Solutions to Binary Imbalanced Classification

Ziyuan Feng
Department of Computer Science and Engineering, Fudan University
zyfeng15@fudan.edu.cn

I. Introduction

This paper studies the binary imbalanced classification problem, presents various approaches to handle data imbalance, and examines the effects of these approaches over three real word datasets. Possible approaches include data-level and algorithm-level strategies. Data-level strategies balance the number of two classes by re-sampling, such as under-sampling, over-sampling, and their combination. Algorithm-level strategies either use ensemble learning with re-sampled subsets of data, or modify the loss function to make the model cost-sensitive to different classes. Besides, there are special methods to handle data imbalance for some particular models, such as SVMs and XGBoost.

II. Problem Overview

In binary classification with imbalance data, one class has far more data (samples, or instances) than the other class. The former is called majority class (or majority). The latter is called minority class (or minority). The class which a sample belong to is called a label. This paper does not distinguish between "classes" or "labels" literally.

Typically, without extra strategies, most classification models will think of any sample being majority class as much as possible, while ignoring mistakes on minority class. This makes the true positive rate (TPR), precision, recall, and F-measure obviously high. However, the model does not learn anything about the minority class or how it differentiates from majority class. Therefore, more attention should be paid on these evaluation indicators: true negative rate (TNR), G-mean (geometric mean of TPR and TNR), and area under receiver operating characteristic curve (AUC).

III. Baseline

In this section, logistic regression (LR), support vector machines (SVM), multilayer perceptrons (MLP), and decision trees (DT) are used as baseline models.

These models follow the default settings from sklearn [1]. Logistic regression uses L2 penalty. SVM uses a linear kernel. MLP uses one hidden layer of size 100, "relu" as an activation function, and the Adam solver with initial learning rate 0.001. Decision trees use "Gini" criteria for splitting and have no restriction for depth or the number of leaves.

Common classifiers like logistic regression and SVM have a bias towards classes which have large number of instances. They tend to only predict the majority class. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class. As shown in Table 1, LR and SVM have high true positive rate, precision, and F-measure, but low true negative rate, G-mean, and AUC.

TABLE I
Baseline models on car dataset.

	LR	SVM	MLP	DT
TNR	0.000	0.000	0.333	0.875
G-mean	0.000	0.000	0.577	0.933
AUC	0.497	0.500	0.666	0.935
FPR	1.000	1.000	0.667	0.125
TPR	0.953	1.000	0.998	0.995
Precision	0.995	0.953	0.969	0.933
Recall	0.953	1.000	0.998	0.995
F-measure	0.974	0.976	0.983	0.994

TABLE II
Baseline models on yeast dataset.

	LR	SVM	MLP	DT
TNR	0.514	0.428	0.714	0.737
G-mean	0.727	0.654	0.845	0.841
AUC	0.932	0.714	0.857	0.837
FPR	0.459	0.571	0.285	0.263
TPR	0.982	1.0	1.0	0.968
Precision	0.946	0.985	0.972	0.968
Recall	0.982	1.0	1.0	0.968
F-measure	0.949	0.972	0.986	0.968

TABLE III
Baseline models on wisconsin dataset.

	LR	SVM	MLP	DT
TNR	0.857	0.857	0.928	0.857
G-mean	0.919	0.922	0.960	0.905
AUC	0.924	0.925	0.960	0.907
FPR	0.142	0.142	0.07	0.142
TPR	0.985	0.992	0.992	0.957
Precision	0.985	0.985	0.992	0.985
Recall	0.985	0.992	0.992	0.957
F-measure	0.985	0.989	0.992	0.971

Compared to LR and SVM, MLP and DT do better in treating the minority class. In the following sections, linear regression is used to evaluate different methods of improving performance on minority class. A good method is expected to increase TNR, G-mean, and AUC of a logistic regression classifier.

IV. Data-level Strategies

The main objective of balancing classes is to either increase the number of minority or decrease the number of majority, in order to obtain the approximately the same number of instances of both classes.

A. Under-sampling

1) Random: This strategy randomly eliminates some instances of majority class in the training data. It could improve the running time and storage when dealing with large datasets. However, it suffers from severe information loss because the under-sampled data may be the inaccurate and biased representation of the original data.

2) Condensed nearest-neighbor(CNN): Under-sampling can be viewed as a process of finding representatives of the majority data. Thus, nearest-neighbor method could be useful to determine which sample to be picked. Condensed nearest-neighbor method [2] constructs a subset S of the original dataset T, such that for each sample in T, its nearest neighbor of the same class can be found in S. The subset S can grow iteratively, given by the following algorithm.

procedure CONDENSED NEAREST-NEIGHBOR(T)
  randomly select a sample u from T
  $S \leftarrow \{u\}$
  while S not converge **do**
    for each v in $T - S$ **do**
      $n \leftarrow FindNearestNeighbor(v, S)$ ▷ Find the nearest neighbor of v in S
      if n is of a different class from v **then**
        add v to S
      end if
    end for
  end while
  return S
end procedure

3) Edited nearest-neighbor(ENN) : Another nearest neighbor algorithm considers more than one neighbors. In edited nearest-neighbor [3], a sample is removed whose label differs from the majority of its k neighbors, where k is a hyper-parameter.

Repeating the algorithm multiple times will delete more data. This gives the repeated edited nearest-neighbor (RENN) under-sampling method.

4) Tomek's links: A Tomek's link [4] refers to a pair of samples from different labels which are each other's nearest neighbor. Under-sampling can be done by removing all Tomek's links, or just the samples from majority class of the links.

This method, however, will change the distribution of majority class, resulting in over-fitting on the training set and low performance on the test set.

5) One-side-selection(OSS): This algorithm is an extension of Tomek's link.

procedure ONE-SIDE-SELECTION(T)
  $S \leftarrow \emptyset$
  for each v in T **do**
    $n \leftarrow FindNearestNeighbor(v, T)$
    if n is of a different class from v **then**
      add v to S ▷ (v, n) is a Tomek's link.
    end if
  end for
  return S
end procedure

One side selection method has the same problem as Tomek's link.

6) NearMiss: NearMiss [5] is a family of under-sampling methods based on distance calculation.

In NearMiss-1, the samples of the majority class are retained whose average distance to k nearest samples from the minority class is smallest. In NearMiss-2, the samples from majority are retained whose average distance to k farthest samples of minority class is smallest. In NearMiss-3, k nearest neighbors from majority class are selected for each sample in minority class. In three cases, k is a hyper-parameter.

B. Over-sampling

1) Random: Contract to the previous ones, over-sampling randomly replicates instances of minority class. This can be done without information loss, but may lead to over-fitting.

2) Subclass-based: This method assumes each class contains several subclasses, and randomly over-samples all subclasses until all subclasses within the same class have the same number of observations. However, subclasses may be implicit, or such assumption does not necessarily hold for a specific problem. In fact, K-means clustering algorithm is applied to discover subclasses.

3) Synthetic minority over-sampling technique(SMOTE): Instead of replicating minority instances, SMOTE [6] generates similar synthetic instances using a subset of the minority class. SMOTE helps prevent over-fitting, but possibly introduces noise from new instances that fall into regions overlapped with majority class.

procedure SMOTE(M) ▷ M is the minority set
  $S \leftarrow \emptyset$
  for each v in M **do**
    $n_1, \cdots, n_k \leftarrow FindNearestNeighbor(v, M, k)$ ▷ Find the k nearest neighbors of v in M
    randomly choose $r \leq k$ neighbors $n_{i_1}, \cdots, n_{i_r}$
    for each p in $n_{i_1}, \cdots, n_{i_r}$ **do**
      randomly choose α in [0, 1]
      $u \leftarrow \alpha v + (1 - \alpha)n$
      add u to S
    end for
  end for
  return $S \bigcup M$
end procedure

4) Adaptive synthetic (ADASYN): ADASYN [7] uses a density distribution as a criterion, which is a measurement for minority class samples according to the level of difficulty in learning, to automatically decide the number of synthetic samples that need to be generated. More synthetic samples will be generated for samples that are hard to learn than the ones that are easy to learn.

C. Combining under- & over-sampling

Combining over-sampling on minority class and under-sampling on majority class may yield better results than using

only one. Two common combinations are SMOTE+ENN and SMOTE+Tomek's link.

V. Algorith-level Strategies

An alternative approach to handle imbalanced classification is to modify existing models and make them capable for imbalanced datasets. The following sections discuss ensemble methods and model-specific methods.

A. Ensemble methods

Rather than build one classifier on one re-sampled dataset, ensemble methods combine different classifiers built on different re-sampled datasets.

1) Easy Ensemble: In Easy Ensemble [8], a sequence of classifiers are built by sampling subsets from the majority class such that the subset contains the same number of samples as the minority class. In fact, this is the AdaBoost process.

1: **procedure** EASY-ENSEMBLE(P, Q) ▷ P is the minority class, Q is the majority class
2: **for** $i = 1, \cdots, N$ **do**
3: randomly sample a subset Q_i from Q such that $|Q_i| = |P|$
4: build an AdaBoost classifier $F_i(x)$ using $Q_i \bigcup P$
5: **end for**
6: combine $F_i(x), i = 1, \cdots, N$ to obtain an ensemble classifier
7: **end procedure**

2) Balance Cascade: Balance Cascade [8] is a modification of Easy Ensemble in which the misclassified samples are selected again in the next iteration.

1: **procedure** BALANCE-CASCADE(P, Q) ▷ P is the minority class, Q is the majority class
2: **for** $i = 1, \cdots, N$ **do**
3: randomly sample a subset Q_i from Q such that $|Q_i| = |P|$
4: build an AdaBoost classifier $F_i(x)$ using $Q_i \bigcup P$
5: remove samples in Q correctly classified by $F_i(x)$
6: **end for**
7: combine $F_i(x), i = 1, \cdots, N$ to obtain an ensemble classifier
8: **end procedure**

3) Bagging: With the framework provided by Easy Ensemble, bagging can be applied over all re-sampled datasets. The only change to the algorithm is in Line 4 "build a base classifier $F_i(x)$ using $Q_i \bigcup P$".

4) Gradient boosting decision trees: As an ensemble method, gradient boosting decision trees (GBDT) build an additive model iteratively. At each step, a decision tree is chosen as a weak learner to minimize the current loss given by the model built in previous steps. The minimization is solved numerically.

B. Weighted loss function

One way to force the algorithm to pay more attention to the minority class is to modify the loss function so that misclassifying samples from the minority class will be highly penalized, which is called cost-sensitive approaches in [9].

Formally, let x_i and y_i be the features and label of the i-th sample, θ be the model parameters. The posterior probability $P(y_i|x_i, \theta)$ shall be maximized. Let $f(.)$ be a monotonically increasing function with respect to $P(y_i|x_i, \theta)$. Then the loss function (without regularization, same as below) for binary classification can be defined as

$$L(\theta) = -\sum_{j=0}^{1} \sum_{y_i=j} f(P(y_i|x_i, \theta)) \tag{1}$$

Let w_j be the weight corresponding to class j. The weighted loss function can be defined as

$$L(\theta) = -\sum_{j=0}^{1} w_j \sum_{y_i=j} f(P(y_i|x_i, \theta)) \tag{2}$$

In this case, the model will be heavily punished when making mistakes on class 1 as long as $w_1 > w_0 > 0$.

In scikit-learn [1], the balancing weights are often calculated as a number inversely proportional to class frequencies in the training data. For example, the weight corresponding to class j is

$$w_j = \frac{N}{C \times |\{y_i = j | i = 1 \cdots N\}|} \tag{3}$$

where N is the total number of samples, C is the number of classes. In binary classification, $C = 2$.

Both Logistic Regression(LR) and multi-layer perceptron(MLP) classifiers use cross-entropy loss in scikit-learn [1]. The weighted loss function is

$$L(\theta) = -\sum_{j=0}^{1} w_j \sum_{y_i=j} \log(P(y_i = 1|x_i, \theta)) \tag{4}$$

In Support Vector Machine(SVM) with hinge loss, the weighted loss function is

$$L(\theta) = -\sum_{j=0}^{1} w_j \sum_{y_i=j} y_i \max\{0, 1+\theta^T x_i\} + (1-y_i) \max\{0, 1-\theta^T x_i\} \tag{5}$$

C. Model-specific methods

Some solutions to imbalanced classification are model-based, which means they cannot generalize to other models or algorithms.

1) kernel-based methods: Instead of balancing the data or improving algorithms, the use of kernels in SVMs optimizes features to make the problem more "classifiable". A kernel function is used to map the linear non-separable space into a higher dimensional space where separation is more likely to be achievable.

Integrating kernel methods with sampling gives the Granular Support Vector Machines Repetitive Under-sampling algorithm (GSVM-RU) [10], which can effectively analyze the inherent data distribution by observing the trade-offs between the local significance of a subset of data and its global correlation.

2) tree-based methods: XGBoost [11] is an advanced gradient boosting decision tree framework for classification. There are two ways to handle imbalance in XGBoost. One is to restrict the complexity of the trees by capping the maximum weight of a regression tree. The weight of a decision tree could be defined differently. For example, it is the weighted sum of the number of leaves and the L2 norm of leaf scores.

The other way is to control the balance of positive and negative weights explicitly. Since XGBoost is optimized using an objective function, it is easy to modify the objective to make the model cost-sensitive.

VI. EXPERIMENTS

TABLE IV
results on cat dataset: minority ratio=3.76%

	baseline	under-sampling								over-sampling			
	LR	random	CNN	ENN	TL	OSS	NM-1	NM-2	NM-3	random	subclass	SMOTE	ADASYN
TNR	0.0	0.196	0.318	**0.353**	0.0	0.0	0.061	0.177	0.093	0.232	0.232	0.240	0.234
G-mean	0.0	0.440	**0.553**	0.510	0.0	0.0	0.242	0.420	0.296	0.481	0.481	0.489	0.483
AUC	0.497	0.940	**0.951**	**0.951**	0.497	0.497	0.656	0.931	0.808	0.952	0.952	0.952	0.952
F-measure	0.974	0.912	0.972	0.977	0.974	0.974	0.761	0.905	0.868	0.932	0.932	0.935	0.933

	under- & over-sampling		ensemble				weighted loss			model-specific	
	SMOTE+ENN	SMOTE+TL	EE	BC	bagging	GBDT	LR	SVM	DT	Gaussian SVM	XGBoost
TNR	0.229	0.242	0.875	0.874	0.833	**0.983**	0.230	0.214	0.986	0.980	0.973
G-mean	0.478	0.491	0.933	0.931	0.912	**0.991**	0.479	0.462	0.992	0.985	0.986
AUC	0.951	0.952	0.935	0.935	0.916	**1.000**	0.952	0.950	0.977	0.998	**1.000**
F-measure	0.931	0.936	0.994	0.994	0.995	**0.999**	0.932	0.922	0.999	0.995	**0.999**

TABLE V
results on yeast dataset: minority ratio=9.92%

	baseline	under-sampling								over-sampling			
	LR	random	CNN	ENN	TL	OSS	NM-1	NM-2	NM-3	random	subclass	SMOTE	ADASYN
TNR	0.0	0.460	0.624	0.0	0.0	0.0	0.510	0.808	**0.698**	0.571	0.572	0.586	0.455
G-mean	0.0	0.670	0.670	0.0	0.0	0.0	0.704	**0.886**	0.827	0.747	0.798	0.757	0.671
AUC	0.5	0.925	0.914	0.5	0.5	0.5	0.903	0.932	0.922	0.935	0.935	0.937	**0.949**
F-measure	0.952	0.933	0.959	0.952	0.952	0.952	0.943	0.977	0.971	0.954	0.955	0.957	0.930

	under- & over-sampling		ensemble				weighted loss			model-specific	
	SMOTE+ENN	SMOTE+TL	EE	BC	bagging	GBDT	LR	SVM	DT	Gaussian SVM	XGBoost
TNR	0.520	0.573	**0.928**	0.928	0.857	0.778	0.857	0.857	0.714	0.0	0.750
G-mean	0.714	0.749	**0.960**	0.957	0.919	0.863	0.875	0.888	0.836	0.0	0.846
AUC	0.938	0.939	0.960	0.959	0.921	**0.980**	0.875	0.889	0.846	0.5	0.967
F-measure	0.945	0.954	0.992	0.992	0.985	0.970	0.936	0.952	0.975	0.952	0.966

TABLE VI
results on wisconsin dataset: minority ratio=34.97%

	baseline	under-sampling								over-sampling			
	LR	random	CNN	ENN	TL	OSS	NM-1	NM-2	NM-3	random	subclass	SMOTE	ADASYN
TNR	0.857	0.951	0.838	0.940	0.950	0.920	0.951	**0.958**	0.926	0.947	0.944	0.951	0.941
G-mean	0.919	0.966	0.914	0.961	0.962	0.954	0.963	0.965	0.959	0.964	0.962	0.964	**0.966**
AUC	0.924	0.993	0.994	0.995	0.994	0.994	0.994	0.994	0.994	0.994	0.993	0.994	**0.995**
F-measure	0.985	0.977	0.941	0.974	0,973	0.971	0.974	0.974	0.975	0.976	0.974	0.975	0.979

	under- & over-sampling		ensemble				weighted loss			model-specific	
	SMOTE+ENN	SMOTE+TL	EE	BC	bagging	GBDT	LR	SVM	DT	Gaussian SVM	XGBoost
TNR	0.936	0.951	0.956	0.956	0.927	**0.971**	0.951	0.947	0.932	0.915	0.940
G-mean	0.958	0.966	0.963	0.963	0.952	**0.974**	0.968	0.965	0.942	0.954	0.959
AUC	0.994	0.994	0.963	0.963	0.952	0.974	**0.995**	**0.995**	0.938	0.988	0.989
F-measure	0.972	0.977	0.974	0.974	0.970	0.981	0.979	0.977	0.959	0.972	0.973

A. Settings

Experiments are conducted over three imbalanced datasets, car, yeast, and wisconsin, with different minority ratios. The machine learning framework used in the experiment is scikit-learn [1] and most of the strategies are implemented in imbalanced-learn [12]. The default parameter settings of these models are used in the experiment. What's more, 5-fold validation is applied and all evaluation indicators are averaged over 5 folds.

In edited nearest neighbor, NearMiss family, and SMOTE, k is set as 3.

In XGBoost, weights of the minority class is scaled corresponding to the minority ratio of the dataset. For example, this value is 10 in the car dataset. Moreover, the maximum depth of each tree is 2 and XGBoost is set to optimize AUC. Another setting of parameters to control imbalance is the maximum weight update. The one which yields better TPR results is chosen.

B. Result analysis

Table IV, V, and VI show experiment results on car, yeast, and wisconsin datasets respectively. Since the minority ratios of different datasets vary a lot, the best method on one dataset may not be the best on another.

Gradient boosting decision trees (GBDT) achieve highest true negative rate on car and wisconsin datasets, but performs worse on yeast. One possible reason is that the features in both car and wisconsin are categorical, but those in yeast are numerical. GBDT uses decision trees as base learners which handles categorical features better than continuous features. This is because the splitting criteria of a decision tree can directly considered as some combinations of categorical features, while continuous values have to be encoded somehow to be represented in a tree.

In the cat dataset where data imbalance is extreme, condensed nearest neighbors and edited nearest neighbors achieve better performance than any other under-sampling methods. Among over-sampling methods, SMOTE leads to the best result. Combining SMOTE with Tomek's link results in slightly better performance than single SMOTE. However, combining SMOTE with ENN gets worse results than either of them. Possible explanation is that performing ENN to under-sample after over-sampling with SMOTE change the distribution of the original dataset.

Tomek's link (TL) and one-side-selection (OSS) have no effect in the car and yeast datasets because they clean up too many majority samples near a minority sample so that the information of the decision boundary is lost in small datasets. In the wisconsin dataset, TL and OSS work. Therefore, TL and OSS should be performed in imbalanced datasets of which minority ratio is not too small.

Ensemble methods yield significantly better results than sampling strategies in all three datasets. They perform the best in yeast. Among ensemble methods, Easy Ensemble and Balance Cascade, which use AdaBoost, outperform bagging, but lose to GBDT.

Weighted loss function also work in three datasets. For logistic regression, support vector machines, and decision trees, there are significant improvements after using class weights, which can be seen by comparing Table I-III and IV-VI respectively.

The Gaussian SVM shows extraordinary performance in the car dataset. By simply adding a Gaussian kernel and keeping everything else the same as before, SVM achieves a nearly perfect solution. However, such effect may not be held in other datasets. In the yeast dataset, adding a Gaussian kernel has no effect at all. This may be caused by different distribution of datasets in high dimension space.

VII. Conclusion

This paper explores various strategies to solve binary classification problems in imbalanced datasets. For data-level strategies, either under-sampling or over-sampling is performed to balance the size of two classes. Under-sampling finds and keeps only the representatives of the majority class in the original datasets and removes the other majority samples. Over-sampling generates new samples based on original data distribution in the minority class. Combining both is also a strategy, but does not yield better results in the experiment. For algorithm-level strategies, ensemble models combine weak classifiers built on balanced re-sampled subsets to alleviate the preference towards majority class. Weighted loss functions yields cost-sensitive models which respond differently to two classes. Gradient boosting decision trees give best results in two of three datasets. Algorithm-level strategies usually outperform data-level strategies in a large scale. They should be the prior choice to tackle binary imbalance classification problems. In addition, SVMs with a Gaussian kernel outperforms those with a linear kernel in particular imbalance datasets. XGBoost is able to improve slightly by fine-tuning its parameters related to weight scaling and boosting steps.

In conclusion, solutions to binary classification on imbalanced datasets requires exploration to the data distribution and a lot of efforts in model selection.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] P. Hart, "The condensed nearest neighbor rule (corresp.)," *IEEE transactions on information theory*, vol. 14, no. 3, pp. 515–516, 1968.

[3] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 408–421, 1972.

[4] I. Tomek, "Two modifications of cnn," *IEEE Trans. Systems, Man and Cybernetics*, vol. 6, pp. 769–772, 1976.

[5] I. Mani and I. Zhang, "knn approach to unbalanced data distributions: a case study involving information extraction," in *Proceedings of workshop on learning from imbalanced datasets*, vol. 126, 2003.

[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[7] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 1322–1328.

[8] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2009.

[9] Z.-H. Zhou and X.-Y. Liu, "On multi-class cost-sensitive learning," *Computational Intelligence*, vol. 26, no. 3, pp. 232–257, 2010.

[10] Y. Tang and Y.-Q. Zhang, "Granular svm with repetitive undersampling for highly imbalanced protein homology prediction," in *Granular Computing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 457–460.

[11] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[12] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

APPENDIX

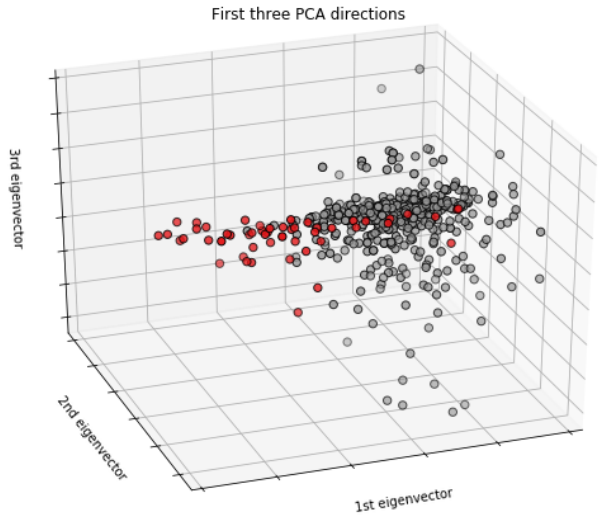

Fig. 1: Visualization of yeast dataset after PCA. Red dots are minority class.

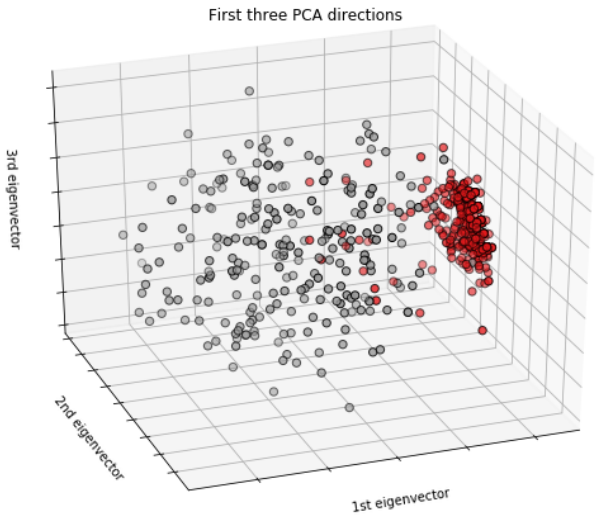

Fig. 2: Visualization of wisconsin dataset after PCA. Red dots are minority class.

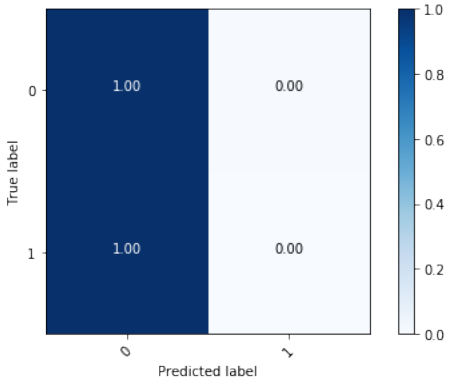

Fig. 3: Confusion matrix of logistic regression on car dataset. This is the baseline model.

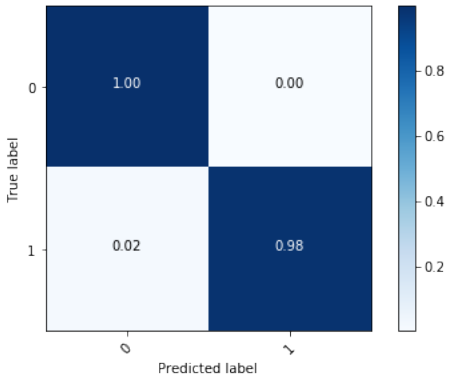

Fig. 4: Confusion matrix of GBDT on car dataset. This is the best model on the dataset.

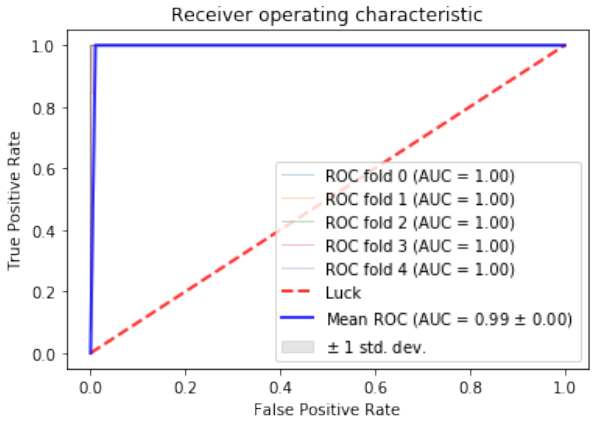

Fig. 5: The AUC curve of GBDT on car dataset. This is the best model on the dataset.

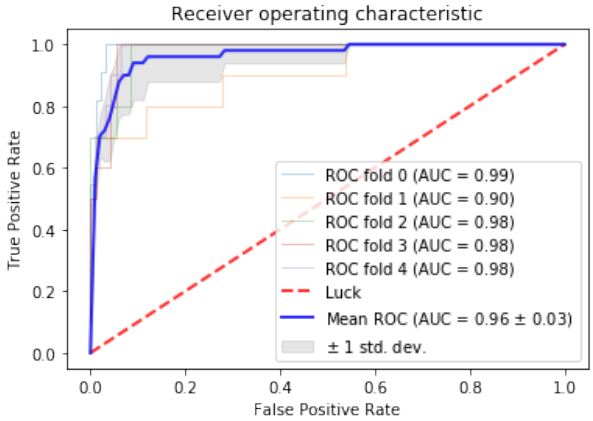

Fig. 6: The AUC curve of XGBoost on yeast dataset.