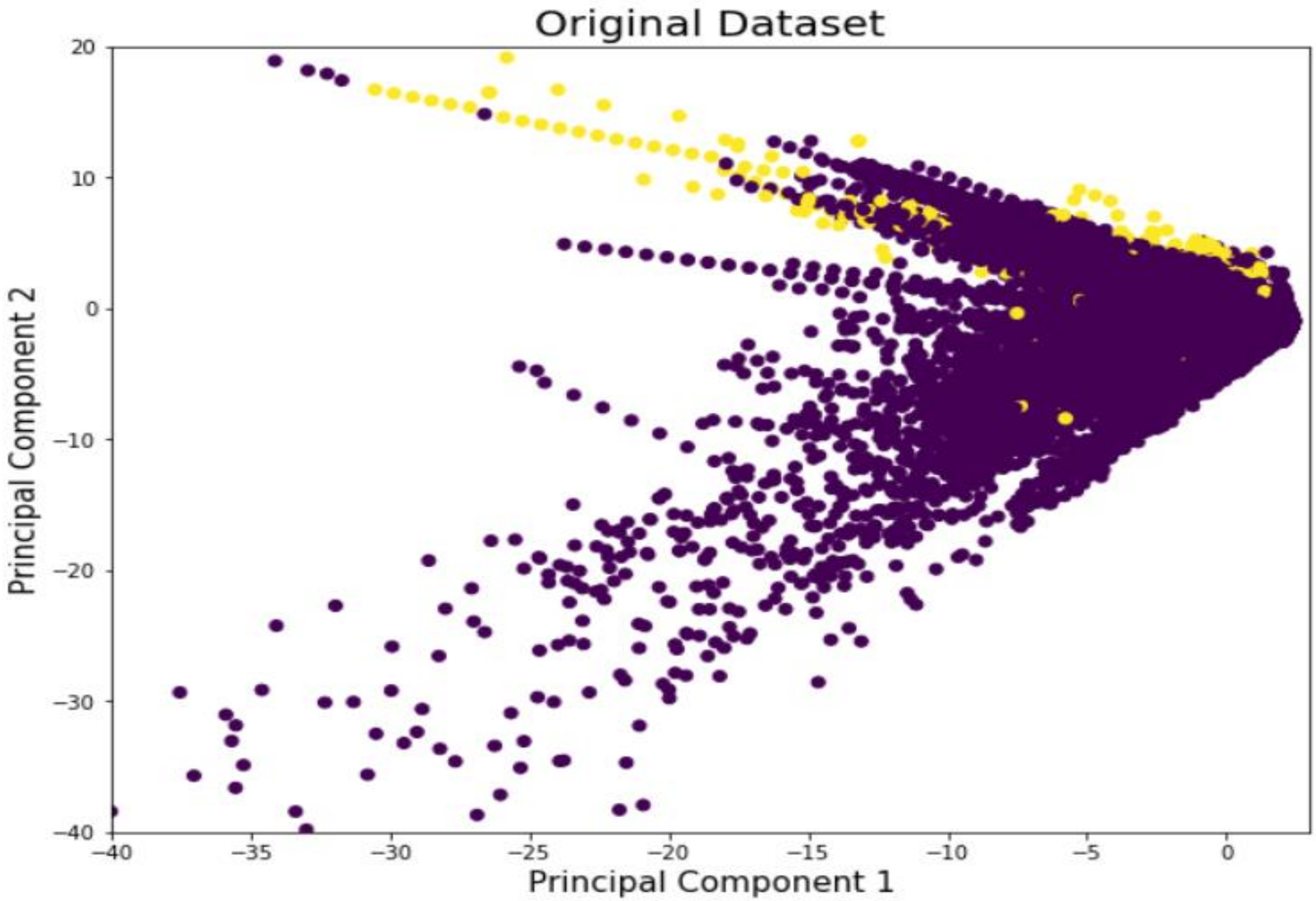
Imbalanced Classification Problem

A Credit Card Fraud Detection example

The Dataset

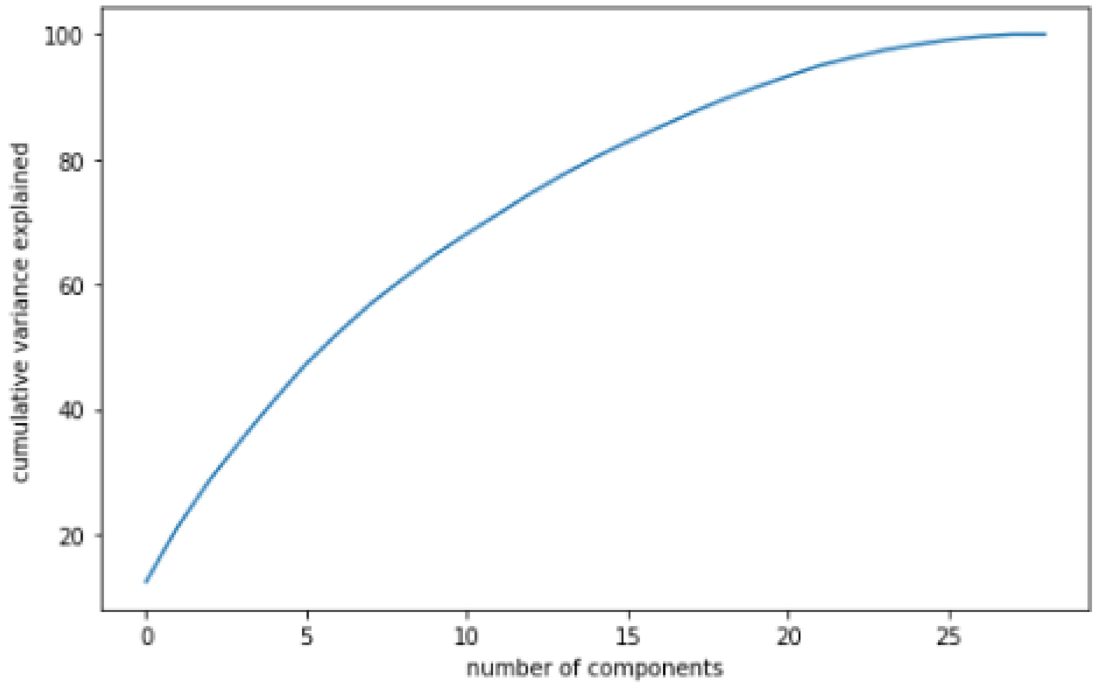
- Contains 284806 credit card transactions occurred in two days
- Original features transformed to Principal Components
- Amount – Transaction Amount
- Time – seconds elapsed after the first transaction

The Dataset



Original Dataset
20
10
0
−10
−20
−30
−40
Principal Component 2
−40
−35
−30
−25
−20
−15
−10
−5
0
Principal Component 1

Proportion of Variance

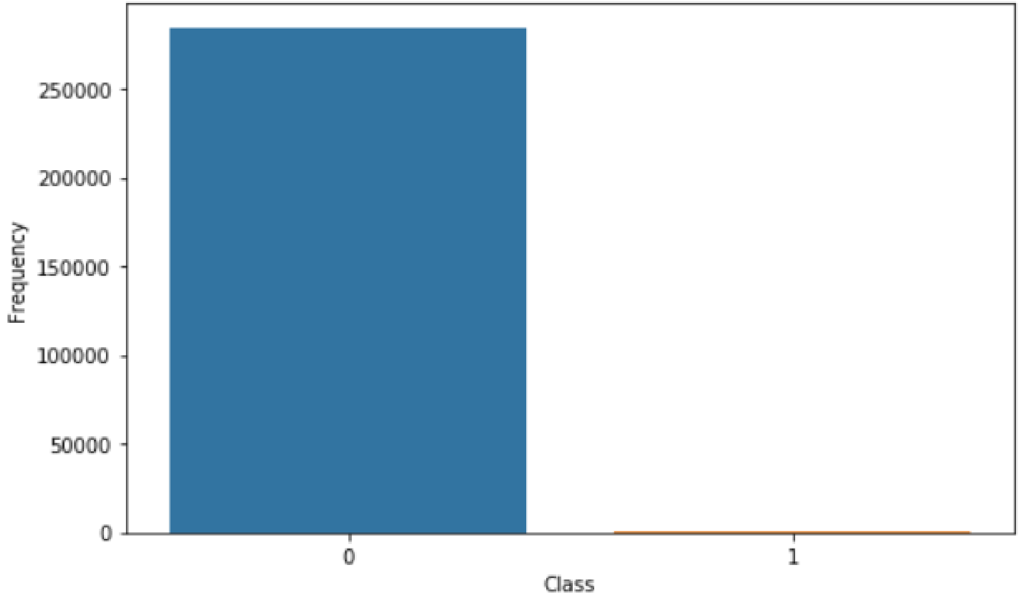


Class Imbalance

```
print(" No. of Genuine transactions = " + str(data['Class'].value_counts().iloc[0]))
print(" No. of Fraudulent transactions = " + str(data['Class'].value_counts().iloc[1]))
```

```
No. of Genuine transactions = 284315
No. of Fraudulent transactions = 492
```

```
plt.figure(figsize = (8,5))
sns.countplot(data['Class'])
plt.xlabel("Class")
plt.ylabel("Frequency")
plt.show()
```



Approaching Imbalanced Classification

Predictive Accuracy – not appropriate

1. Collect more data?

   Might expose a different and balanced perspective

2. Performance Metrics

3. Resampling – to generate balanced and unbiased training sets

Performance Metrics

	Pred. Negative	Pred. Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Table : Confusion Matrix

$$Sensitivity(Recall) = \frac{TP}{TP + FN}$$

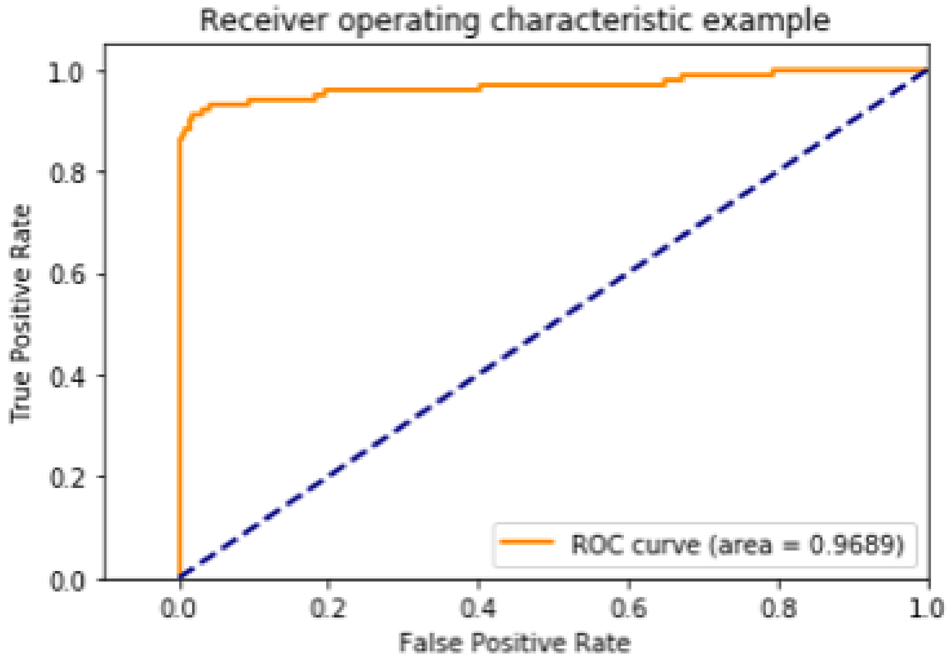
$$Specificity = \frac{TN}{TN + FP}$$

Sensitivity – Predictive Performance for Minority Class

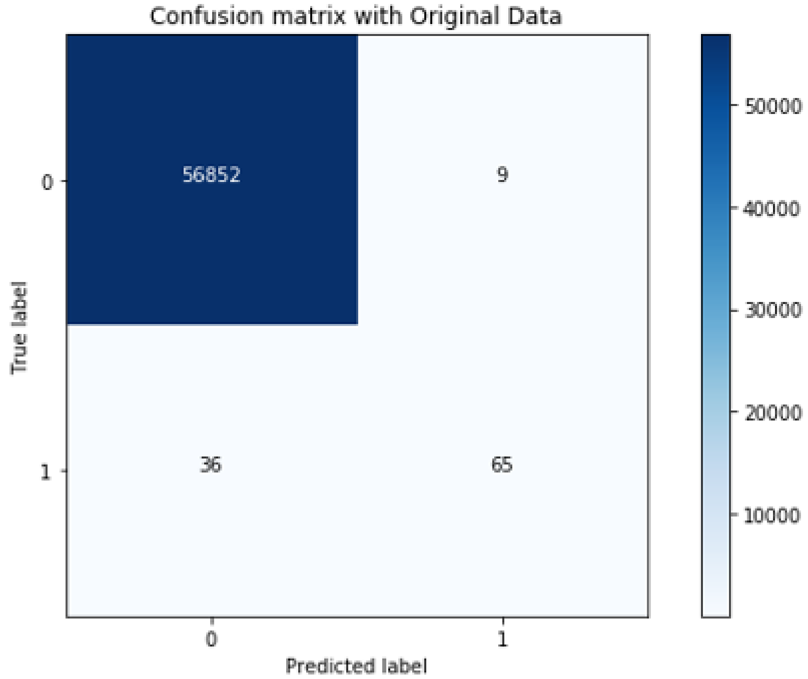
Specificity - Predictive Performance for Majority Class

Performance Metrics

AU - ROC Curve : Summarizes performance over a range of tradeoffs between TP and FP



Fitting Logistic Regression to Original Data



Sensitivity/Recall	Specificity
0.643	.999

Resampling Techniques

1. Oversampling

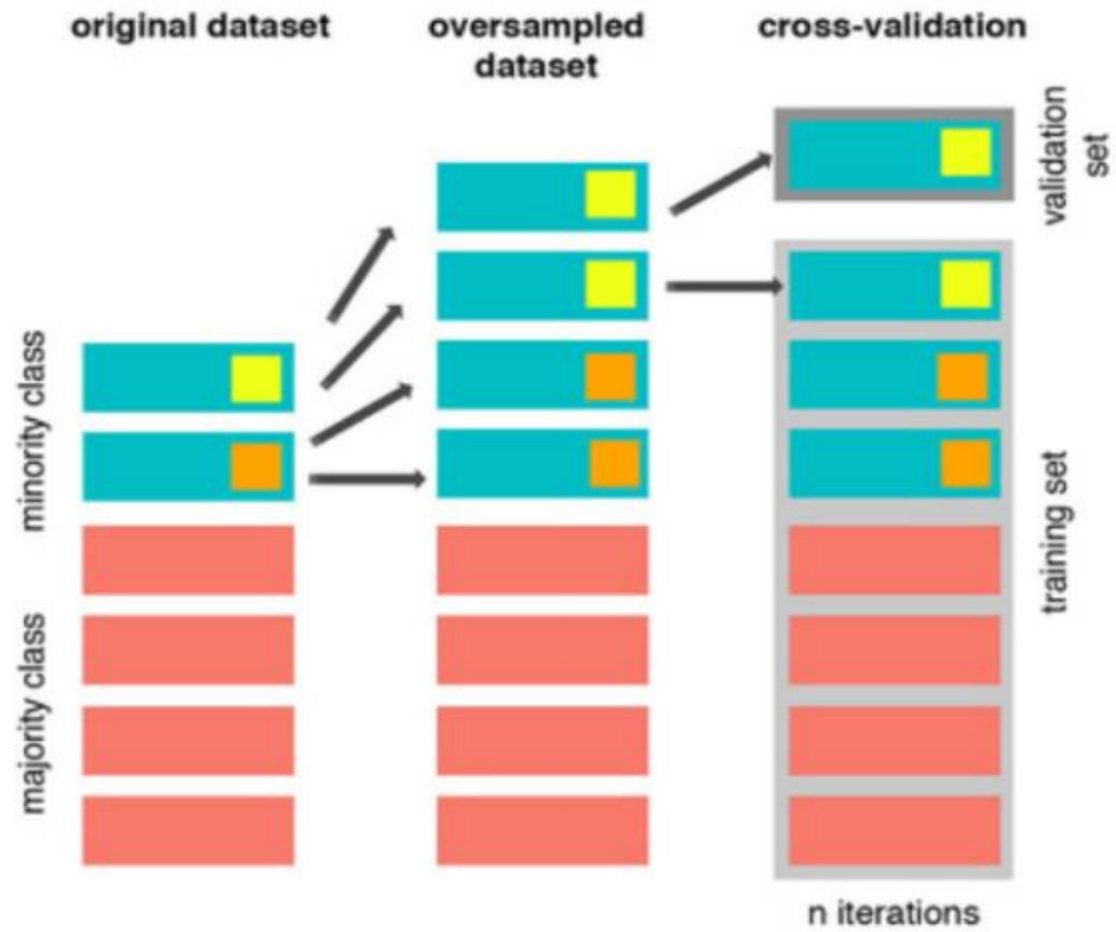
I. SMOTE
II. ADASYN

2. Random Undersampling

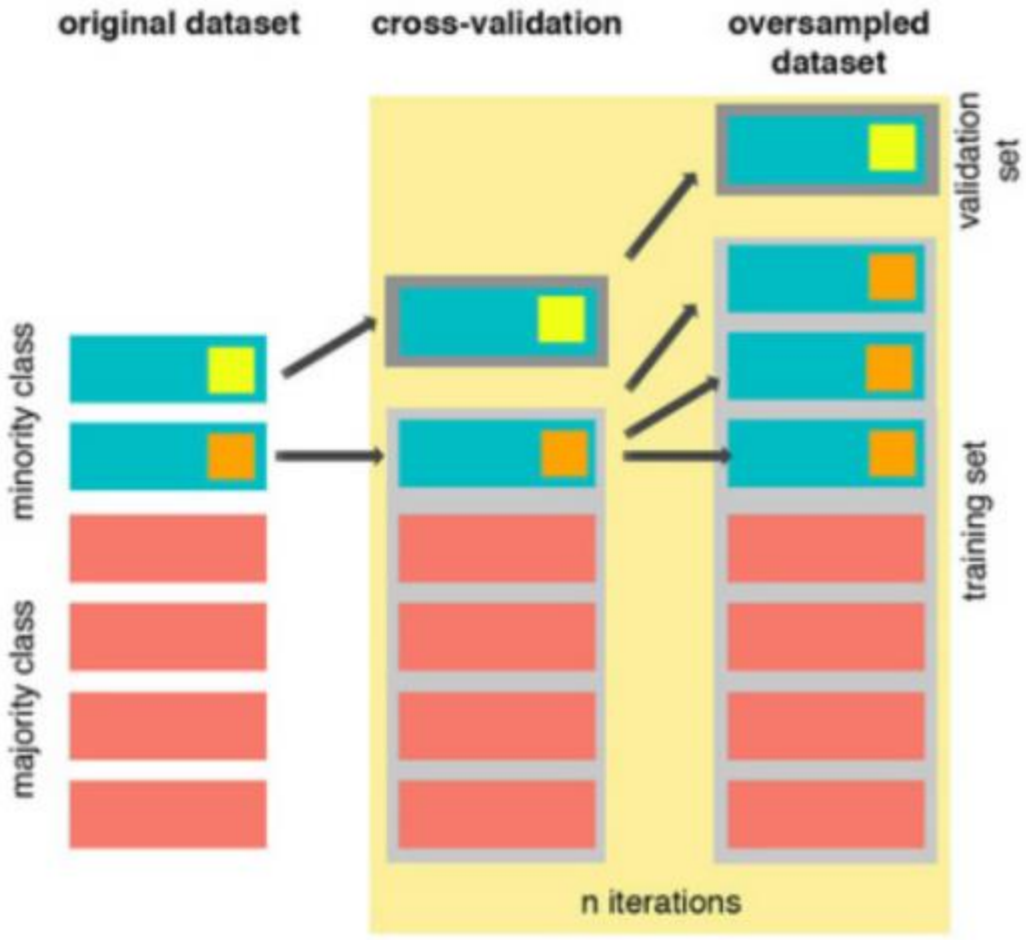
3. Ensemble Methods

I. Balanced Bagging
II. Easy Ensemble
III. Balanced Cascade

The Wrong Way to Oversample

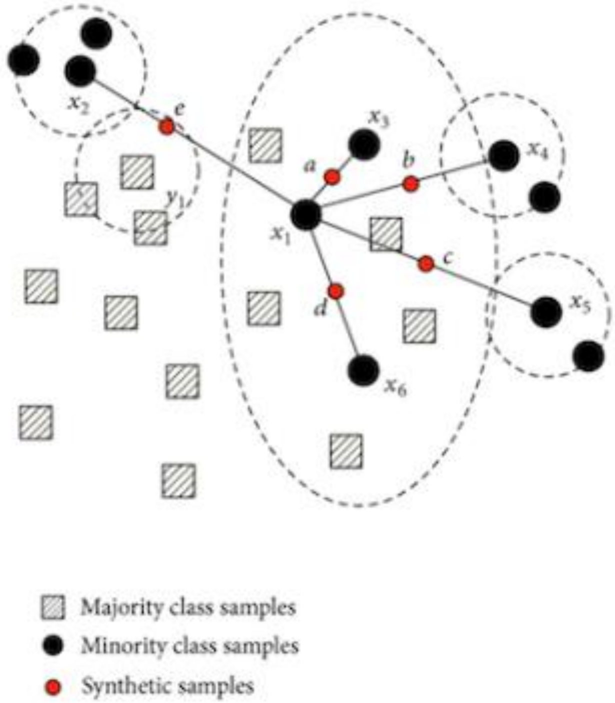


The Right Way



SMOTE

Forces the decision region of minority class to become more general and considers areas with predominant majority examples.



Visualization of SMOTE

ADASYN

- Assigns a weight to different minority class examples and generates more synthetic data for examples that are hard to learn.

- In essence, shifts the decision boundary towards more difficult minority class examples.

Steps for ADASYN

1. Calculate the no. of Synthetic examples to be generated based on desired balance level.

2. Find k-NN for each minority class example

3. Calculate the fraction of neighbors belonging to majority class.

4. Use the fraction to calculate the number of synthetic examples to be generated

5. Generate new samples by the process used by SMOTE.

Random Undersampling

- Randomly select majority class examples to create a 50/50 class ratio

Downside:

Ignores potentially useful majority class data and suffers from poor performance

Sensitivity/Recall	Specificity
0.94	0.88

Ensemble Methods

- Also known as Informed Undersampling
- Combines several classifiers built using different samples from majority class.

Balanced Bagging

Based on:

1. Bootstrap – plenty of diverse observations
2. Aggregation – helps reduce variance

Combines results obtained from different bootstrapped samples for both classes.

Sensitivity/Recall	Specificity
0.94	0.93

Easy Ensemble

AdaBoost : Combines multiple weak classifiers into a single strong classifier.

Initialize the weight for each data point.

For iteration m = 1, …, M

Fit weak classifiers to the dataset and select one with lowest classification error.

Calculate the weight for m-th weak classifier

$$\theta_m = \frac{1}{2} ln(\frac{1-\epsilon_m}{\epsilon_m}).$$

Update the weight for each data point as

$$w_{m+1}(x_i, y_i) = \frac{w_m(x_i, y_i) exp[-\theta_m y_i f_m(x_i)]}{Z_m},$$

Get the final prediction by summing up weighted prediction of each classifier.

$$F(x) = sign(\sum_{m=1}^{M} \theta_m f_m(x)),$$

Easy Ensemble

The `EasyEnsemble` algorithm.

1: {Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from $\mathcal{N}$, and s_i, the number of iterations to train an AdaBoost ensemble H_i}

2: $i \Leftarrow 0$

3: **repeat**

4: $\quad i \Leftarrow i + 1$

5: $\quad$ Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.

6: $\quad$ Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i, i.e.

$$H_i(x) = \operatorname{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right).$$

7: **until** $i = T$

8: Output: An ensemble:

$$H(x) = \operatorname{sgn}\left(\sum_{i=1}^{T} \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right).$$

Balanced Cascade

Easy Ensemble : Unsupervised Strategy since it uses independent random sampling with replacement.

Balanced Cascades :

Drops correctly classified majority examples from further sampling as the information is already contained in an AdaBoost Ensemble.

Results based on Logistic Regression

		Oversampling Ratio	Sensitivity/Recall	Specificity	AUC
Original			0.64	0.99	0.96
Undersampling			0.94	0.88	0.9689
Oversampling	SMOTE	0.25	0.90	0.99	0.9970
	SMOTE	0.5	0.91	0.99	0.9885
	SMOTE	1	0.94	0.97	0.9898
Oversampling	ADASYN	0.25	0.94	0.97	0.9900
	ADASYN	0.5	0.96	0.95	0.9907
	ADASYN	1	0.97	0.90	0.9911
Ensemble	BalancedBagging		0.94	0.93	0.9729
	EasyEnsemble		0.91	0.97	0.9781