Classifying Ocean Plankton Using Multiclass Ensemble Techniques For Imbalanced Data

Jim Caine
caine.jim@gmail.com
March, 2015
DePaul University

Abstract

Classification on imbalanced datasets is a relevant topic for many real-world datasets. Various ensemble techniques have been proposed to deal with class imbalance. SMOTEBoost [1] and RUSBoost [2] are two leading approaches. In this paper, SMOTEBoost and RUSBoost are extended to a multinomial classification problem with imbalanced data. In both techniques, the main idea is to balance the classes using a random sampling procedure before each round of boosting. This idea is extended to two additional synthetic oversampling procedures using class centroids instead of k-nearest neighbors. Finally, random synthetic oversampling procedures are used as a pre-processing step before fitting random forests.

Keywords: Imbalanced classification, multinomial classification, SMOTEBoost, RUSBoost, AdaBoost, SMOTE, minority oversampling, plankton

1 Introduction

The study of plankton population in the ocean is of great interest to marine scientists because plankton population is a good indicator of ocean and ecosystem health. Researchers at Oregon State University have captured over 50 million images of plankton, resulting in over 80 TB of image data that must be analyzed. Manual analysis is not feasible, as one day's of image would take a year or more to manually analyze. This paper introduces an image classification algorithm to accurately predict the species of each plankton, and thus, allowing researchers to estimate the population of particular plankton throughout the ocean.

A major obstacle in image classification is representing the image as a set of features. Feature extraction has been studied extensively, and therefore, is not a major concentration of this paper. Three simple rules are described in Feature Mining for Image Classification [3], stating that the features extracted from an image should be informative, invariant to noise, and fast to compute. Based on these rules, two different approaches are used to extract features from the images in the dataset: radial boundary features and region features. These methods are explored in further detail in Section 3.1.

The dataset of interest consists of 30,000 images of plankton from 121 unique classes. As shown in Figure 1, the distribution of the classes in the dataset varies greatly from species to species. For example, the dataset contains 1,979 images of the majority species (trichodesmium puff), but only 10 instances of the most minority species. This imbalance in the dataset is likely to cause classification algorithms to be biased towards the species that have a larger number of instances (the majority class). This suspicion is confirmed after fitting several standard models to the dataset by comparing the recall for the minority classes and the majority classes.

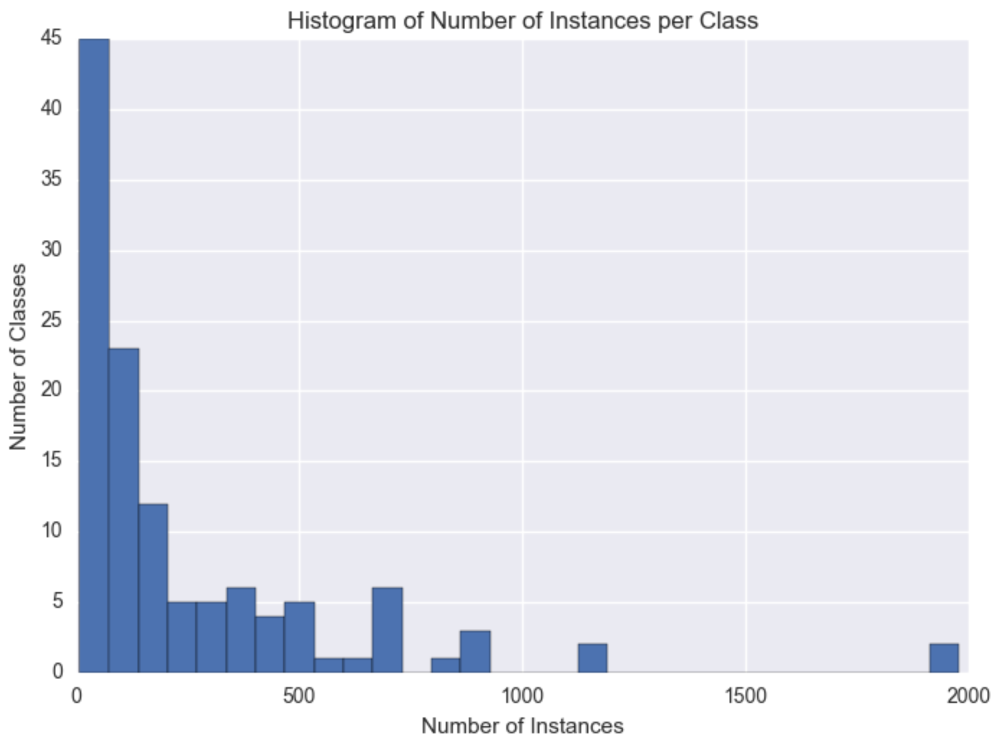

Figure 1: A histogram of the number of instances per species in the dataset shows large class imbalance in the dataset.

Several variations of the AdaBoost M.2 algorithm have been proposed to solve the class imbalance problem. Two leading approaches are SMOTEBoost [1] and RUSBoost [2]. In both techniques, the training data is modified at each boosting round to better balance the sample. In this paper, both SMOTEBoost and RUSBoost are extended to a multiclass algorithm by oversampling each class proportionally to balance the total dataset. These algorithms are then applied to the plankton dataset to classify each image into 1 of the 121 species. SMOTEBoost is then generalized to handle two new synthetic oversampling techniques that create synthetic vectors by projecting to the class centroid as opposed to a k-nearest neighbor that is done in SMOTE. Finally, the three synthetic oversampling techniques explored in the AdaBoost algorithm are used as a preprocessing step before fitting a random forest classifier to the data.

2 Related Work

Multiple frameworks for classification on imbalanced datasets have been proposed in the past. Galar et al. [4] survey common ensemble-based learning algorithms that are used to for classification on imbalanced datasets. In particular, an overview of common oversampling techniques is presented and a taxonomy for ensemble based-learning algorithms on imbalanced data is proposed. Within the taxonomy, there are four main branches: cost-sensitive boosting, boosting-based ensembles, bagging-based ensembles, and hybrid ensembles. Cost-sensitive boosting updates the weights of the minority class differently than the majority class. Boosting-based ensembles alter the weight distribution used to train the next classifier toward the minority class at each iteration. Bagging-based ensembles use a new sample of data (favoring the minority class) for each classifier. Finally, hybrid ensembles combine both the bagging and boosting techniques discussed prior.

SMOTEBoost [1] and RUSBoost [2] fall under the boosting-based ensembles branch and are both variations of the AdaBoost M.2 algorithm [5]. The AdaBoost M.2 algorithm is a variation of the original AdaBoost algorithm. In the AdaBoost M.2 algorithm, each instance is initialized a set of weights. For T iterations, a random sample of the data is selected based on the weights and a weak leaner is trained on this random sample of data. The pseudo-loss is calculated and the weights on each of the instances in the test set are recalculated. A complete description of the AdaBoost M.2 algorithm can be found in Freund and Schapire's paper 'Experiments with a New Boosting Algorithm'.

SMOTEBoost utilizes the AdaBoost M.2 algorithm, except randomly injects synthetic samples of the minority class into the training set at each boosting round to reduce the bias towards the majority class in the model. The synthetic samples are created using a technique called SMOTE (Synthetic Minority Oversampling TEchnique) [6]. SMOTE creates synthetic samples by projecting each minority instance towards the difference between that instance and one of it's k-nearest neighbors. The magnitude of the projection is randomly chosen in each dimension. Instead of applying SMOTE to the dataset before training an AdaBoost M.2 model, it is found that randomly injecting the synthetic samples during each iteration increases performance. By injecting synthetic samples of the minority class instances, the margin for the minority class is extended and therefore it should be easier for a weak learner to identify the minority cases. The random injection of synthetic samples also increases the diversity of the classifiers, as each classifier is presented with different synthetic samples. The amount of SMOTE that is added to the minority class is a parameter and will vary for different datasets.

RUSBoost (Random Undersampling Boost) utilizes the same algorithm as SMOTEBoost, however, instead of randomly injecting synthetic samples into the training set at each boosting round, RUSBoost reduces the size of the training set by randomly deleted instances from the majority class. The amount to instances to delete from the majority class is a parameter similar to the amount of SMOTE added to the minority class in SMOTEBoost. The primary advantages of RUSBoost compared to SMOTEBoost is in the speed of the algorithm. RUSBoost decreases the size of the training set, while SMOTEBoost increases the size of the training set, meaning that RUSBoost has far smaller training times. Also RUSBoost does not have to generate random synthetic samples during each round of boosting. The primary drawback to RUSBoost is loss of information inherent with sampling the majority class. This is often not the case, however, as many datasets show similar if not better performance on RUSBoost than SMOTEBoost.

Many other frameworks and sampling procedures that exist that have not been mentioned and are beyond the scope of this paper. A particular interesting approach is MSMOTEBoost [7]. (Modified SMOTEBoost). In MSMOTE, each instance is labeled as noise, safe, or a border point according to the composition to it's k-nearest neighbors. In theory, this should reduce the amount of noise that is being added to the dataset through the synthetic vectors. The label of can also be incorporated in the AdaBoost M.2 algorithm by reducing the weights of noise points during each boosting round.

3 Methodology

The experiments performed in this paper are applied to a dataset released by Kaggle in the National Data Science Bowl competition. The dataset consists of images of plankton. The dataset is split into a training set and a validation set, where the training set has a class label for each image, indicating which species the plankton is from out of 121 possible species. Because the validation set of data contains no class labels, it is not used throughout the paper. The training set contains over 30,000 images. The training set goes through a series of preprocessing steps as described in Section 3.1 to extract features from the raw image and normalize the features. The remaining dataset is then split into a training and testing set of data.

The majority of the algorithms evaluated in this paper rely on creating random synthetic instances of the minority class. Because it be computationally intensive to create new synthetic example at each iteration in an ensemble algorithm, synthetic vectors for each instance in the dataset are created a priori. This process is detailed in Section 3.2.

In Section 3.3, SMOTEBoost and RUSBoost are extended to a multinomial classification problem. Each algorithm is written to accept : the base learner, the number of classifiers in the ensemble (number of boosting rounds), and a ratio that controls the amount of synthetic minority instances that are added before each boosting round. The SMOTEBoost is then extended to handle other random oversampling techniques.

3.1 Feature Extraction and Preprocessing

Before applying any ensemble algorithms for classification, the images in the dataset must first be transformed into a feature set using feature extraction. The images of the plankton are not naturally aligned, meaning that the orientation and depth of plankton in the images vary, even if they are among the same class. Sample images for five different classes of plankton are shown in Figure 2. Although plankton of the same class appear similar by the eye, the image sizes, rotation of the plankton, and appearance of the plankton can differ greatly. Therefore, features that are extracted must be robust to the rotation and size of the images. Two different techniques fitting this criterion are used: (largest) region features and radial density features.

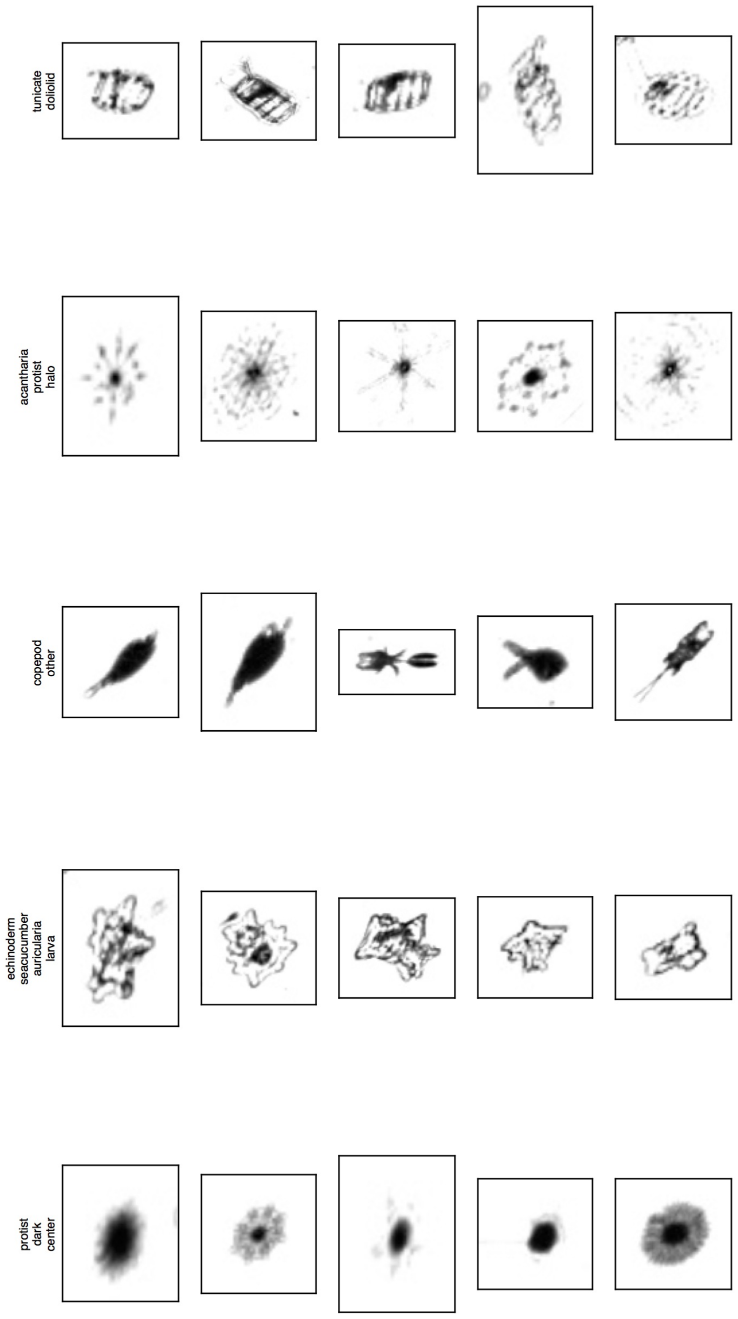

Figure 2: Images of five plankton for five separate classes show noise amongst the images

Radial image features are computed by looking at the color intensities (grey-scale pixel values) in radially symmetric regions across a square image. Each image is first resized to a NxN square image. The first region is the 4x4 square in the middle of the resized image. The pixel values in the four pixels contained in the 4x4 square are found, and four features describing the mean, standard deviation, skew, and kurtosis of the pixel values are used as features. Next, the square expands to an 8x8 square, but excludes any pixels that have already been used (in the 4x4 square). The same process is applied to this region and four additional features are found for each image. Figure 3 shows the regions for a 32x32 image. In this analysis, all the images were reshaped to a 16x16 pixel image, resulting in 4 distinct square regions and 16 attributes.

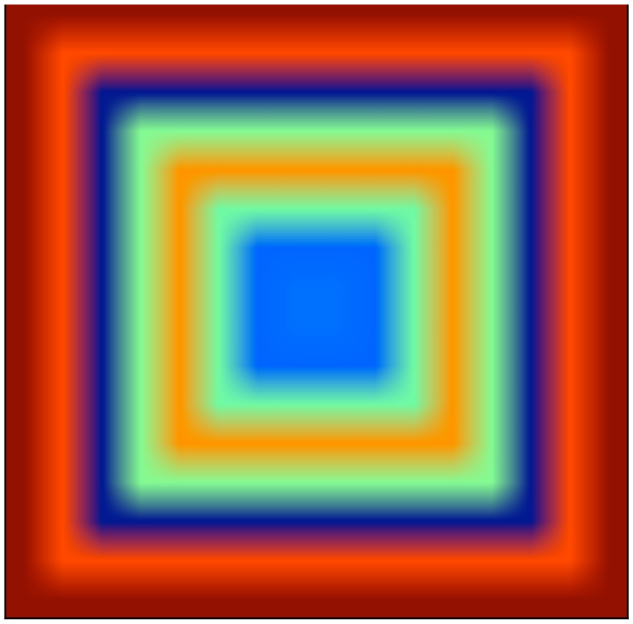

Figure 3: Example of Radial Boundaries For A 32x32 Image

This second feature extraction technique used in this analysis is region features. A number of mutations are performed on the image, resulting in the ability for the image to segment different regions of the image. Figure 4 shows examples of the image mutation process. First, the original image is converted into a binary scale based on a threshold of the mean pixel value of the image. The white pixels in the image are then dilated so that a 4x4 radius of each white pixel is turned white. This dilation is performed to decrease the noise of the image and therefore the noise in the amount of regions found. Regions are then found by finding parts of the image that have no connecting white pixels. The image can then be rotated and resized for direct pixel to pixel comparison, but this is not applied to the dataset to avoid redundancy at the benefit of dealing with less features in the dataset and decreased training times. Overall, six region features are used in this analysis and are described in Table 1.

Attribute	Description
Number of regions	The total number of disconnected regions after dilation
Eccentricity	The ratio between the distance between the minor and major axis of the largest region
Convex area	The number of pixels of the convex hull image of largest region
Convex to total area	The convex area of largest region divided by the total area of largest region
Extent	The ratio of pixels in the largest region to the pixels in the total bounding box
Filled area	The number of pixels of the largest region

Table 1: Description of Features Extracted By Analyzing Largest Region

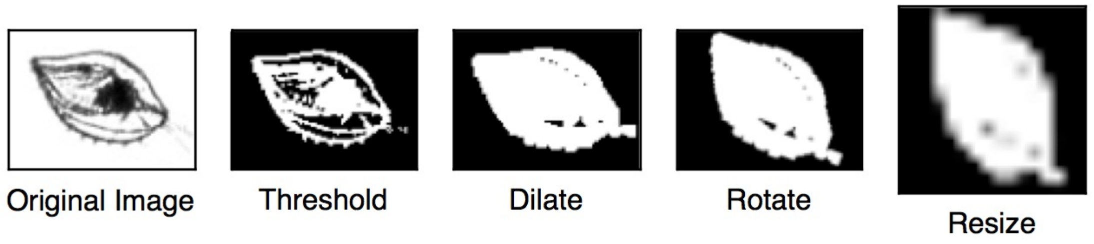

Figure 4: Series of mutations applied to each image to find the largest region and compare pixel by pixel features

The final dataset consists of 30,336 images and 22 numeric features. 70% of the dataset is then randomly partitioned into a train set and 30% is partitioned into a test set to test the classification algorithms.

3.2 Generating Synthetic Samples and Synthetic Datasets

Because it is computationally intensive to calculate synthetic data at each round of boosting in AdaBoost for multiple iterations and trials, a large amount of synthetic vectors for each instance in the training data is created a priori. In particular, three synthetic datasets are created: (1) SMOTE, (2) projection from class centroid (PFCC), and (3) projection from instance to class centroid (PFITCC). As described in Section 2, the SMOTE algorithm finds a k-nearest neighbor for each instance and projects the original instance in the direction of the different between the k-nearest neighbor and that instance at a random magnitude of zero through one. PFITCC works very similarly, except instead of projecting towards the different of a k-nearest neighbor, this technique projects the instance in the direction of the difference between that instance and the class centroid. Again, the magnitude of the projection is random in each dimension between zero and one. PFCC is slightly different, in that it does not consider each instance in the train set. Instead, it simply looks at the class centroid in the train set to generate synthetic instances. This technique simply projects randomly from the class centroid. For each technique, the magnitude of the projection can be changed using the parameter proj_ratio. This simply magnifies the random vector between zero and one by the proj_ratio. For example, if it is desired to increase the magnitude of the projections by a factor of two, a proj_ratio equal to 2 would be appropriate.

Algorithms 1-3 describe the three minority oversampling techniques used in this analysis. In SMOTE, the k-nearest neighbors is set to 1, meaning that the projection for each instance is always towards the instance's k-nearest neighbor. This seems like the most appropriate choice for this dataset due to the large magnitude of the number of classes. Changing this parameter could provide interesting results and is saved for future work. Various settings for the proj_ratio parameter are testing ranging 0.1 to 1.0.

Psuedo-algorithm 1: SMOTE

```
Input: Dataset D, parameter proj_ratio
synthetic_vectors = []
For each instance (i) in D
  Find the k-nearest neighbors of i
  Randomly choose one of the k-nearest neighbors
  Calculate the distance between the k-nearest neighbor and i
  Randomly adjust the magnitude of the difference by multiplying each dimension by a random
    number between 0 and proj_ratio
  Add the new difference to the original instance
  Append the new instance to the synthetic vector list
```

Psuedo-algorithm 2: PFCC

```
Input: Dataset D, parameter proj_ratio
synthetic_vectors = []
For each class (c) in D:
  Find the class centroid
  For N times:
    Randomly project from the class centroid in each dimension from –proj_ratio to +proj_ratio
    Append the new instance to the synthetic vector list
```

Psuedo-algorithm 3: PFITCC

```
Input: Dataset D, parameter proj_ratio
synthetic_vectors = []
For each instance (i) in D
  Find the class centroid for the instance
  Calculate the distance between the class centroid and i
  Randomly adjust the magnitude of the difference by multiplying each dimension by a random
    number between 0 and proj_ratio
  Add the new difference to the original instance
  Append the new instance to the synthetic vector list
```

The three sampling techniques provide an interesting relationship. SMOTE should be able to provide extended margins for the minority cases, but may be increasing noise in the dataset by projecting towards instances that are not of the same class. Using the cluster centroid instead should push the samples towards the cluster centroid, thus reducing noise at the cost of decreasing the margin for that particular class. Using the projections directly from the cluster center (PFCC) is an extreme case of PFITCC.

Various datasets are created a priori by combing the synthetic datasets and the dataframe containing the test and train data. Special attention is taken to remove the test instances from the synthetic dataset to ensure that the model will not be trained with test data.

3.3 AdaBoost M.2 With Random Oversampling

As described in Section 2, SMOTEBoost and RUSBoost are variations of the AdaBoost M.2 algorithm but randomly vary the training data before the model is fit during each boosting round. Following the same variation, SMOTEBoost is extended to the two additional sampling procedures that are described in Section 3.2: PFCC and PFITCC. The general algorithm for these three approaches is outlined in Algorithm 4. RUSBoost follows nearly the same algorithm, however, instead of adding synthetic instances from the minority class, RUSBoost randomly deletes some of the instances in the majority class.

Algorithm 4: AdaBoost M.2 With Random Oversample Injection For Multinomial Classification

```
Given: Set S with corresponding labels, parameter synthetic_ratio
Initialize: Number of synthetic samples to generate for each class
            Weights of the initialize dataset (set to 1/# instances)
For t = 1, 2, 3, ... , T:
  Identify hard cases by randomly sampling data points according to their weights
  For each class:
    Generate (synthetic_ratio * number of synthetic samples to generate for specific class)
    synthetic samples
  Train a weak learner on training set
  Calculate the error on original data set
  Update beta
  Update weights on original data set
Return: Class with highest vote OR weighted probability predictions
```

There are only two noticeable differences between Algorithm 4 and SMOTEBoost. First, Algorithm 4 only states to generate synthetic data, where as SMOTEBoost insists that the synthetic data is generated using SMOTE. Second, Algorithm 4 must initialize the number of synthetic samples to generate for each class. SMOTEBoost only concentrates on imbalanced binary classification tasks, and thus, does not provide a technique for understanding how many synthetic samples of each class must be generated. In Algorithm 4, the number of synthetic samples generated for each class is determined by an initialization of the number of synthetic samples needed to generate for each class and a parameter synthetic_ratio. For this multiclass problem, the number of synthetic examples to generate for each class is equal to the number of instances of the most majority class minus the number of instances for that particular class. The synthetic_ratio parameter than magnifies that by a factor equal to the ratio. This is slightly different for RUSBoost, where the number of samples for each class is equal to the minimum of all the classes or a constant (rus_ratio).

4 Results

Experiments on the algorithms and datasets discussed in Section 3 are applied to the plankton dataset to assess performance. Each classifier is evaluated by accuracy and multiclass log loss. Both train and test metrics are calculated to evaluate if the classifier over fits the dataset. Multiclass log loss is the multiclass equivalent of the logarithmic loss metric. This metric is improved when the prediction of the correct class is close to one, and is discounted when the prediction of the correct class is far from one. Good performing algorithms have a multiclass log loss value close to zero, while worse performing algorithms have higher multiclass log loss values. The formula for multiclass log loss is below:

$$logloss = -\frac{1}{N} * \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} * \log(p_{i,j})$$

where:

N is the number of observations

M is the number of class labels

$y_{i,j}$ is equal to one when the instance is in class j and zero otherwise

$p_{i,j}$ is the predicted probability that instance i is in class j

First, the AdaBoost M.2 algorithm along with the random minority sample injections discussed in Section 3 are applied to the dataset. The parameters are presented in Table 2. The minority sampling technique indicates how the training instances are sampled before training the model in each iteration of the AdaBoost M.2 algorithm. No sampling is equivalent to AdaBoost M.2 and SMOTE sampling is equivalent to SMOTEBoost. The number of estimators indicates how many models will be trained. The synthetic ratio controls how many synthetic instances will be added to the train set. A synthetic ratio of 1.0 indicates that the sum of real instances and synthetic instances will be equal for all classes. The performance for all iterations can be found in Appendix A.

Parameter	Parameter Values
Minority sampling technique	None (AdaBoost M.2), SMOTE (SMOTEBoost), PFCC, PFITCC, RUS
Number of estimators (boosting rounds)	2, 5, 10, 20
Synthetic ratio	0.01, 0.1, 1

Table 2: Parameters For AdaBoost M.2 Minority Sample Injection Algorithm

As shown in Figure 5, the AdaBoost algorithm performs very poorly and the performance actually decreases as the number of iterations grows. This is likely because of two reasons. First, the AdaBoost algorithm selects the instances that have already been misclassified. If these instances represent noise in the data, then the resulting classifiers in the ensemble are being trained on noisy data. Second, the base learner is a very simple decision tree with a maximum depth of 2. This particular weak learner may not be strong enough for the multiclass problem. Future work will need to be put in to test if performance increases as the weak learner actually gets stronger.

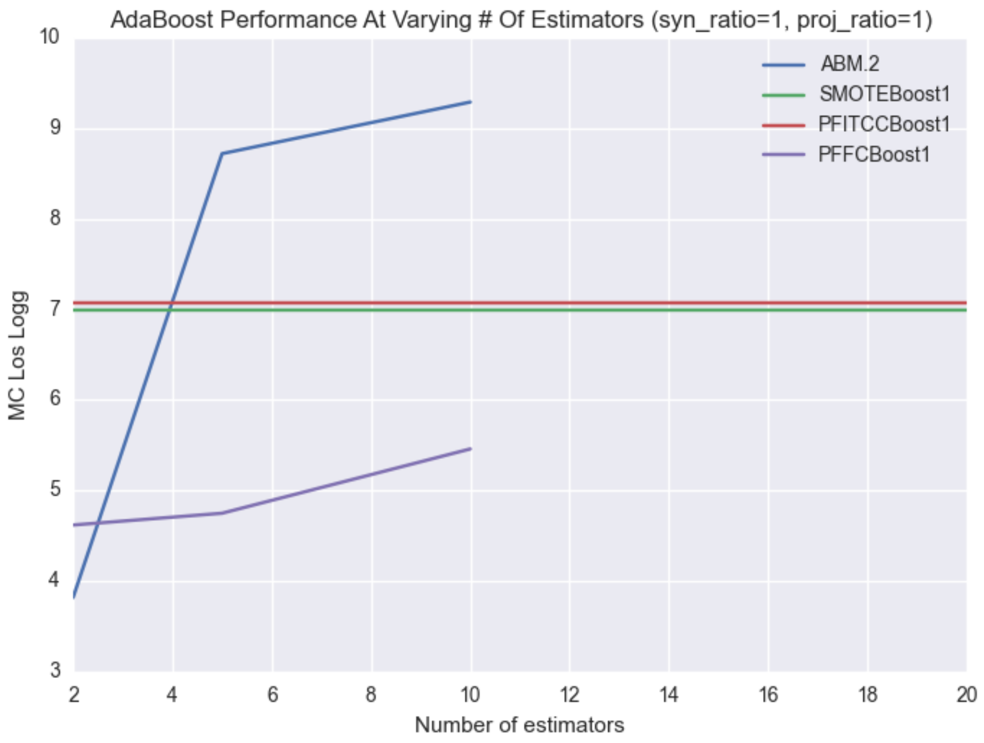

Figure 5: The AdaBoost model on all datasets actually decreases in performance as the number of iterations increase most likely due to the algorithm selecting noisy data or the weak classifier not being strong enough – ABM.2 and PFFCBoost1 have infinite multiclass log loss values at n_estimators = 20

Finally, a random forest algorithm is applied to the data. The parameters that are varied are presented in Table 3. The preprocessing sample procedure indicates what dataset is used to train the model. It is important to note that special attention is made to remove all instances in the test sample from the train sample to ensure validity of the test data. The number of synthetic minority cases in each dataset is chosen to be the difference between the number of samples in the majority class and the number of samples in that particular minority class, thus ensuring a balanced set of data. Different ratios can be tested in future work to optimize class balance. The projection distance refers to the magnitude of the projection in each synthetic instance as described in Section 3.2. The dataset grows very large after the synthetic oversampling techniques are applied. Therefore, training an ensemble classifier becomes computationally intensive. Because of this, the number of trees in the ensemble is limited to 25. For the original dataset, the number of trees is increased to 1,000. The performance for all iterations can be found in Appendix A.

Parameter	Parameter Values
Preprocessing sample procedure	None, SMOTE, PFCC, PFITCC
Projection distance	0.1, 1.0
Number of estimators (trees)	5,10,25
Max depth	10, 20, 30

Table 3: Parameters Random Forest Algorithm

As shown in Figure X, the random forest algorithm performs much better when the number of trees is increased, but at diminishing returns for very large number of trees. The classifiers with larger numbers of max depth perform poorly at low number of trees in the forest, but quickly improve as trees are added to the forest. This is likely because the more complex model is overfit at number of trees, but is soon generalized as the number of diverse classifiers in the ensemble increases. A similar graph is shown in Figure X but shows the random forest performance on synthetically sampled datasets. The SMOTEBoost algorithm performs the best out of all sampling procedures that are tested. This indicates that the expanding the margin of the minority class proves to be the best approach for this multiclass problem. Overall, random forest with the SMOTE dataset is the best performer. In fact, the random forest on the SMOTE with 25 trees outperforms the random forest on the original dataset with 1,000 trees. Also, as seen in Figure X, the more complex base learners (higher max depth) increase their performance with little trees added to the model. This is likely because the training set is larger and therefore a more complex base learner is necessary to find the patterns in the data. The downside to this is a much longer training time, but because the performance improves quickly in the first few iterations, the random forest on the SMOTE dataset is the preferred algorithm.

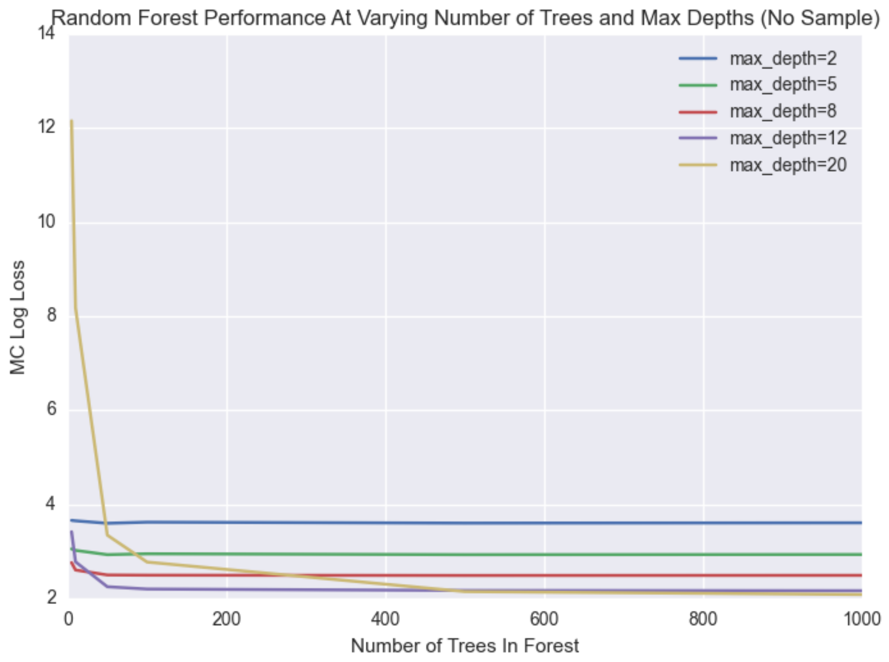

Figure 6: Random Forest (no sample) model performs better when more trees are in the ensemble. More complex base learners perform poor at smaller values of number of trees but quickly outperform the more simple base learners whose performance remains constant

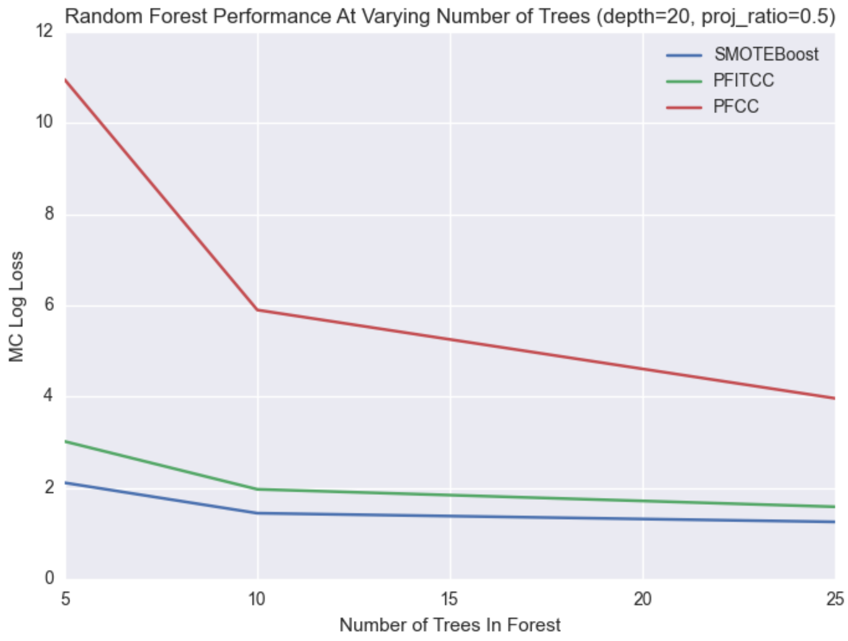

Figure 7: Random Forest on dataset with synthetic minority oversampling performance shows that SMOTEBoost is the best performer and outperforms Random Forest on the original dataset with far less trees in the model

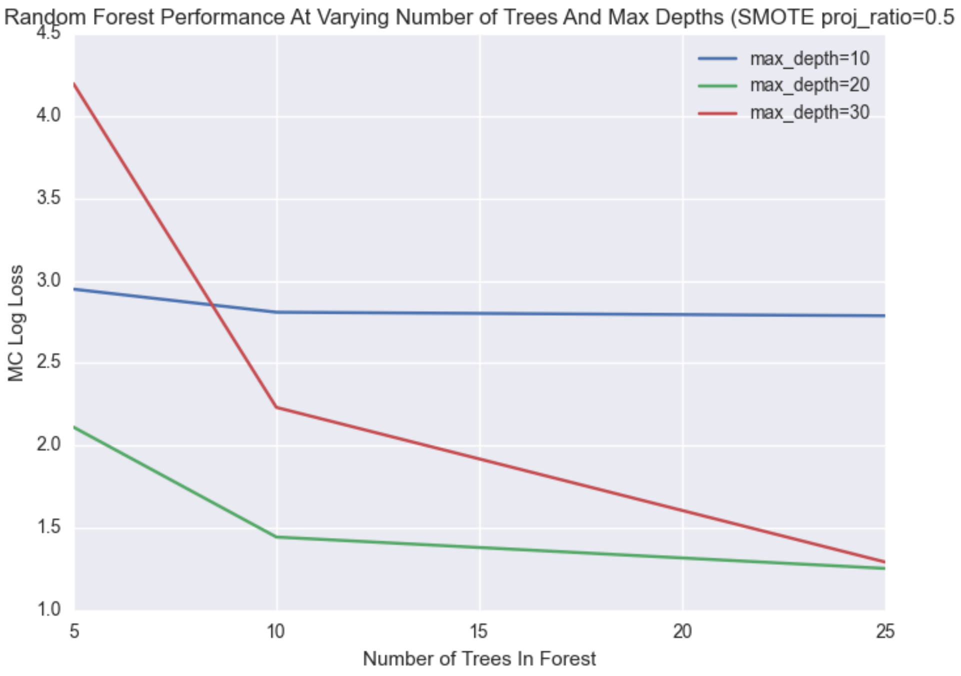

Figure 8: Random Forest performance for varying depths shows that more complex base learners quickly generalize their model only after a few iterations

As seen in Table 4, the top performing ensemble classifier was the random forest classifier on the SMOTE dataset. In general, the random forest classifiers outperformed the AdaBoost classifiers. As discussed earlier in this section, this is likely because AdaBoost prefers to sample misclassified instances, which are likely noise in the dataset. The random forest classifier does very well using more complicated base learners and the model quickly generalizes to the test set as the forest grows with trees.

Algorithm	Optimal Parameters	Train Accuracy	Test Accuracy	Train MC Log Loss	Test MC Log Loss
AdaBoost M.2	n_estimators=2; synthetic_rate=.01	13.75%	14.41%	3.795	3.807
RUSBoost	n_estimators=100, rr=100	7.78%	7.60%	4.189	4.340
SMOTEBoost	n_estimators=2; proj_ratio=1.0; synthetic_rate=0.1	4.42%	4.90%	4.656	4.675
PFITCCBoost	n_estimators=2; proj_ratio=1.0; synthetic_rate=0.1	5.70%	6.45%	4.551	4.545
PFCCBoost	n_estimators=2; proj_ratio=3.0; synthetic_rate=0.1	4.63%	13.69%	3.738	3.793
Random Forest	n_trees = 1,000; max_depth=20	99.95%	47.58%	0.427	2.084
Random Forest SMOTE	**n_trees = 25; max_depth=20**	**99.49%**	**77.85%**	**0.262**	**1.255**
Random Forest PFITCC	n_trees = 10; max_depth=20; proj_ratio=0.5	98.36%	60.48%	0.321	1.969
Random Forest PFCC	n_trees = 25; max_depth=10; proj_ratio=0.5	69.89%	23.27%	1.843	3.224

Table 4: Optimum results for all ensemble algorithms show Random Forest on the SMOTE dataset is the best performing classifier

5 Conclusions and Future Work

Various ensemble learning algorithms are applied to the real-world dataset of plankton images in an attempt to minimize the multiclass log loss value. First, boosting algorithms that inject random samples of synthetic data (or undersamples for RUSBoost) are applied to the dataset. These boosting algorithms did not improve the classification over a even a weak learner, and in fact, decreased performance over each iteration. This is likely because the AdaBoost algorithm prefers to select misclassified examples which is often the noisiest points in the data.

Synthetic sampling techniques were also used to preprocess the dataset to balance the minority classes and then are applied to the random forest classifier with great success. The size of the dataset is increased due to the injection of many synthetic samples and this causes the training time to increase greatly. That being said, more complex learners (deep decision trees) are seen to generalize their model to the test data after adding very few trees to the random forest. Overall, the random forest with the SMOTE dataset is the best performing classifier. After adding only 25 trees to the forest, the test multiclass log loss reached a minimum of 1.255, nearly 2 times better than any other classifier. The SMOTE dataset is thought to outperform the datasets with synthetic instances based on cluster because the SMOTE dataset increases the margin for the minority cases, allowing the classifier to become less biased towards the majority cases.

This research has sparked a lot of curiosity into why the AdaBoost algorithm did not perform better. Future work could be directed to understanding why this occurs and trying to correct it. In particular, the MSMOTE boost algorithm described in Section 2 could utilized to increase the samples of safe cases and decrease the noise that goes into each iteration of the AdaBoost algorithm. Further, computational limitations capped the number of trees grown in the Random Forest SMOTE classifier at 25. Deeper forests should be attempted on this algorithm to see how much more performance can be improved.

6 Appendix A – Results

Boosting Algorithms							
bl	ne	sd	sr	acctrain	acctest	kstrain	kstest
DT(max_depth=2)	2	m2	0.01	13.75%	14.41%	3.7948	3.8073
DT(max_depth=2)	2	m2	0.1	13.72%	14.40%	3.7944	3.8262
DT(max_depth=2)	2	m2	1	13.64%	14.29%	3.7925	3.8236
DT(max_depth=2)	2	smote1	0.01	4.42%	4.90%	4.6559	4.6754
DT(max_depth=2)	2	smote1	0.1	0.11%	0.09%	5.8797	5.8930
DT(max_depth=2)	2	smote1	1	0.10%	0.09%	6.9835	6.9961
DT(max_depth=2)	2	smote2	0.01	4.66%	5.26%	4.6078	4.6299
DT(max_depth=2)	2	smote2	0.1	0.10%	0.09%	5.9770	5.9924
DT(max_depth=2)	2	smote2	1	0.10%	0.09%	6.9976	7.0103
DT(max_depth=2)	2	smote3	0.01	5.20%	6.16%	4.6239	4.6190
DT(max_depth=2)	2	smote3	0.1	0.10%	0.11%	6.0109	6.0076
DT(max_depth=2)	2	smote3	1	0.09%	0.08%	6.9576	6.9669
DT(max_depth=2)	2	smote01	0.01	5.04%	5.66%	4.5958	4.6237
DT(max_depth=2)	2	smote01	0.1	0.10%	0.11%	5.9084	5.9142
DT(max_depth=2)	2	smote01	1	0.10%	0.11%	6.9164	6.9253
DT(max_depth=2)	2	smote05	0.01	4.72%	5.38%	4.6610	4.6789
DT(max_depth=2)	2	smote05	0.1	0.10%	0.10%	5.9194	5.9236
DT(max_depth=2)	2	smote05	1	0.10%	0.10%	6.9248	6.9341
DT(max_depth=2)	5	m2	0.01	13.84%	14.70%	8.4945	8.4205
DT(max_depth=2)	5	m2	0.1	13.95%	14.77%	7.7386	7.6613
DT(max_depth=2)	5	m2	1	14.54%	15.30%	8.8064	8.7183
DT(max_depth=2)	5	smote1	0.01	4.53%	5.09%	4.7542	4.7597
DT(max_depth=2)	5	smote1	0.1	0.10%	0.09%	5.9719	5.9844
DT(max_depth=2)	5	smote1	1	0.10%	0.09%	6.9846	6.9970
DT(max_depth=2)	5	smote2	0.01	4.70%	5.26%	4.8515	4.8456
DT(max_depth=2)	5	smote2	0.1	0.10%	0.09%	5.9799	5.9838
DT(max_depth=2)	5	smote2	1	0.10%	0.09%	6.9478	6.9569
DT(max_depth=2)	5	smote3	0.01	4.70%	5.24%	4.7780	4.7591
DT(max_depth=2)	5	smote3	0.1	0.09%	0.08%	5.9387	5.9353
DT(max_depth=2)	5	smote3	1	0.09%	0.09%	7.0044	7.0112
DT(max_depth=2)	5	smote01	0.01	3.34%	3.98%	4.8702	4.8597
DT(max_depth=2)	5	smote01	0.1	0.12%	0.14%	5.7745	5.7711
DT(max_depth=2)	5	smote01	1	0.10%	0.11%	6.9155	6.9237
DT(max_depth=2)	5	smote05	0.01	3.37%	3.99%	4.7383	4.7303
DT(max_depth=2)	5	smote05	0.1	0.10%	0.10%	5.9679	5.9849
DT(max_depth=2)	5	smote05	1	0.10%	0.10%	6.9223	6.9317
DT(max_depth=2)	10	m2	0.01	13.97%	14.73%	8.1956	8.1911

DT(max_depth=2)	10	m2	0.1	13.82%	14.60%	8.7592	8.6410
DT(max_depth=2)	10	m2	1	13.91%	14.68%	9.2985	9.2883
DT(max_depth=2)	10	smote1	0.01	4.00%	4.55%	9.4871	9.4283
DT(max_depth=2)	10	smote1	0.1	0.10%	0.09%	5.9711	5.9953
DT(max_depth=2)	10	smote1	1	0.10%	0.09%	6.9852	6.9984
DT(max_depth=2)	10	smote2	0.01	4.11%	4.68%	10.3752	10.3766
DT(max_depth=2)	10	smote2	0.1	0.09%	0.09%	5.9969	6.0099
DT(max_depth=2)	10	smote2	1	0.10%	0.09%	6.9985	7.0114
DT(max_depth=2)	10	smote3	0.01	4.19%	4.84%	9.4191	9.4183
DT(max_depth=2)	10	smote3	0.1	0.08%	0.08%	6.0107	6.0173
DT(max_depth=2)	10	smote3	1	0.09%	0.08%	6.9506	6.9597
DT(max_depth=2)	10	smote01	0.01	3.36%	3.97%	6.6498	6.6101
DT(max_depth=2)	10	smote01	0.1	0.10%	0.11%	5.9121	5.9202
DT(max_depth=2)	10	smote01	1	0.10%	0.11%	6.9234	6.9300
DT(max_depth=2)	10	smote05	0.01	3.77%	4.27%	9.6985	9.6546
DT(max_depth=2)	10	smote05	0.1	0.10%	0.10%	5.9179	5.9238
DT(max_depth=2)	10	smote05	1	0.10%	0.10%	6.9293	6.9365
DT(max_depth=2)	20	m2	0.01	13.92%	14.74%	8.7495	8.8442
DT(max_depth=2)	20	m2	0.1	0.11%	0.11%	n/a	n/a
DT(max_depth=2)	20	m2	1	14.03%	14.96%	n/a	n/a
DT(max_depth=2)	20	smote1	0.01	0.07%	0.05%	n/a	n/a
DT(max_depth=2)	20	smote1	0.1	0.10%	0.09%	5.9728	5.9968
DT(max_depth=2)	20	smote1	1	0.10%	0.09%	6.9837	6.9976
DT(max_depth=2)	20	smote2	0.01	0.07%	0.05%	n/a	n/a
DT(max_depth=2)	20	smote2	0.1	0.10%	0.11%	6.0528	6.0619
DT(max_depth=2)	20	smote2	1	0.10%	0.09%	6.9972	7.0098
DT(max_depth=2)	20	smote3	0.01	4.27%	4.88%	8.1236	8.2222
DT(max_depth=2)	20	smote3	0.1	0.10%	0.11%	6.0215	6.0144
DT(max_depth=2)	20	smote3	1	0.09%	0.08%	7.0238	7.0318
DT(max_depth=2)	20	smote01	0.01	3.96%	4.56%	9.2005	9.2072
DT(max_depth=2)	20	smote01	0.1	0.10%	0.11%	5.9120	5.9266
DT(max_depth=2)	20	smote01	1	0.10%	0.11%	6.9168	6.9260
DT(max_depth=2)	20	smote05	0.01	0.07%	0.08%	n/a	n/a
DT(max_depth=2)	20	smote05	0.1	0.10%	0.10%	5.9151	5.9226
DT(max_depth=2)	20	smote05	1	0.10%	0.10%	6.9252	6.9345
DT(max_depth=2)	2	PFFC1	0.01	14.39%	13.75%	3.7479	3.7989
DT(max_depth=2)	2	PFFC1	0.1	14.25%	13.67%	3.8901	3.9368
DT(max_depth=2)	2	PFFC1	1	6.78%	6.84%	4.7206	4.7458
DT(max_depth=2)	2	PFFC2	0.01	14.38%	13.75%	3.7424	3.7954
DT(max_depth=2)	2	PFFC2	0.1	14.90%	14.34%	3.8516	3.8839

DT(max_depth=2)	2	PFFC2	1	7.40%	7.35%	4.7269	4.7553
DT(max_depth=2)	2	PFFC3	0.01	14.63%	13.96%	3.7381	3.7935
DT(max_depth=2)	2	PFFC3	0.1	14.82%	14.41%	3.8726	3.9041
DT(max_depth=2)	2	PFFC3	1	7.27%	7.22%	4.7159	4.7203
DT(max_depth=2)	2	PFFC0.1	0.01	14.36%	13.73%	3.7718	3.8325
DT(max_depth=2)	2	PFFC0.1	0.1	14.38%	13.74%	3.8929	3.9459
DT(max_depth=2)	2	PFFC0.1	1	6.43%	6.42%	4.6912	4.7291
DT(max_depth=2)	2	PFFC0.5	0.01	14.28%	13.66%	3.7746	3.8263
DT(max_depth=2)	2	PFFC0.5	0.1	14.17%	13.80%	3.8956	3.9469
DT(max_depth=2)	2	PFFC0.5	1	6.56%	6.54%	4.7580	4.7815
DT(max_depth=2)	5	PFFC1	0.01	14.36%	13.85%	6.9594	6.8914
DT(max_depth=2)	5	PFFC1	0.1	14.20%	13.84%	4.8619	4.8343
DT(max_depth=2)	5	PFFC1	1	11.00%	10.78%	4.6271	4.6524
DT(max_depth=2)	5	PFFC2	0.01	14.34%	13.81%	6.9701	6.9357
DT(max_depth=2)	5	PFFC2	0.1	14.40%	13.76%	5.0362	4.9784
DT(max_depth=2)	5	PFFC2	1	7.40%	7.32%	4.7497	4.7837
DT(max_depth=2)	5	PFFC3	0.01	14.48%	13.98%	7.5928	7.5666
DT(max_depth=2)	5	PFFC3	0.1	14.82%	14.40%	4.9921	4.9479
DT(max_depth=2)	5	PFFC3	1	7.30%	7.25%	4.7458	4.7644
DT(max_depth=2)	5	PFFC0.1	0.01	14.32%	13.81%	6.8529	6.8527
DT(max_depth=2)	5	PFFC0.1	0.1	14.30%	13.79%	6.1076	6.1352
DT(max_depth=2)	5	PFFC0.1	1	14.43%	13.74%	4.4901	4.5230
DT(max_depth=2)	5	PFFC0.5	0.01	14.34%	14.01%	6.0305	6.0025
DT(max_depth=2)	5	PFFC0.5	0.1	14.25%	13.88%	5.8942	5.8299
DT(max_depth=2)	5	PFFC0.5	1	9.23%	8.97%	4.5412	4.5864
DT(max_depth=2)	10	PFFC1	0.01	14.40%	13.87%	7.5799	7.4431
DT(max_depth=2)	10	PFFC1	0.1	14.20%	13.66%	5.5445	5.4122
DT(max_depth=2)	10	PFFC1	1	6.32%	5.99%	5.5176	5.4582
DT(max_depth=2)	10	PFFC2	0.01	14.42%	13.97%	7.8320	7.7040
DT(max_depth=2)	10	PFFC2	0.1	14.45%	13.85%	n/a	n/a
DT(max_depth=2)	10	PFFC2	1	6.85%	6.65%	5.3508	5.3412
DT(max_depth=2)	10	PFFC3	0.01	14.34%	13.85%	7.1594	7.1105
DT(max_depth=2)	10	PFFC3	0.1	14.52%	13.98%	5.6136	5.6338
DT(max_depth=2)	10	PFFC3	1	6.87%	6.67%	5.1924	5.1752
DT(max_depth=2)	10	PFFC0.1	0.01	14.33%	13.79%	7.5982	7.5606
DT(max_depth=2)	10	PFFC0.1	0.1	14.29%	13.77%	n/a	n/a
DT(max_depth=2)	10	PFFC0.1	1	13.73%	13.37%	6.9196	6.7867
DT(max_depth=2)	10	PFFC0.5	0.01	14.40%	13.88%	7.0989	7.0338
DT(max_depth=2)	10	PFFC0.5	0.1	14.25%	13.87%	n/a	n/a
DT(max_depth=2)	10	PFFC0.5	1	9.24%	8.99%	5.4483	5.3962

DT(max_depth=2)	20	PFFC1	0.01	14.31%	13.78%	7.8000	7.6724
DT(max_depth=2)	20	PFFC1	0.1	0.58%	0.48%	n/a	n/a
DT(max_depth=2)	20	PFFC1	1	0.58%	0.47%	n/a	n/a
DT(max_depth=2)	20	PFFC2	0.01	0.20%	0.20%	n/a	n/a
DT(max_depth=2)	20	PFFC2	0.1	0.13%	0.14%	n/a	n/a
DT(max_depth=2)	20	PFFC2	1	0.11%	0.14%	n/a	n/a
DT(max_depth=2)	20	PFFC3	0.01	0.16%	0.11%	n/a	n/a
DT(max_depth=2)	20	PFFC3	0.1	0.19%	0.21%	n/a	n/a
DT(max_depth=2)	20	PFFC3	1	0.12%	0.12%	n/a	n/a
DT(max_depth=2)	20	PFFC0.1	0.01	14.40%	13.89%	7.5922	7.5077
DT(max_depth=2)	20	PFFC0.1	0.1	0.65%	0.57%	n/a	n/a
DT(max_depth=2)	20	PFFC0.1	1	0.73%	0.60%	n/a	n/a
DT(max_depth=2)	20	PFFC0.5	0.01	0.63%	0.57%	n/a	n/a
DT(max_depth=2)	20	PFFC0.5	0.1	0.78%	0.65%	n/a	n/a
DT(max_depth=2)	20	PFFC0.5	1	0.61%	0.54%	n/a	n/a
DT(max_depth=2)	2	PFITCC1	0.01	5.70%	6.45%	4.5512	4.5453
DT(max_depth=2)	2	PFITCC1	0.1	0.10%	0.08%	6.0651	6.0671
DT(max_depth=2)	2	PFITCC1	1	0.10%	0.12%	7.0764	7.0847
DT(max_depth=2)	2	PFITCC2	0.01	0.10%	0.14%	4.6972	4.6984
DT(max_depth=2)	2	PFITCC2	0.1	0.06%	0.05%	6.1302	6.1209
DT(max_depth=2)	2	PFITCC2	1	0.08%	0.12%	6.9633	6.9533
DT(max_depth=2)	2	PFITCC3	0.01	4.77%	5.62%	4.8223	4.8241
DT(max_depth=2)	2	PFITCC3	0.1	0.05%	0.10%	6.1022	6.0950
DT(max_depth=2)	2	PFITCC3	1	0.08%	0.05%	7.1396	7.1442
DT(max_depth=2)	2	PFITCC0.1	0.01	4.94%	5.73%	4.6686	4.6765
DT(max_depth=2)	2	PFITCC0.1	0.1	0.10%	0.10%	5.9749	5.9824
DT(max_depth=2)	2	PFITCC0.1	1	0.11%	0.12%	6.9570	6.9625
DT(max_depth=2)	2	PFITCC0.5	0.01	4.91%	5.70%	4.7499	4.7429
DT(max_depth=2)	2	PFITCC0.5	0.1	0.12%	0.13%	6.0304	6.0320
DT(max_depth=2)	2	PFITCC0.5	1	0.11%	0.09%	7.0721	7.0806
DT(max_depth=2)	5	PFITCC1	0.01	4.79%	5.38%	4.7473	4.7306
DT(max_depth=2)	5	PFITCC1	0.1	0.12%	0.14%	5.8455	5.8307
DT(max_depth=2)	5	PFITCC1	1	0.10%	0.12%	7.0724	7.0811
DT(max_depth=2)	5	PFITCC2	0.01	4.87%	5.55%	4.9140	4.9152
DT(max_depth=2)	5	PFITCC2	0.1	0.12%	0.16%	5.8796	5.8654
DT(max_depth=2)	5	PFITCC2	1	0.08%	0.12%	6.9706	6.9607
DT(max_depth=2)	5	PFITCC3	0.01	5.10%	5.79%	4.8553	4.8361
DT(max_depth=2)	5	PFITCC3	0.1	0.08%	0.05%	6.0968	6.0872
DT(max_depth=2)	5	PFITCC3	1	0.08%	0.05%	7.1371	7.1416
DT(max_depth=2)	5	PFITCC0.1	0.01	8.22%	8.75%	5.4570	5.4256

DT(max_depth=2)	5	PFITCC0.1	0.1	0.11%	0.11%	5.9677	5.9761
DT(max_depth=2)	5	PFITCC0.1	1	0.11%	0.12%	6.9561	6.9642
DT(max_depth=2)	5	PFITCC0.5	0.01	4.59%	5.23%	4.7656	4.7444
DT(max_depth=2)	5	PFITCC0.5	0.1	0.10%	0.08%	6.0667	6.0644
DT(max_depth=2)	5	PFITCC0.5	1	0.11%	0.09%	7.0695	7.0776
DT(max_depth=2)	10	PFITCC1	0.01	4.71%	5.31%	9.0054	9.0589
DT(max_depth=2)	10	PFITCC1	0.1	0.11%	0.14%	5.9447	5.9314
DT(max_depth=2)	10	PFITCC1	1	0.10%	0.12%	7.0715	7.0802
DT(max_depth=2)	10	PFITCC2	0.01	5.12%	5.80%	9.2794	9.3348
DT(max_depth=2)	10	PFITCC2	0.1	0.07%	0.07%	6.1033	6.0907
DT(max_depth=2)	10	PFITCC2	1	0.08%	0.12%	6.9625	6.9526
DT(max_depth=2)	10	PFITCC3	0.01	5.18%	5.85%	7.7897	7.7844
DT(max_depth=2)	10	PFITCC3	0.1	0.06%	0.05%	6.1192	6.1093
DT(max_depth=2)	10	PFITCC3	1	0.08%	0.05%	7.1334	7.1378
DT(max_depth=2)	10	PFITCC0.1	0.01	9.71%	10.26%	7.0443	7.1659
DT(max_depth=2)	10	PFITCC0.1	0.1	0.11%	0.13%	5.8398	5.8457
DT(max_depth=2)	10	PFITCC0.1	1	0.11%	0.12%	6.9551	6.9631
DT(max_depth=2)	10	PFITCC0.5	0.01	4.43%	5.11%	6.8456	6.8082
DT(max_depth=2)	10	PFITCC0.5	0.1	0.12%	0.13%	6.0394	6.0489
DT(max_depth=2)	10	PFITCC0.5	1	0.10%	0.09%	7.0621	7.0612
DT(max_depth=2)	20	PFITCC1	0.01	4.69%	5.24%	6.2073	6.2826
DT(max_depth=2)	20	PFITCC1	0.1	0.09%	0.11%	5.9558	5.9408
DT(max_depth=2)	20	PFITCC1	1	0.10%	0.12%	7.0713	7.0801
DT(max_depth=2)	20	PFITCC2	0.01	0.03%	0.05%	n/a	n/a
DT(max_depth=2)	20	PFITCC2	0.1	0.09%	0.12%	5.9760	5.9663
DT(max_depth=2)	20	PFITCC2	1	0.08%	0.12%	6.9625	6.9521
DT(max_depth=2)	20	PFITCC3	0.01	0.05%	0.07%	n/a	n/a
DT(max_depth=2)	20	PFITCC3	0.1	0.06%	0.05%	6.1206	6.1087
DT(max_depth=2)	20	PFITCC3	1	0.08%	0.05%	7.1392	7.1439
DT(max_depth=2)	20	PFITCC0.1	0.01	0.08%	0.07%	n/a	n/a
DT(max_depth=2)	20	PFITCC0.1	0.1	0.12%	0.15%	5.8024	5.7965
DT(max_depth=2)	20	PFITCC0.1	1	0.12%	0.15%	6.8158	6.8158
DT(max_depth=2)	20	PFITCC0.5	0.01	0.06%	0.07%	n/a	n/a
DT(max_depth=2)	20	PFITCC0.5	0.1	0.12%	0.13%	6.0263	6.0279
DT(max_depth=2)	20	PFITCC0.5	1	0.11%	0.09%	7.0743	7.0828

Random Forest Classifer - No Sample					
ne	md	accuracy_train	accuracy_test	mc_log_loss_train	mc_log_loss_test
5	2	15.86%	15.96%	3.65646221	3.659814638
5	5	27.68%	26.95%	2.978382498	3.053164438
5	8	40.00%	34.56%	2.384451244	2.761472697
5	12	59.76%	38.04%	1.606248503	3.41803069
5	20	92.52%	36.37%	0.579238351	12.15523472
10	2	18.12%	18.20%	3.644333379	3.650867267
10	5	29.61%	28.75%	2.96623739	3.02175835
10	8	41.84%	36.42%	2.323205605	2.604492179
10	12	66.10%	41.60%	1.479559115	2.77784376
10	20	98.27%	40.45%	0.454570525	8.179245335
50	2	18.86%	18.84%	3.589976148	3.597810412
50	5	31.28%	30.71%	2.873714376	2.930872392
50	8	44.49%	38.78%	2.257638008	2.499522584
50	12	73.36%	44.00%	1.402655152	2.248857407
50	20	99.90%	45.86%	0.436081925	3.345892459
100	2	19.07%	19.01%	3.613359275	3.621705954
100	5	31.19%	30.29%	2.892438585	2.949323645
100	8	44.71%	38.90%	2.251457676	2.494842505
100	12	74.85%	44.65%	1.386887025	2.200414008
100	20	99.89%	47.14%	0.431681969	2.773702755
500	2	19.54%	19.63%	3.595192243	3.60131927
500	5	31.64%	31.01%	2.872620474	2.931047333
500	8	45.10%	39.08%	2.248481384	2.488562245
500	12	74.73%	44.96%	1.386963368	2.169464222
500	20	99.94%	47.77%	0.426680803	2.146094544
1000	2	19.45%	19.47%	3.600660372	3.607777661
1000	5	31.60%	30.81%	2.877222381	2.934210603
1000	8	45.06%	39.14%	2.250669617	2.491227361
1000	12	74.80%	44.99%	1.38519683	2.164146552
1000	20	99.95%	47.58%	0.426608441	2.083879454

Random Forest - SMOTE						
ne	md	syn_ratio	accuracy_train	accuracy_test	mc_log_loss_train	mc_log_loss_test
5	10	0.5	51.89%	25.85%	2.179412467	2.949158335
5	20	0.5	96.69%	64.73%	0.312032372	2.112198461
5	30	0.5	99.66%	71.84%	0.074481757	4.196351346
10	10	0.5	58.78%	31.53%	2.064751102	2.809229298
10	20	0.5	99.04%	74.00%	0.244772965	1.445118484
10	30	0.5	99.97%	80.15%	0.066079415	2.231766901
25	10	0.5	64.01%	33.93%	2.023840689	2.787906274
25	20	0.5	99.49%	77.85%	0.262312275	1.254746625
25	30	0.5	100.00%	85.20%	0.064746901	1.293404519
5	10	1	50.69%	25.97%	2.209165784	2.969920688
5	20	1	96.02%	55.04%	0.346688738	2.746045132
5	30	1	99.42%	59.07%	0.110902087	6.288656555
10	10	1	57.49%	30.40%	2.133689492	2.874051295
10	20	1	98.41%	63.08%	0.314590669	1.784275036
10	30	1	99.96%	69.08%	0.094518857	3.354829784
25	10	1	61.14%	32.27%	2.119691166	2.864756031
25	20	1	99.39%	69.26%	0.294716897	1.517997831
25	30	1	100.00%	76.30%	0.090615809	1.712460275

Random Forest - PFITCC						
ne	md	syn_ratio	accuracy_train	accuracy_test	mc_los_log_train	mc_loss_log_test
5	10	0.5	49.37%	27.95%	2.296690699	2.959694874
5	20	0.5	95.78%	52.38%	0.359809383	3.018893921
5	30	0.5	99.33%	55.01%	0.129170639	7.835474459
10	10	0.5	56.24%	30.19%	2.209740208	2.899273712
10	20	0.5	98.36%	60.48%	0.320298596	1.969066367
10	30	0.5	99.94%	64.29%	0.105409343	4.018578153
25	10	0.5	60.84%	33.27%	2.17405015	2.866985382
25	20	0.5	99.27%	66.91%	0.316588363	1.586020443
25	30	0.5	100.00%	71.55%	0.102343941	1.994630145
5	10	1	45.72%	25.75%	2.454661142	3.068062858
5	20	1	94.64%	43.04%	0.442414416	3.740611706
5	30	1	99.00%	44.04%	0.174626332	10.2676816
10	10	1	51.34%	28.69%	2.431257895	3.030722939
10	20	1	97.70%	50.87%	0.398527537	2.372574461
10	30	1	99.91%	51.54%	0.148396292	6.135746862
25	10	1	56.89%	31.10%	2.361427182	2.969330032
25	20	1	98.98%	55.91%	0.378728422	1.925858317
25	30	1	99.99%	58.70%	0.141907109	3.11784503

Random Forest - PFCC						
ne	md	syn_ratio	accuracy_train	accuracy_test	ks_train	ks_test
5	10	0.5	52.11%	18.60%	2.266635744	3.750206689
5	20	0.5	92.96%	30.24%	0.436304996	10.95185734
5	30	0.5	97.72%	30.16%	0.195417296	16.47959482
10	10	0.5	61.22%	20.19%	2.069884558	3.52627106
10	20	0.5	94.01%	33.75%	0.506758027	5.901364403
10	30	0.5	99.17%	33.48%	0.165254697	12.35572728
25	10	0.5	69.89%	23.27%	1.842882604	3.223602055
25	20	0.5	95.76%	36.04%	0.490112406	3.963745432
25	30	0.5	99.60%	37.29%	0.151962956	8.043519976
5	10	1	33.52%	18.00%	2.818005436	3.710479788
5	20	1	62.08%	27.10%	1.645831955	5.444494388
5	30	1	87.19%	28.08%	0.765767616	11.87629111
10	10	1	37.22%	20.57%	2.775915867	3.489148117
10	20	1	71.11%	29.67%	1.486865664	4.280336517
10	30	1	92.54%	32.20%	0.732354435	7.79340079
25	10	1	41.34%	20.96%	2.68581997	3.414183245
25	20	1	75.71%	31.38%	1.475895377	3.419149224
25	30	1	96.98%	34.35%	0.664771429	5.40673444

References

[1] Chawla, N. V.; Lazarevic, A.; Hall, L. O. & Bowyer, K. W. (2003), SMOTEBoost: Improving Prediction of the Minority Class in Boosting., *in* Nada Lavrac; Dragan Gamberger; Hendrik Blockeel & Ljupco Todorovski, ed., 'PKDD' , Springer, , pp. 107-119 .

[2] Seiffert, C.; Khoshgoftaar, T. M.; Hulse, J. V. & Napolitano, A. (2008), RUSBoost: Improving classification performance when training data is skewed., *in* 'ICPR' , IEEE, , pp. 1-4 .

[3] Dollár, P.; Tu, Z.; Tao, H. & Belongie, S. (2007), Feature Mining for Image Classification., *in* 'CVPR' , IEEE Computer Society, .

[4] Galar, M.; Fernández, A.; Tartas, E. B.; Sola, H. B. & Herrera, F. (2012), 'A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches.', *IEEE Transactions on Systems, Man, and Cybernetics, Part C* **42** (4) , 463-484

[5] AdaBoost M.2: Y.FreundandR.Schapire,"Experimentswithanewboostingalgorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, 1996, pp. 148–156.

[6] Chawla, N. V.; Bowyer, K. W.; Hall, L. O. & Kegelmeyer, W. P. (2002), 'SMOTE: Synthetic Minority Over-sampling Technique',*Journal of Artificial Intelligence Research* **16** , 321--357 .

[7] S. Hu, Y. Liang, L. Ma, and Y. He, "MSMOTE: Improving classification performance when training data is imbalanced," in *Proc. 2nd Int. Workshop Comput. Sci. Eng.*, 2009, vol. 2, pp. 13–17.