Sampling Techniques for Handling Imbalanced Data

The Team

Sarah Nooravi
Marketing Analyst at MobilityWare
Linkedin: https://www.linkedin.com/in/snooravi/

Eric Weber
Principal, Data Management at Corelogic
Linkedin: https://www.linkedin.com/in/eric-weber-060397b7/

MobilityWare - We are Hiring!

Outline

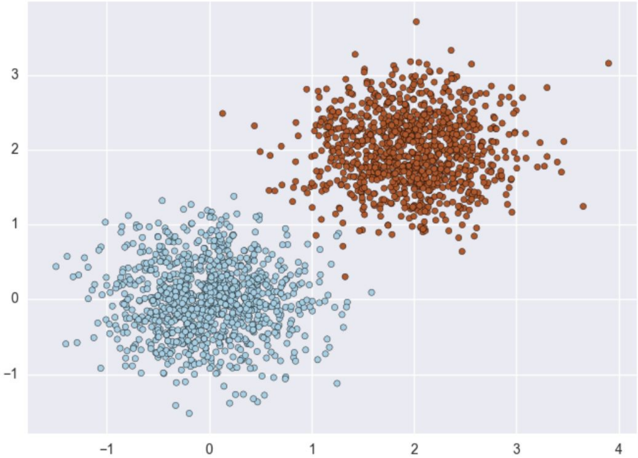
- Defining Imbalance
- Why is Imbalance an Issue
- Possible Remedies
- SMOTE
- Demo!

1

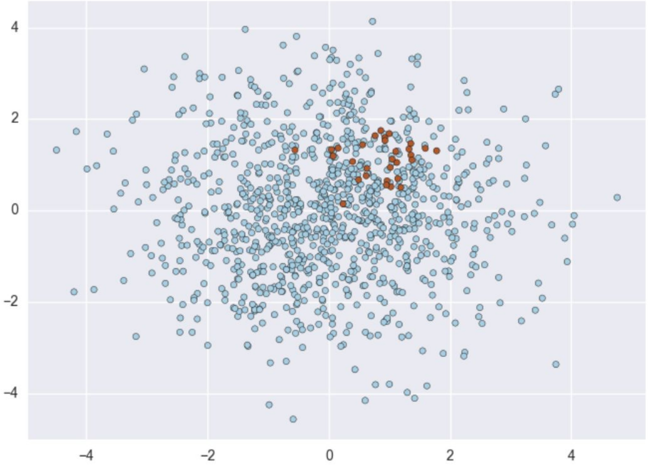
Defining Imbalance

A dataset is said to be imbalanced when the binomial (or multinomial) response variable has one or more classes that are underrepresented in the training data with respect to the rest of the classes.

Example of Balanced and Imbalanced Data



Balanced
Negatives ~ Positives



Imbalanced
Negatives > Positives

By convention:

- **Positive Class**: the class with fewer samples
- **Negative Class**: class with majority samples

Real World Examples of highly Imbalanced Datasets

2

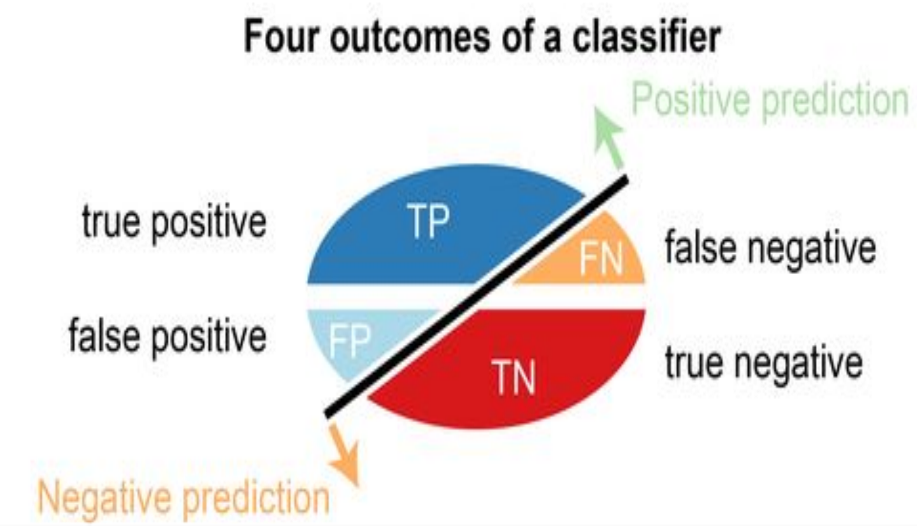
Why is Imbalance an Issue

When Accuracy Fails as an Evaluation Metric

Conventional algorithms are often biased towards the **majority class** because their loss functions attempt to optimize quantities such as **error rate**, not taking into account the class distributions.

Suppose you have two classes—A and B. Class A is 90% of your data-set and class B is the other 10%, but you are most interested in identifying instances of class B. You can reach an accuracy of 90% by simply predicting class A every time, but this provides a useless classifier for your intended use case.

TPR, FPR, Precision, Recall



$$\text{TPR (sensitivity)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

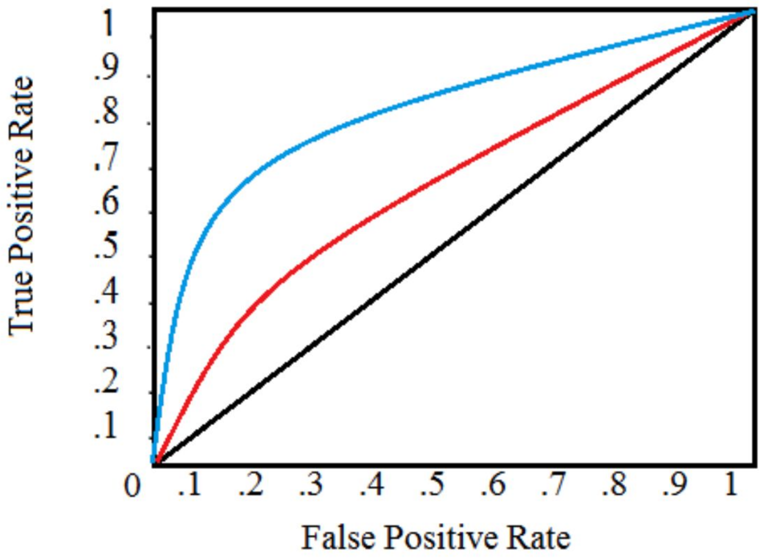
$$\text{FPR (1-specificity)} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

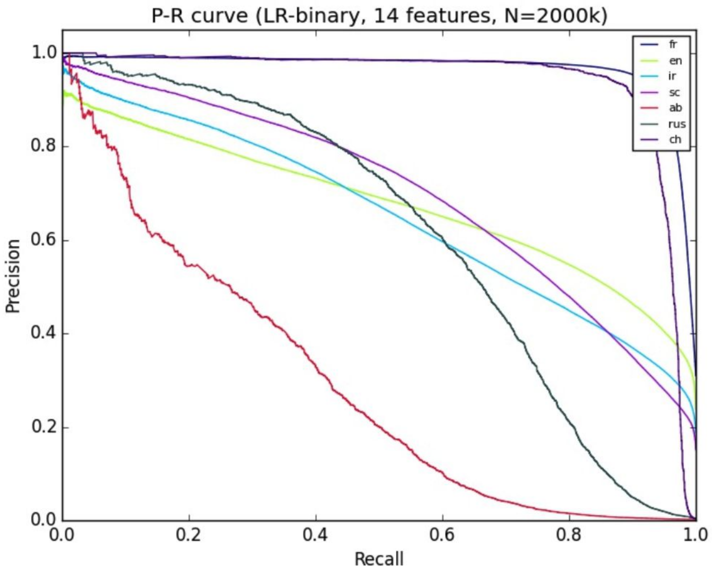
$$\text{Precision} = \frac{TP}{TP + FP}$$

ROC and Precision-Recall Curve

ROC: Receiver Operating Curve



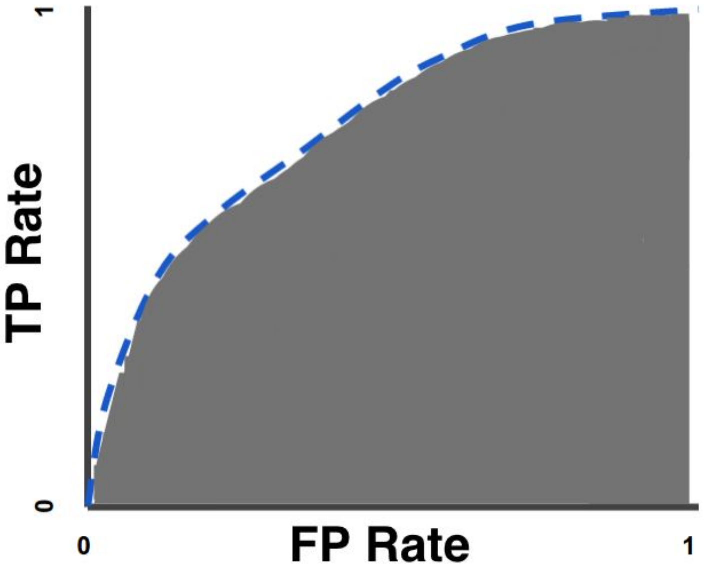
Precision-Recall Curve



Single Number Metrics

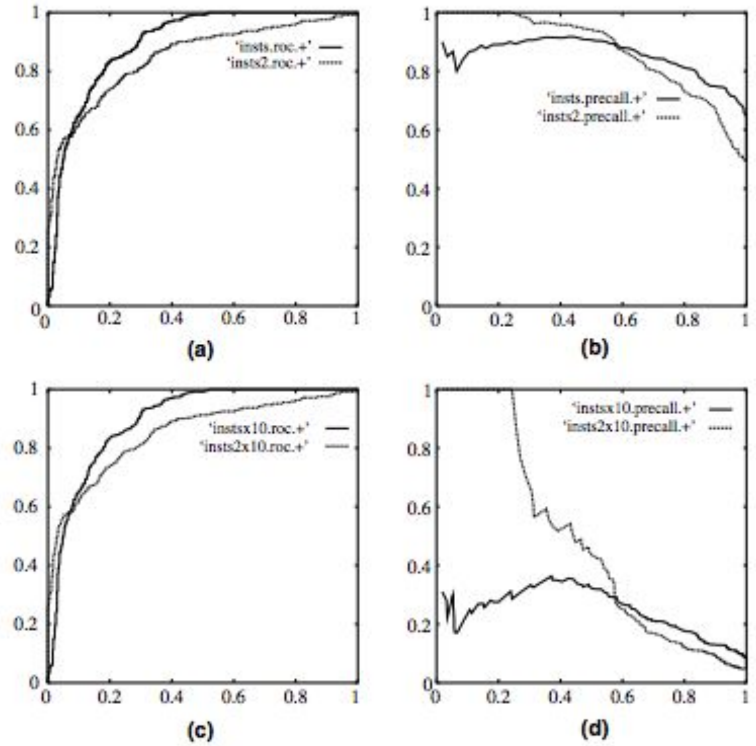
- **AUC**: Area Under the ROC Curve
- **F1 Score**: the harmonic mean of precision and recall

AUC: Area Under the Curve



ROC Resistance to Skewness

Fig. 5 ROC and precision-recall curves under the class skew. (a) ROC Curves, 1:1, (b) precision-recall curves, 1:1; (c) ROC curves, 1:10 and (d) precision recall curves, 1:10.

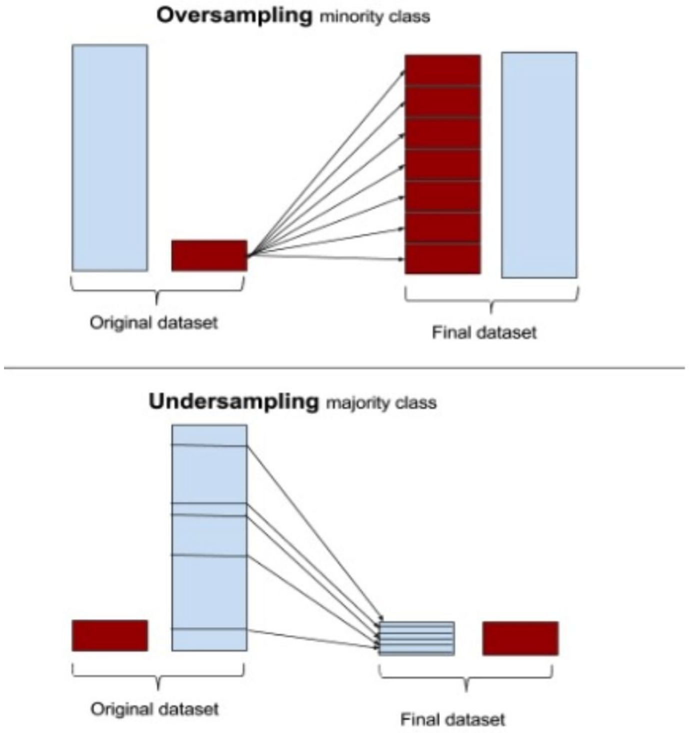


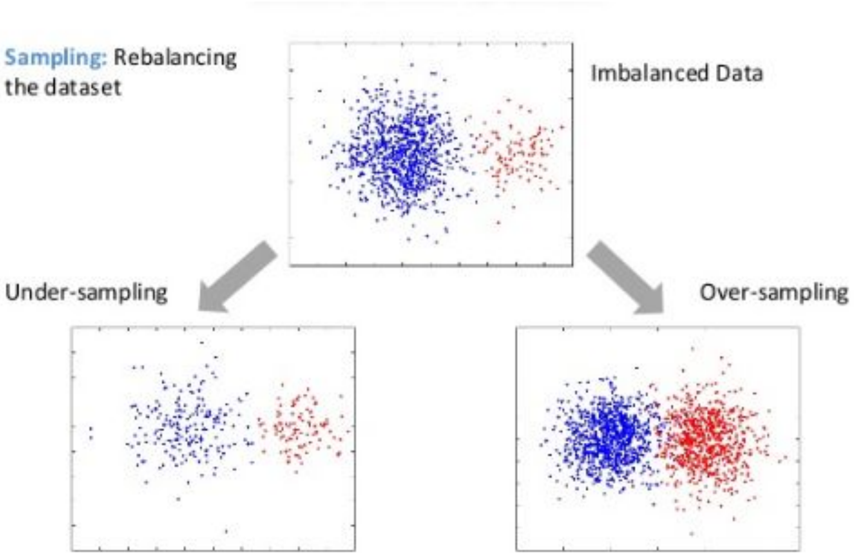
3 Possible Remedies

Possible Remedies

- Do nothing--you got lucky!
- Balance the training set using sampling techniques:
  - Random/Focused Oversampling
  - Random/Focused Undersampling
  - **Synthesizing new examples: SMOTE**
  - Many more...
- Cost-Sensitive Learning
- Anomaly Detection
- Buy or create more data

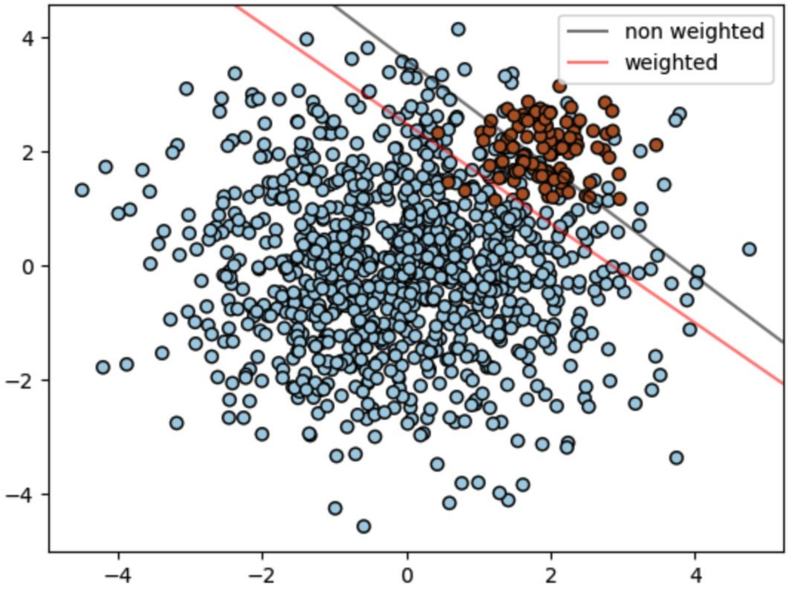
Over/Under Sampling





Class Weights

- Increasing the “importance” of classes




```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# we create clusters with 1000 and 100 points
rng = np.random.RandomState(0)
n_samples_1 = 1000
n_samples_2 = 100
X = np.r_[1.5 * rng.randn(n_samples_1, 2),
          0.5 * rng.randn(n_samples_2, 2) + [2, 2]]
y = [0] * (n_samples_1) + [1] * (n_samples_2)

# fit the model and get the separating hyperplane
clf = svm.SVC(kernel='linear', C=1.0)
clf.fit(X, y)

# fit the model and get the separating hyperplane using weighted classes
wclf = svm.SVC(kernel='linear', class_weight={1: 10})
wclf.fit(X, y)

# plot separating hyperplanes and samples
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolors='k')
plt.legend()

# plot the decision functions for both classifiers
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T

# get the separating hyperplane
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
a = ax.contour(XX, YY, Z, colors='k', levels=[0], alpha=0.5, linestyles=['-'])

# get the separating hyperplane for weighted classes
Z = wclf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins for weighted classes
b = ax.contour(XX, YY, Z, colors='r', levels=[0], alpha=0.5, linestyles=['-'])

plt.legend([a.collections[0], b.collections[0]], ["non weighted", "weighted"],
           loc="upper right")
plt.show()
```

4

Synthetic Minority Oversampling Technique (SMOTE)

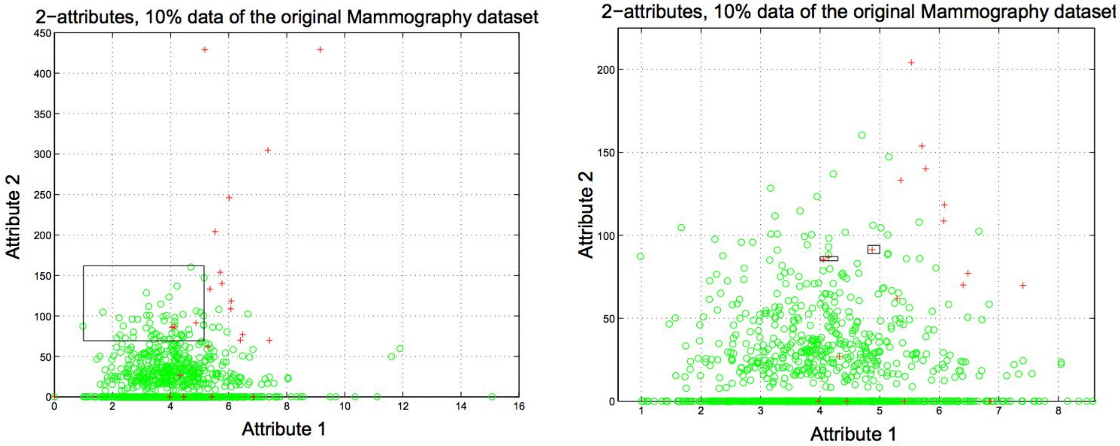
Summary of Literature To-Date

- Under-sampling the majority class enables better classifiers to be built than over-sampling the minority class. A combination of the two as done in previous work does not lead to classifiers that outperform those built utilizing only undersampling.

- **However, the over-sampling of the minority class has been done by sampling with replacement from the original data.**

- SMOTE uses a different variation of over-sampling.

Disadvantages of Prior Methods



(a) (b)

Figure 3: a) Decision region in which the three minority class samples (shown by '+') reside after building a decision tree. This decision region is indicated by the solid-line rectangle. b) A zoomed-in view of the chosen minority class samples for the same dataset. Small solid-line rectangles show the decision regions as a result of over-sampling the minority class with replication. c) A zoomed-in view of the chosen minority class samples for the same dataset. Dashed lines show the decision region after over-sampling the minority class with synthetic generation.

Randomly oversampling the minority class...

- Did not generalize very well (prone to overfitting)
- Produced very specific decision boundaries

We propose an oversampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement

“

Journal of Artificial Intelligence Research 16 (2002) 321–357 Submitted 09/01; published 06/02

SMOTE: Synthetic Minority Over-sampling Technique

Nitesh V. Chawla CHAWLA@CSEE.USF.EDU
Department of Computer Science and Engineering, ENB 118
University of South Florida
4202 E. Fowler Ave.
Tampa, FL 33620-5399, USA

Kevin W. Bowyer KWB@CSE.ND.EDU
Department of Computer Science and Engineering
384 Fitzpatrick Hall
University of Notre Dame
Notre Dame, IN 46556, USA

Lawrence O. Hall HALL@CSEE.USF.EDU
Department of Computer Science and Engineering, ENB 118
University of South Florida
4202 E. Fowler Ave.
Tampa, FL 33620-5399, USA

W. Philip Kegelmeyer WPK@CALIFORNIA.SANDIA.GOV
Sandia National Laboratories
Biosystems Research Department, P.O. Box 969, MS 9951
Livermore, CA, 94551-0969, USA

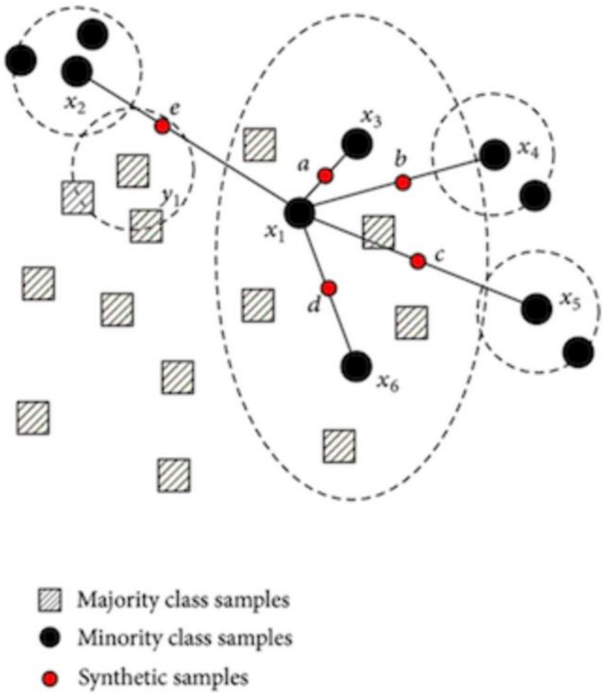
SMOTE - Motivation

- This approach is inspired by a technique that proved successful in handwritten character recognition (Ha & Bunke, 1997).
- They created extra training data by performing certain operations on real data. In their case, operations like rotation and skew were natural ways to perturb the training data.

SMOTE

- The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors.

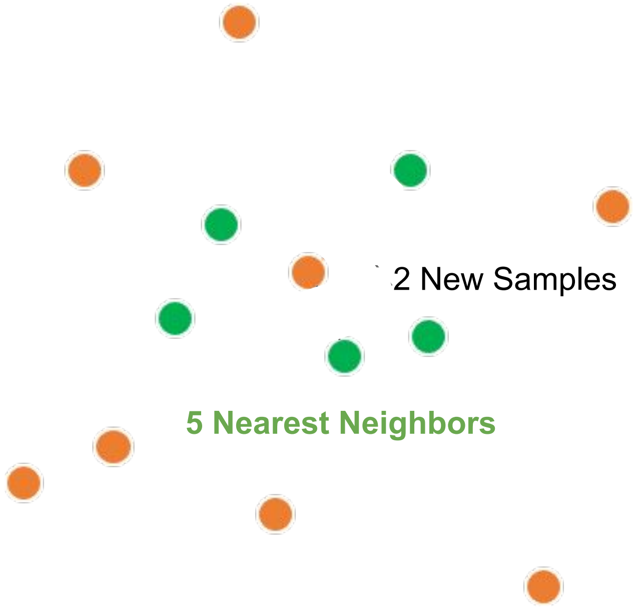
- Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen.



Visualization of SMOTE

SMOTE - Visual Example

- For instance, if the amount of over-sampling needed is 200%, only **two** neighbors from the **five** nearest neighbors are chosen and one sample is generated in the direction of each.



SMOTE - Method

- Synthetic samples are generated in the following way:
  - Take the difference between the feature vector (sample) under consideration and its nearest neighbor.
  - Multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration.

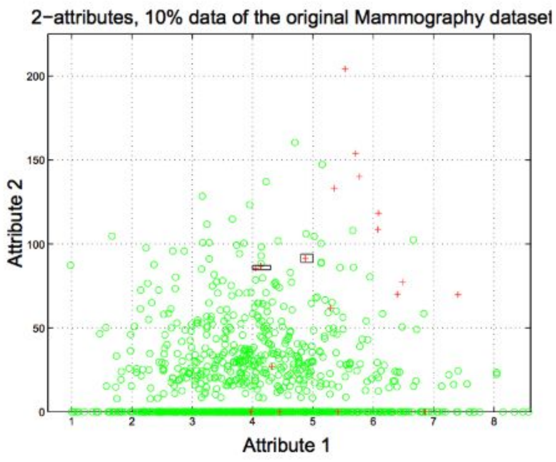
Example

Consider a sample (6,4) and let (4,3) be its nearest neighbor.
(6,4) is the sample for which k-nearest neighbors are being identified.
(4,3) is one of its k-nearest neighbors.
Let:
$f1\_1 = 6 \quad f2\_1 = 4 \quad f2\_1 - f1\_1 = -2$
$f1\_2 = 4 \quad f2\_2 = 3 \quad f2\_2 - f1\_2 = -1$
The new samples will be generated as
$(f1', f2') = (6,4) + rand(0\text{-}1) * (-2,-1)$
rand(0-1) generates a random number between 0 and 1.

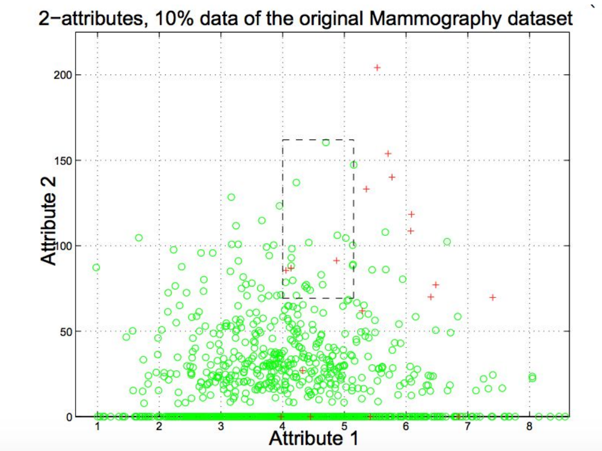
Table 1: Example of generation of synthetic examples (SMOTE).

SMOTE - Advantage

- The synthetic examples cause the classifier to create larger and less specific decision regions.
- More general regions are now learned for the minority class samples. The effect is that decision trees generalize better.



(b)



Things to keep in mind

No matter what you do for training, always test on the natural (stratified) distribution your classifier is going to operate upon.

- For Python: See `sklearn.cross_validation.StratifiedKFold`

Python Library for Implementation

In R:

- SMOTE and variants are available in R in the `unbalanced` package and in Python in the [`UnbalancedDataset`] package.
- The R package `unbalanced` implements a number of sampling techniques specific to imbalanced datasets

In Python:

- `scikit-learn.cross_validation` has basic sampling algorithms.
- Imbalanced-learn:
  - Documentation: Here

DEMO

Credit Card Fraud Detection

Content

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Pseudo Code

Algorithm *SMOTE*(T, N, k)

Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k

Output: $(N/100) * T$ synthetic minority class samples

(∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
if $N < 100$
    then Randomize the T minority class samples
        $T = (N/100) * T$
        $N = 100$
endif
$N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
k = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0
$Synthetic[\,][\,]$: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)

Pseudo Code

```
for i ← 1 to T
        Compute k nearest neighbors for i, and save the indices in the nnarray
        Populate(N, i, nnarray)
endfor

Populate(N, i, nnarray) (* Function to generate the synthetic samples. *)
while N ≠ 0
        Choose a random number between 1 and k, call it nn. This step chooses one of
        the k nearest neighbors of i.
        for attr ← 1 to numattrs
                Compute: dif = Sample[nnarray[nn]][attr] − Sample[i][attr]
                Compute: gap = random number between 0 and 1
                Synthetic[newindex][attr] = Sample[i][attr] + gap ∗ dif
        endfor
        newindex++
        N = N − 1
endwhile
return (* End of Populate. *)
End of Pseudo-Code.
```

Resources

An Introduction to ROC Analysis: Here

Original Paper: Here

Handling Imbalanced Datasets: Here

Learning from Imbalanced Classes: Here

SVM Scikit learn: Here

Top 10 Data Science Practitioner Pitfalls: Here

Classification ROC and AUC: Here

Visualizing Confusion Matrix: Here

Great Visual for understanding ROC: Here

Kaggle Fraud Detection: Here