While working as a data scientist, some of the most frequently occurring problem statements are related to binary classification. A common problem when solving these problem statements is imbalance classification. When observation in one class is higher than in other classes, a class imbalance exists. Example: To detect fraudulent credit card transactions. Eg: fraudulent transaction is around 400 compared to the non-fraudulent transaction of around 90000.

Class Imbalance in machine learning oversampling in machine learning is a common problem in machine learning, especially in classification problems. Imbalance data can hamper our model accuracy big time. It appears in many domains, including fraud detection, spam filtering, disease screening, SaaS subscription churn, advertising click-throughs, etc. Let’s understand how to deal with imbalanced data in machine learning. We will understand oversampling and under sampling, about the class imbalance in machine learning and how to deal with class imbalance in classification.

A model trained on an imbalanced dataset perform poorly on the minority class. At best, this can cause loss to the business in the case of a churn analysis. At worst, it can pervade systemic bias of a face recognition system.

To address the challenges posed by class imbalance, we can employ three main approaches: using appropriate evaluation metrics, data-level methods, and algorithm-level methods.

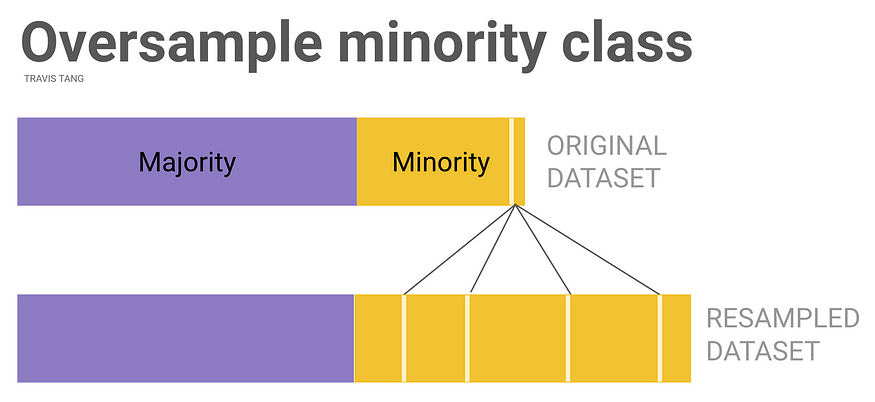
The common approach to class imbalance is resampling. These can entail oversampling the majority class, undersampling the minority class, or a combination of both.

1. Random oversampling
2. Random undersampling
3. Oversampling with SMOTE
4. Oversampling with ADASYN
5. Undersampling with Tomek Link
6. Oversampling with SMOTE, then undersample with TOMEK Link (SMOTE-Tomek)

# Data-Level Methods

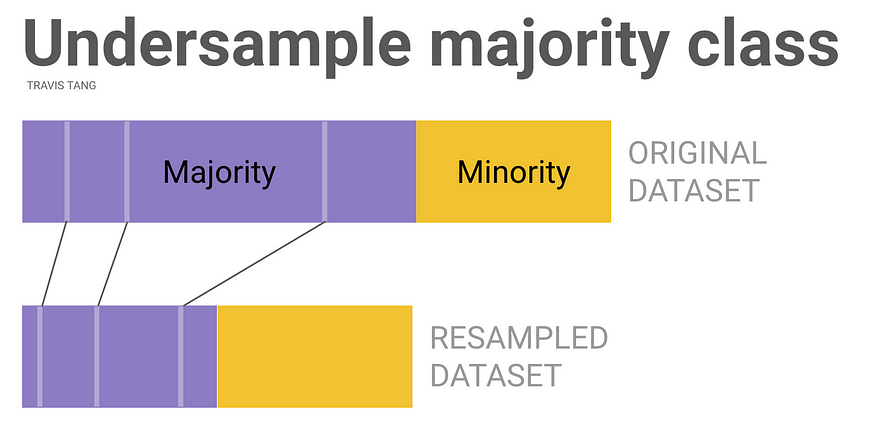
# Random Oversampling

Random oversampling duplicates existing examples from the minority class with replacement. Each data point in the minority class has an equal probability of being duplicated.

1. 

**Strategy 2. Random Undersampling**

Conversely, random undersampling removes existing samples from the majority class. Each data point in the majority class has an equal chance of being removed.



**Strategy 3. Oversampling with SMOTE**

SMOTE is a method of oversampling. Intuitively, SMOTE creates synthetic data points by interpolating between the minority data points that are close by to one another.

Here’s how SMOTE works (simplified).

1. Randomly select some data points in the minority class.
2. For every selected point, identify its *k* nearest neighbour(s).
3. For every neighbor, add a new point somewhere between the data point and the neighbor.
4. Repeat steps 2 to 4 until sufficient synthetic data points are created.

Oversampling with ADASYN (+ How it’s different from SMOTE)

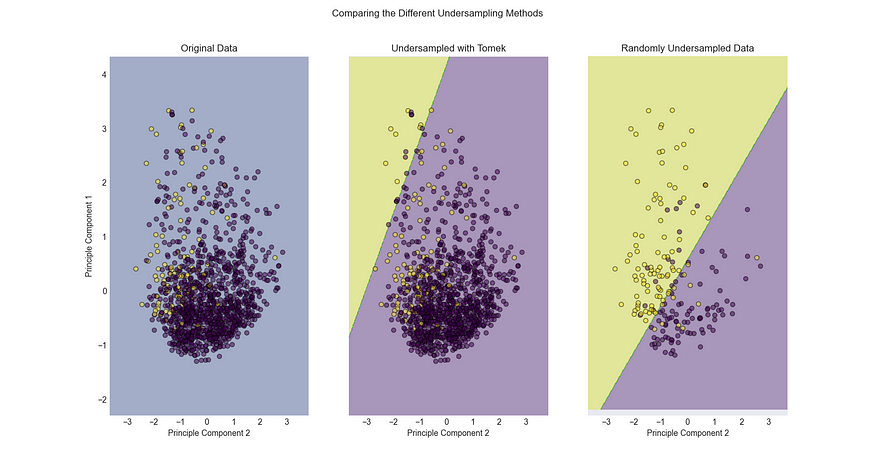
ADASYN is a cousin of SMOTE: both SMOTE and ADASYN generate new samples by interpolation. But there’s on critical difference. ADASYN generates samples next to the original samples that are wrongly classified by a KNN classifier. Conversely, SMOTE differentiates between samples that are correctly or wrongly classified by the KNN classifier.

**Under-sampling with Tomek Links**

A tomek link is a pair of points that are very close to one another but are of different classes.

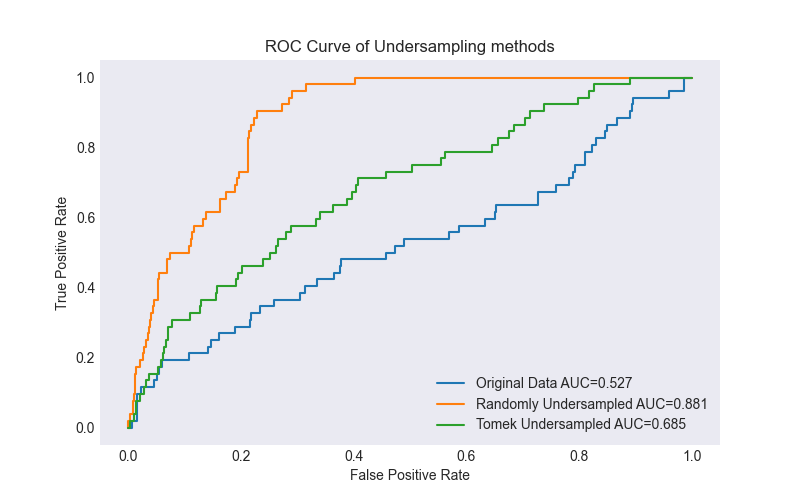
To under-sample with Tomek Links, we will identify all Tomek Links in the data set. For each pair of data point in the Tomek Link, we will remove the majority class.

let’s compare Tomek undersampling with random undersampling.



In our particular dataset, removing Tomek Link did little to ease the class imbalance. This is because there are limited number of Tomek Links in the dataaset.

Let’s see how the performance of undersampling with Tomek Link differs from that of random undersampling.



We observe that **random undersampling did better than Tomek Link undersampling.**This is because Tomek Link did not remove the class imbalance completely like random undersampling did.

**SMOTEK: Oversample with SMOTE, then Undersample with Tomek Links**

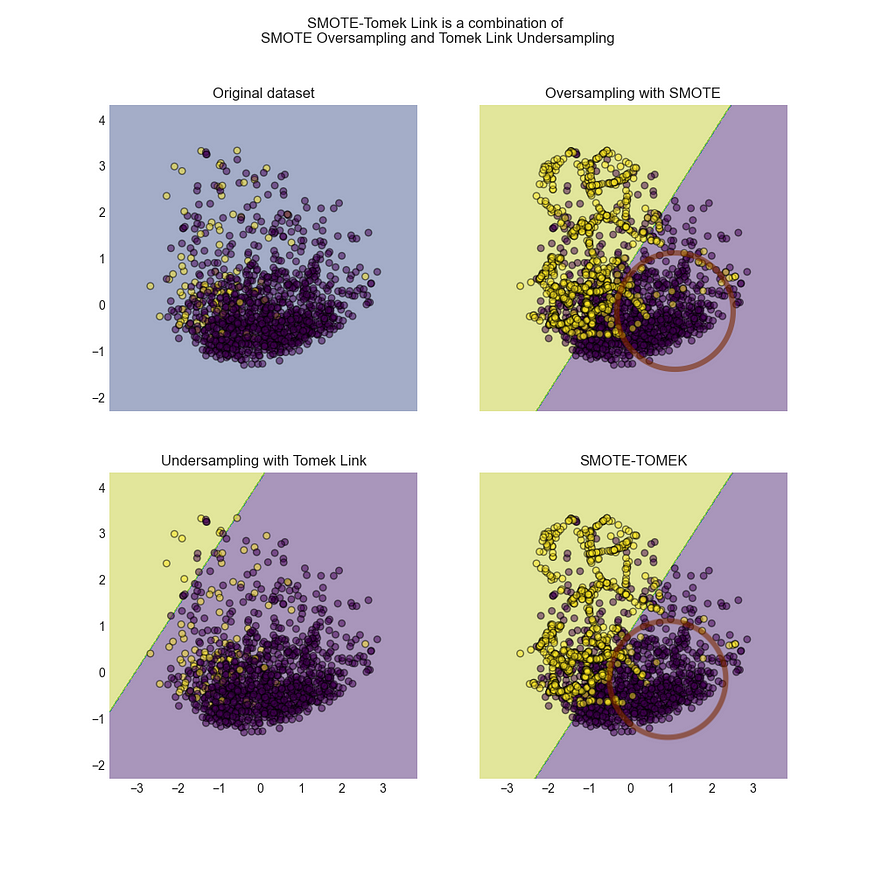
Now that we have learnt about oversampling and undersampling. Can we combine these techniques?

Of course! **SMOTE-TOMEK** is a technique that combines oversampling (SMOTE) with undersampling (with Tomek Links).

We will apply it to our dataset.

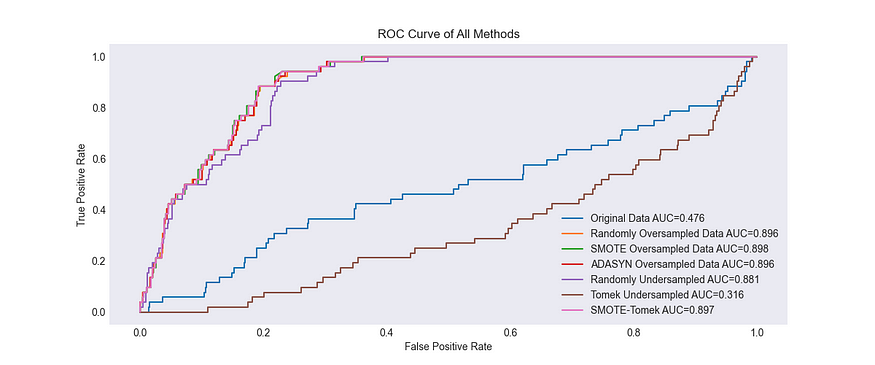
from imblearn.combine import SMOTETomek  
from sklearn.svm import LinearSVC  
  
# Perform random sampling  
smotetomek = SMOTETomek(random\_state=0)  
X\_train\_smotetomek, y\_train\_smotetomek = smotetomek.fit\_resample(X\_train\_pca, y\_train)  
  
# Plot linear SVC  
clf\_smotetomek = SVC(kernel='linear',probability=True)  
clf\_smotetomek.fit(X\_train\_smotetomek, y\_train\_smotetomek)

Let’s compare SMOTE, Tomek, and SMOTE-Tomek.



Comparing SMOTE-Tomek with SMOTE Only, we see the difference being circled in brown. SMOTE-Tomek removes the points that are close to the boundary.

For the grand finale, we will compare all the techniques that we have described above. Viola, SMOTE-TOMEK performed the best.



Overall, you can use oversampling, undersampling or a combination of both to deal with data imbalance. If you have the computational resources, it is often better to use a combination of over- and under-sampling; Oversampling is a good strategy when you have few datapoints; while undersampling is good when there are potentially many similar data points.

Also, measuring the performance of imbalance dataset can be tricky. Make sure you use the right classification metrics. Luckily, metrics like [ROC Curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html), [F1 score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html) and [geometric mean scores](https://imbalanced-learn.org/dev/references/metrics.html) are already available to us.

**Using the Right Evaluation Metrics**

**Choosing the appropriate evaluation metrics is crucial when dealing with class imbalance.**

**Traditional metrics like overall accuracy and error rate can be misleading in imbalanced scenarios, as they are dominated by the performance on the majority class.**

**Instead, we should focus on metrics that provide a more comprehensive view of the model’s performance, such as:**

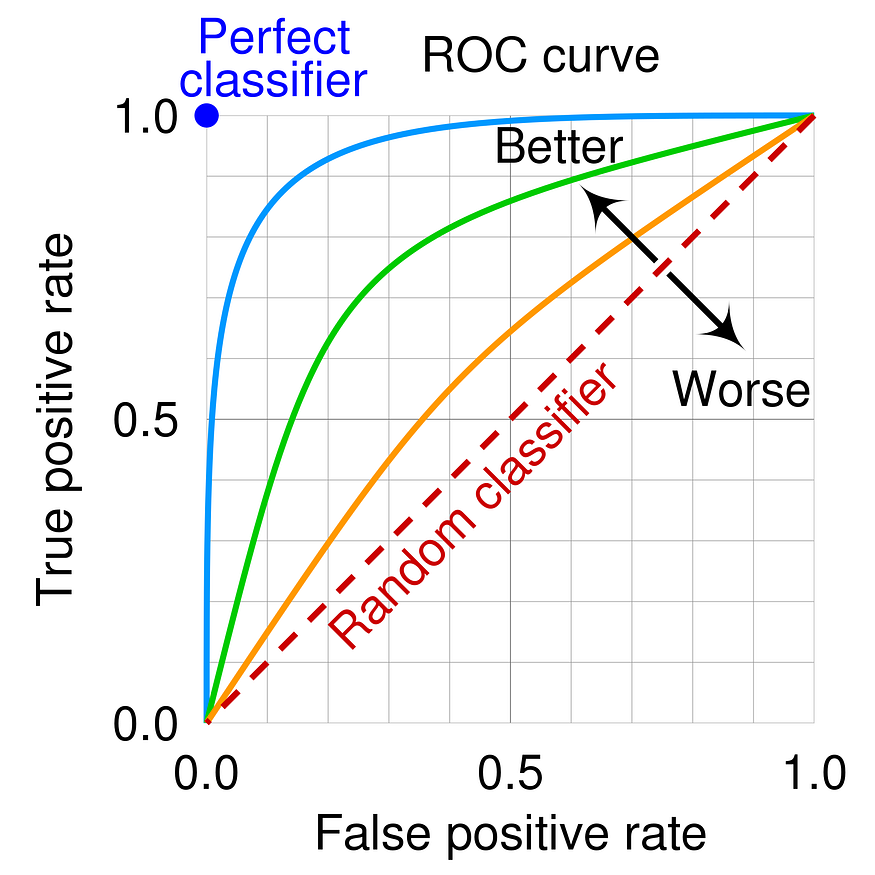
* **Precision: Measures the proportion of true positive predictions among all positive predictions.**
* **Recall (Sensitivity or True Positive Rate): Measures the proportion of true positive predictions among all actual positive samples.**
* **F1 Score: The harmonic mean of precision and recall, providing a balanced measure of the model’s performance.**

**Additionally, we can use the Receiver Operating Characteristic (ROC) curve and the Precision-Recall curve to visualize the model’s performance at different classification thresholds.**

**These curves help us understand the trade-off between true positive rate and false positive rate (ROC curve) or precision and recall (Precision-Recall curve), allowing us to select an appropriate threshold based on our specific requirements.**

**The Precision-Recall curve is particularly useful in evaluating models for imbalanced datasets.**

ROC is insensitive to class imbalance, making it a great tool to evaluate models with class imbalance. It does not depend on the class prevalence. This is in contrast to evaluation metrics such as accuracy, which can be misleading in the presence of class imbalance.



.

An ROC curve plots the true positive rate (TPR) on the y-axis against the false positive rate (FPR) on the x-axis for all possible classification thresholds. The TPR is the proportion of positive instances that are correctly classified as positive, and the FPR is the proportion of negative instances that are incorrectly classified as positive.

A model with good performance will have a ROC curve that is closer to the top-left corner of the plot, as this indicates a higher TPR and a lower FPR. A model makes completely random guesses will fall on the line with TPR = FPR.

*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. Our implementation currently uses five nearest neighbors. 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. 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. This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general*

[Class Imbalance Strategies — A Visual Guide with Code | by Travis Tang | Towards Data Science](https://towardsdatascience.com/class-imbalance-strategies-a-visual-guide-with-code-8bc8fae71e1a)

[Machine Learning: How to Handle Class Imbalance | by Ken Hoffman | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/machine-learning-how-to-handle-class-imbalance-920e48c3e970)

[Practical ML: Addressing Class Imbalance | by Juan C Olamendy | Medium](https://medium.com/@juanc.olamendy/practical-ml-addressing-class-imbalance-25c4f1b97ee3)

[10 Techniques to handle imbalance class in Machine Learning (analyticsvidhya.com)](https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/)