemble models which later all will be compared. This process will be repeated on the scaled data to observe scaling effect. All models will be trained with hyper-parameter tuning using cross validation.

4 Experiments and Results

4.1 Ensemble models performance

Selected ensemble models first trained with different hyper-parameters and best performing parameter selected to fit the data using number of estimators in all of the classifiers. Table below summarize the max f1 score for each classifier in training and test data.

Classifier	F1 score (training)	F1 score (test)
RandomForestClassifier	**1.0**	**0.6686**
AdaBoostClassifier	0.3670	0.3892
EasyEnsembleClassifier	0.5001	0.5917
BalancedRandomForestClassifier	0.4890	0.5148
BalancedBaggingClassifier	0.5633	0.6035

If we observe the f1 score of all the classifiers in the all n_estimators we can see after the random forest best performing imbalanced ensemble model was BalancedBaggingClassifier.

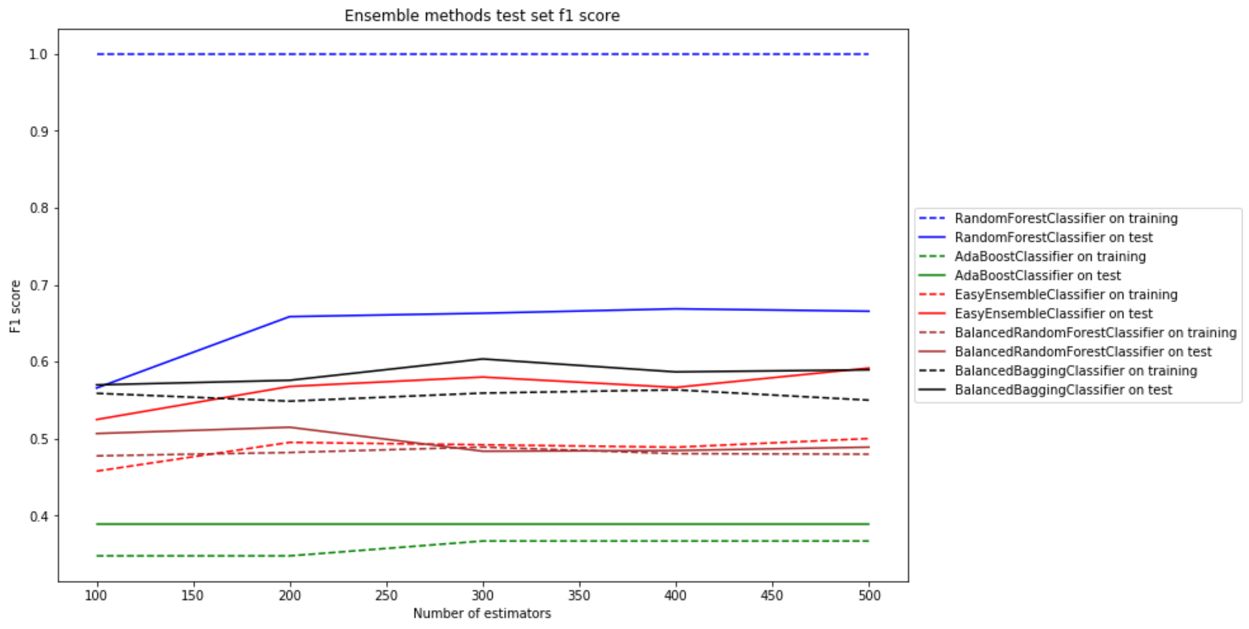

Figure 2: F1 score of ensemble models.

We can investigate RandomForestClassifier predictions further by visualizing its confusion matrix.

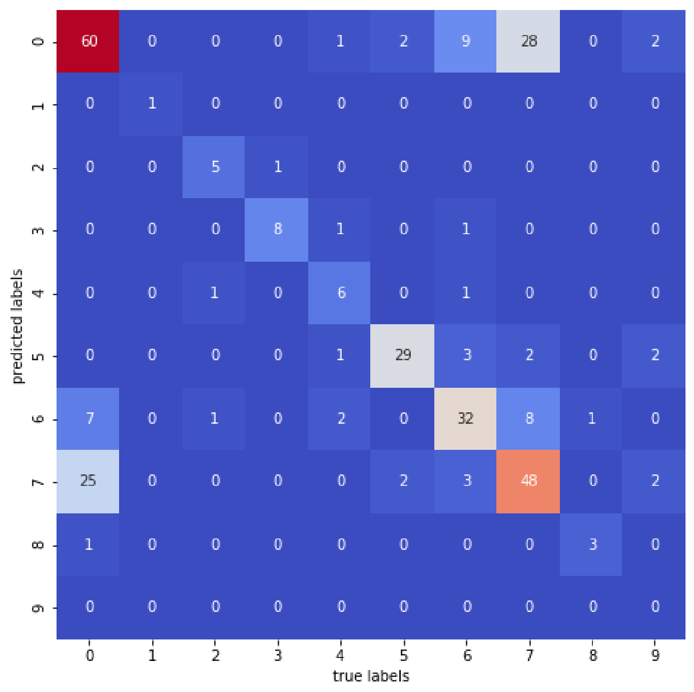

Figure 3: Confusion matrix of RandomForestClassifier.

We can see that ”CYT” (0) class is often mis-classified as ”NUC” (7) and vice versa. Rest of the classes predicted well.

4.2 Over-sampling v baseline classifiers

5 Conclusion

References

[1] Leo Breiman. “Random Forests”. In: *Mach. Learn.* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: `10.1023/A:1010933404324`. URL: `https://doi.org/10.1023/A:1010933404324`.

[2] Chao Chen. “Using Random Forest to Learn Imbalanced Data”. In: 2004.

[3] P. Hart. “The Condensed Nearest Neighbor Rule (Corresp.)” In: *IEEE Trans. Inf. Theor.* 14.3 (Sept. 2006), pp. 515–516. ISSN: 0018-9448. DOI: `10.1109/TIT.1968.1054155`. URL: `https://doi.org/10.1109/TIT.1968.1054155`.

[4] Jorma Laurikkala. “Improving Identification of Difficult Small Classes by Balancing Class Distribution”. In: *Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine*. AIME ’01. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 63–66. ISBN: 3-540-42294-3. URL: http://dl.acm.org/citation.cfm?id=648155.757340.

[5] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. “Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning”. In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5. URL: http://jmlr.org/papers/v18/16-365.

[6] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. “Exploratory Undersampling for Class-imbalance Learning”. In: *Trans. Sys. Man Cyber. Part B* 39.2 (Apr. 2009), pp. 539–550. ISSN: 1083-4419. DOI: 10.1109/TSMCB.2008.2007853. URL: http://dx.doi.org/10.1109/TSMCB.2008.2007853.

[7] Kenta Nakai and Paul Horton. *UCI Machine Learning Repository*. Sept. 1996. URL: http://archive.ics.uci.edu/ml/datasets/yeast.

[8] Robert E. Schapire. “A Brief Introduction to Boosting”. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’99. Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, pp. 1401–1406. URL: http://dl.acm.org/citation.cfm?id=1624312.1624417.

[9] C. Seiffert et al. “RUSBoost: A Hybrid Approach to Alleviating Class Imbalance”. In: *Trans. Sys. Man Cyber. Part A* 40.1 (Jan. 2010), pp. 185–197. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2009.2029559. URL: https://doi.org/10.1109/TSMCA.2009.2029559.