CLASS IMBALANCE

Data are said to suffer the ***Class Imbalance Problem*** when the class distributions are highly ***imbalanced***. This is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes. Machine Learning algorithms tend to produce unacceptable predictions when faced with imbalanced datasets.

Various techniques to treat imbalanced classes such as-
1.Random under-sampling

2.Random over-sampling

3.NearMiss

When observation in one class is higher than the observation in other classes then there exists a class imbalance. Example: To detect fraudulent credit card transactions. As you can see in the below graph fraudulent transaction is around 400 when compared with non-fraudulent transaction around 90000.

Class Imbalance is a common problem in machine learning, especially in classification problems. Imbalance data can hamper our model accuracy big time.

Class Imbalance appear in many domains, including:

Fraud detection

Spam filtering

Disease screening

SaaS subscription churn

Advertising click-throughs

The Problem with Class Imbalance

Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce errors.

However, if the data set in imbalance then In such cases, you get a pretty high accuracy just by predicting the **majority class**, but you fail to capture the **minority class**, which is most often the point of creating the model in the first place.

Credit card fraud detection example

Let's say we have a dataset of credit card companies where we have to find out whether the credit card transaction was fraudulent or not.

But here's the catch… the fraud transaction is relatively rare, only 6% of the transaction is fraudulent.

Now, before you even start, do you see how the problem might break? Imagine if you didn't bother training a model at all. Instead, what if you just wrote a single line of code that always predicts 'no fraudulent transaction'

The accuracy of the classification method is misleading.

All those **non-fraudulent** transactions, you'd have 100% accuracy.

Those transactions which are **fraudulent**, you'd have 0% accuracy.

Your overall **accuracy would be high** simply because the most transaction is not fraudulent(not because your model is any good).

This is clearly a problem because many machine learning algorithms are designed to maximize overall accuracy. In this article, we will see different techniques to handle the imbalanced data.

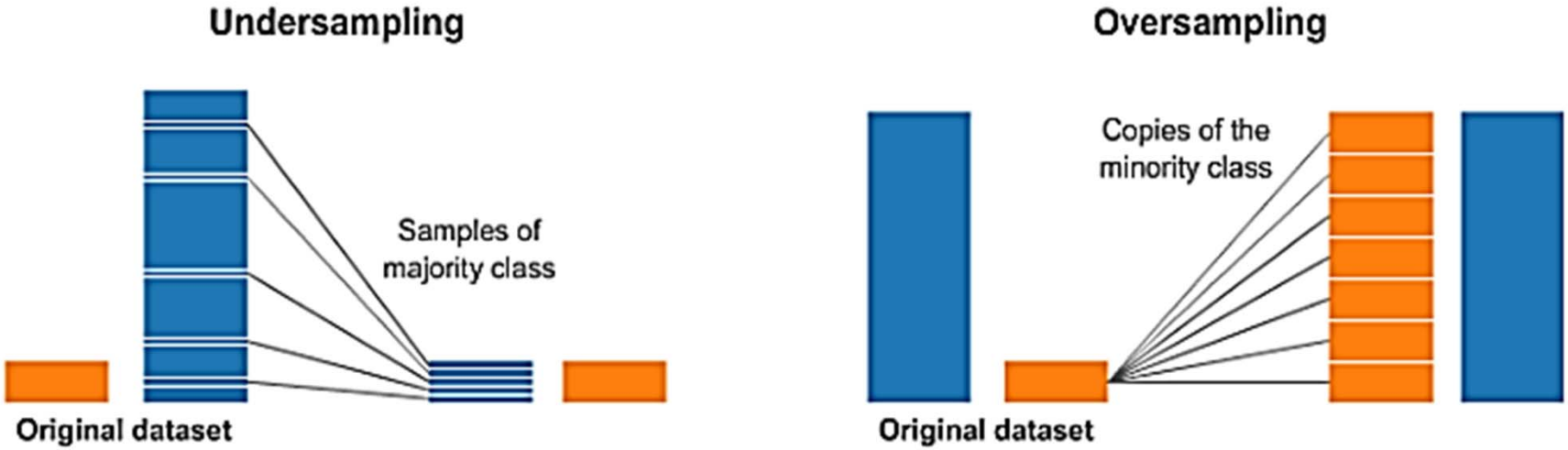
The Metric Trap

One of the major issues that new developer users fall into when dealing with unbalanced datasets relates to the metrics used to evaluate their model. Using simpler metrics like **accuracy score** can be misleading. In a dataset with highly unbalanced classes, the classifier will always “predicts” the most common class without performing any analysis of the features and it will have a high accuracy rate, obviously not the correct one.

We can see 99% accuracy, we are getting very high accuracy because it is predicting mostly the **majority** class that is 0 (Non-fraudulent).

Resampling Technique

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) and/or adding more examples from the minority class (over-sampling).



Despite the advantage of balancing classes, these techniques also have their weaknesses (there is no free lunch).

The simplest implementation of **over-sampling** is to duplicate random records from the minority class, which can cause overfishing.

In **under-sampling**, the simplest technique involves removing random records from the majority class, which can cause loss of information.

1. Random Under-Sampling

Undersampling can be defined as **removing some observations of the majority class**. This is done until the majority and minority class is balanced out.

Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback to undersampling is that we are removing information that may be valuable.

2. Random Over-Sampling

Oversampling can be defined as adding more copies to the minority class. Oversampling can be a good choice when you don’t have a ton of data to work with.

A drawback to consider when undersampling is that it can cause overfitting and poor generalization to your test set.

Balance data with the imbalanced-learn module

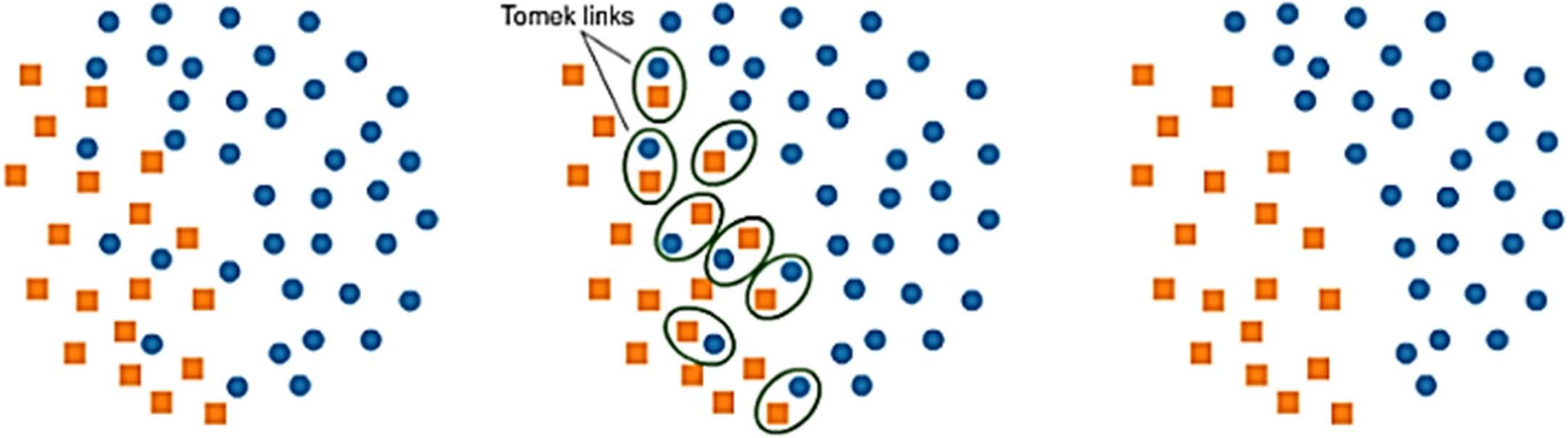
A number of more sophisticated resampling techniques have been proposed in the scientific literature.

For example, we can cluster the records of the majority class, and do the under-sampling by removing records from each cluster, thus seeking to preserve information. In over-sampling, instead of creating exact copies of the minority class records, we can introduce small variations into those copies, creating more diverse synthetic samples.

Under-sampling: Tomek links

Tomek links are pairs of very close instances but of opposite classes. Removing the instances of the majority class of each pair increases the space between the two classes, facilitating the classification process.

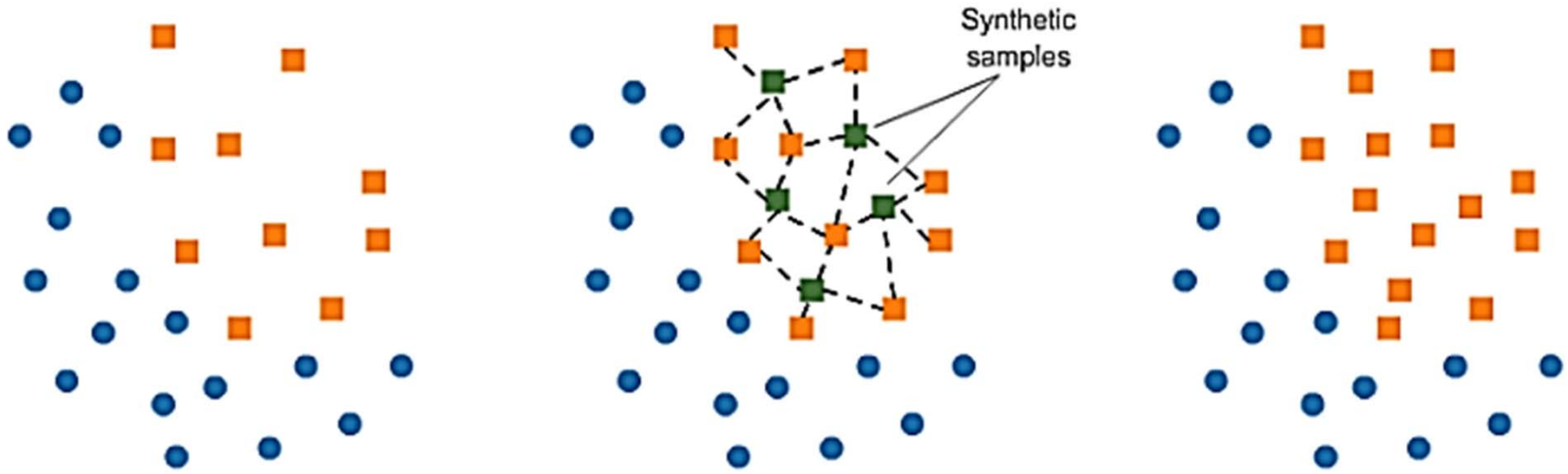
Tomek's link exists if the two samples are the nearest neighbors of each other



Synthetic Minority Oversampling Technique (SMOTE)

This technique generates synthetic data for the minority class.

SMOTE (Synthetic Minority Oversampling Technique) works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The **synthetic points are added** between the chosen point and its neighbors.



SMOTE algorithm works in 4 simple steps:

1. Choose a minority class as the input vector
2. Find its k nearest neighbors (***k_neighbors*** is specified as an argument in the ***SMOTE()*** function)
3. Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbor
4. Repeat the steps until data is balanced

NearMiss

NearMiss is an under-sampling technique. Instead of resampling the Minority class, using a distance, this will make the majority class equal to the minority class.

Change the performance metric

Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be misleading.

Metrics that can provide better insight are:

Confusion Matrix: a table showing correct predictions and types of incorrect predictions.

Precision: the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.

Recall: the number of true positives divided by the number of positive values in the test data. The recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.

F1: Score: the weighted average of precision and recall.

Area Under ROC Curve (AUROC): AUROC represents the likelihood of your model distinguishing observations from two classes.
In other words, if you randomly select one observation from each class, what's the probability that your model will be able to "rank" them correctly?

Penalize Algorithms (Cost-Sensitive Training)

The next tactic is to use penalized learning algorithms that increase the cost of classification mistakes on the minority class.

A popular algorithm for this technique is Penalized-SVM.

Change the algorithm

While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets.

Decision trees frequently perform well on imbalanced data. In modern machine learning, tree ensembles (Random Forests, Gradient Boosted Trees, etc.) almost always outperform singular decision trees, so we'll jump right into those:

Tree base algorithm work by learning a hierarchy of if/else questions. This can force both classes to be addressed.

Advantage and disadvantages of Under-sampling

Advantages

It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.

Disadvantages

It can discard potentially useful information which could be important for building rule classifiers.

The sample chosen by random under-sampling may be a biased sample. And it will not be an accurate representation of the population. Thereby, resulting in inaccurate results with the actual test data set.

Advantages and Disadvantage of over-sampling

Advantages

Unlike under-sampling, this method leads to no information loss.

Outperforms under sampling

Disadvantages

It increases the likelihood of overfitting since it replicates the minority class events.

GRAPH MINING

Graph Mining is a Methods for

Mining Frequent Subgraphs

& Mining Variant and Constrained Substructure Patterns

Applications of Graph Mining are :

Graph Indexing

Similarity Search

Classification and Clustering

Why Graph Mining ?

Graphs are ubiquitous Chemical compounds (Cheminformatics). Protein structures, biological pathways/networks (Bioinformactics).

Program control flow, traffic flow, and workflow analysis.

XML databases, Web, and social network analysis.

Graph is a general model Trees, lattices, sequences, and items are degenerated graphs.

Diversity of graphs Directed vs. undirected, labeled vs. unlabeled (edges & vertices), weighted, with angles & geometry (topological vs. 2-D/3-D).

Complexity of algorithms: many problems are of high complexity.

Graph Pattern Mining

Frequent subgraphs

A (sub)graph is **frequent** if its support (occurrence frequency) in a given dataset is no less than a minimum support threshold

Applications of graph pattern mining

- Mining biochemical structures
- Program control flow analysis
- Mining XML structures or Web communities

Closed Frequent Graphs

- Motivation: Handling graph pattern explosion problem
- Closed frequent graph
  - A frequent graph G is *closed* if there exists no supergraph of G that carries the same support as G
- If some of G's subgraphs have the same support, it is unnecessary to output these subgraphs (i.e. non-closed graphs)

Graph Clustering

Graph similarity measure

- **Feature-based similarity measure**
  - Each graph is represented as a feature vector
  - The similarity is defined by the distance of their corresponding vectors
  - Frequent subgraphs can be used as features
- **Structure-based similarity measure**
  - Maximal common subgraph
  - Graph edit distance: insertion, deletion, and relabel
  - Graph alignment distance

Graph Classification

Local structure based approach

- Local structures in a graph, e.g., neighbors surrounding a vertex, paths with fixed length

Graph pattern-based approach

- Subgraph patterns from domain knowledge
- Subgraph patterns from data mining

Graph Pattern-Based Classification

- Subgraph patterns from domain knowledge
  - Molecular descriptors
- Subgraph patterns from data mining
- General idea
  - Each graph is represented as a feature vector **x** = {x1, x2, ..., xn}, where xi is the frequency of the i-th pattern in that graph
  - Each vector is associated with a class label
  - Classify these vectors in a vector space

Graph Mining Challenges

- **High computational cost**
  - Need to tradeoff between efficiency/scalability and quality
- **Sophisticated graphs**
  - May involve weights and/or cycles.
- **High dimensionality**
  - A graph can have many vertices. In a similarity matrix, a vertex is represented as a vector (a row in the matrix) whose dimensionality is the number of vertices in the graph
- **Sparsity**
  - A large graph is often sparse, meaning each vertex on average connects to only a small number of other vertices
  - A similarity matrix from a large sparse graph can also be sparse