CSC 8740 - Advanced Data Mining
Homework Assignment 1

Class-imbalance (also referred to as the long-tail problem) occurs when the classes are not represented equally in a classification problem. This is quite common in practice, especially in many high-value applications (e.g., fraud detection, rare drug reaction prediction, natural disaster prediction, or prediction of gene families). Failure to account for class imbalance often leads to inaccurate predictions and reduced performance in classification algorithms. Imbalanced learning aims to tackle the class imbalance problem to learn an unbiased model from imbalanced data. In this assignment, you will use python (conda distribution) and Jupyter notebooks.

There are many libraries that tackle imbalance in supervised learning. In this homework assignment, you will use 'imbalanced-learn' (imported as `imblearn`–See the website from here: https://imbalanced-learn.org/stable/index.html), which is an open source library relying on popular 'scikit-learn' (imported as `sklearn`) and provides tools when dealing with classification with imbalanced classes. Before you start, please install imblearn to your conda environment. See the installation notes here: https://imbalanced-learn.org/stable/install.html.

Task 1: For this assignment, you are expected to use the abalone dataset[1]. To import this dataset, please use `imblearn` utility functions to ensure class labels are arranged correctly for imbalanced learning benchmarking. To load the dataset, you can use the following code sample:

```
from imblearn.datasets import fetch_datasets
abds = fetch_datasets()["abalone"]
X, y = abds.data, abds.target
```

Note that `fetch_datasets()` downloads datasets that are preprocessed for imbalanced learning.
Next, create a random training and testing data split. For this task, you will use random train_test_split function from sklearn. However, you are expected to use the last 3 digits of your Panther ID. Failure to do so will result in getting 0 from this question.

```
from sklearn.model_selection import train_test_split
last3digits_student_id = 100 # update this
p_seed = last3digits_student_id % 79
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=p_seed)
```

Task 2: Evaluation measures, you will be asked to use different strategies to handle and remedy the class imbalance issue. You are expected to report four evaluation measures for all the classifiers and settings you train:

- balanced accuracy (See `sklearn.metrics.balanced_accuracy_score`),
- geometric mean score (See `imblearn.metrics.geometric_mean_score`),
- macro-averaged F1-score (See `sklearn.metrics.f1_score`),
- Weighted-average F1-score (See `sklearn.metrics.f1_score`) and

[1] Abalone Dataset – https://archive.ics.uci.edu/dataset/1/abalone

- accuracy score.

Task 3: Create a dummy classifier. You can import one using

```
from sklearn.dummy import DummyClassifier
```

This classifier always predicts the majority class label for all instances. Use this as a baseline and report the evaluation metrics.

Task 4: In this task, train a simple decision tree classifier (`DecisionTreeClassifier`) with maximum height set to 4 (Use `max_depth`). Report the evaluation metrics.

Task 5: Apply random undersampling to the majority class in the training set and train your decision tree classifier with the same settings (in Task 4). Report the evaluation metrics. You can apply random sampling yourself, or use the `imblearn` utilities (such as `RandomUnderSampler` from `imblearn.under_sampling`).

Task 6: Another technique to handle the class imbalance is class upweighting. This is similar to simple oversampling (although not exactly random). To apply upweighting, you can change the `class_weight` argument of the classifier (e.g. `class_weight="balanced"`). Train your classifier with upweighting and report the evaluation metrics.

Task 7: An alternative to simple upweighting is the use of synthetic oversampling. One of the most well-known techniques is SMOTE (Synthetic Minority Over-sampling Technique). Using the SMOTE module from `imblearn` (`imblearn.over_sampling.SMOTE`), oversample the minority class in your training data and train your decision tree classifier. Report the evaluation metrics.

Task 8: Another way to handle class imbalance is the bagging approach. In this task, you are expected to train a collection of classifiers using the ensemble methods from `sklearn` and `imblearn`. As a baseline, use the `BaggingClassifier` from `sklearn`. Then, train `EasyEnsembleClassifier`, `RUSBoostClassifier`, `BalancedBaggingClassifier`, and `BalancedRandomForestClassifier`. Report the evaluation metrics. Please see the reference[2].

Task 9: Compare the results from Task 3 to Task 8. Please use appropriate charts to compare the results for different evaluation metrics. Properly label the models and axes in your charts. Identify the most effective model based on your results and provide a one-paragraph analysis.

Notes:

1. Please submit a JupyterNotebook for this homework. Before you submit, make sure you clear all the outputs and re-run all the cells.
2. If TA cannot run your code, you will not receive a grade for that question and TA will not attempt to fix your code or give partial credits if you have errors.

[2] Ensemble Methods – https://imbalanced-learn.org/stable/references/ensemble.html

CSC 8740 - Advanced Data Mining

Homework Assignment 1

3. Do not use other non-standard libraries. Your code is expected to run on a basic conda environment (with imblearn).
4. Do not create extra python files for utilities. Please implement them on the notebook itself if they are needed.

Distribution of points for individual tasks:

Task	Description	Points
1	Load dataset and split data	1
2	Report evaluation measures	15
3	Dummy classifier as baseline	4
4	Decision tree classifier	10
5	Random undersampling	10
6	Class upweighting	10
7	Synthetic oversampling (SMOTE)	20
8	Ensemble methods for class imbalance	20
9	Compare results and visualization	20