Econ 425 Week 5

Imbalanced data

Grigory Franguridi

UCLA Econ

USC CESR

franguri@usc.edu

Example: credit card fraud detection

- Data on credit card transactions:
  - Total transactions: 1,000,000
  - Legitimate $(y = 0)$ transactions: 995,000 (99.5%)
  - Fraudulent $(y = 1)$ transactions: 5,000 (0.5%)
- Goal: develop an ML model that accurately identifies fraudulent transactions

Example: credit card fraud detection

- **Challenges**:
  - **Model bias**: most classification algorithms will favor the majority class, leading to poor identification of fraudulent transactions.
  - **Evaluation metrics**: Traditional accuracy is not an appropriate performance metric
    - model: always predict "legitimate"; accuracy = 99.5%
  - **Overfitting the minority class**: attempts to focus on the minority class can lead the model to overfit the fraudulent transactions, increasing the test error

Imbalanced data: approaches

- **Choosing a proper evaluation metric**
- **Resampling**
- **Synthetic Minority Oversampling Technique (SMOTE)**

Choosing a proper evaluation metric

- Accuracy $= \frac{\text{number of correct predictions}}{n_{test}}$
  - good enough for balanced data, but not ideal for the imbalanced data
- Other metrics:

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

is the measure of how accurate is prediction of the positive class

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

is the measure of the classifier's ability to identify the positive class

Choosing a proper evaluation metric

- F_1 score is a more appropriate metric; it is the harmonic mean of precision and recall:

$$F_1 = 2 * \frac{\text{precison} * \text{recall}}{\text{precison} + \text{recall}}$$

- if the classifier misclassifies the minority (negative) class (i.e., False Positive increases), precision is low and so is F_1 score
- if the classifier identifies the minority class poorly (i.e. more of the majority class wrongfully predicted as the minority class), then False Negatives increase, so recall and F_1 score are low
- conclusion: F_1 score only increases if both the **number** and **quality** of prediction improves

Example: evaluation metrics

Identification of **rare disease**

- Data:
  - Total patients: 1000
  - Patients with disease (Positive class): 30
  - Patients without disease (Negative class): 970
- Model A:
  - True Positives (TP): 20 (correctly identified as sick)
  - False Positives (FP): 10 (incorrectly identified as sick)
  - False Negatives (FN): 10 (incorrectly identified as healthy)
  - True Negatives (TN): 960 (correctly identified as healthy)
- Model B:
  - True Positives (TP): 15
  - False Positives (FP): 5
  - False Negatives (FN): 15
  - False Negatives (FN): 965

Example: evaluation metrics

- **Model A**:
  - Precision $= 20/(20+10) = 0.67$
  - Recall $= 20/(20+10) = 0.67$
  - F1 Score $= 2 * (0.67 * 0.67)/(0.67 + 0.67) = 0.67$
- **Model B**:
  - Precision $= 15/(15+5) = 0.75$
  - Recall $= 15/(15+15) = 0.50$
  - F1 Score $= 2 * (0.75 * 0.50)/(0.75 + 0.50) \approx 0.60$

Example: evaluation metrics

- **Precision** = proportion of positives that were actually correct
  - Model B has a higher precision than Model A, suggesting it is better at ensuring that when it predicts the disease, it is more likely correct
- **Recall** = proportion of actual positives that were identified correctly
  - Model A has a higher recall than Model B, indicating it is better at capturing as many actual cases of the disease as possible
- **F1 score** = combination of precision and recall. Useful in imbalanced datasets because it considers both false positives and false negatives
  - although Model B has higher precision, its recall is significantly lower, resulting in a lower F1 score. Model A, despite having lower precision, has a balanced performance between precision and recall, leading to a higher F1 score

Example: conclusion

F1 score:

- is a more appropriate metric than precision and recall for imbalanced datasets where a model might have a high precision by predicting the majority class correctly but fails to capture the minority class effectively, or vice versa
- helps to identify models that maintain a balance between identifying positive cases correctly and minimizing false positives, which is important in critical applications like medical diagnostics

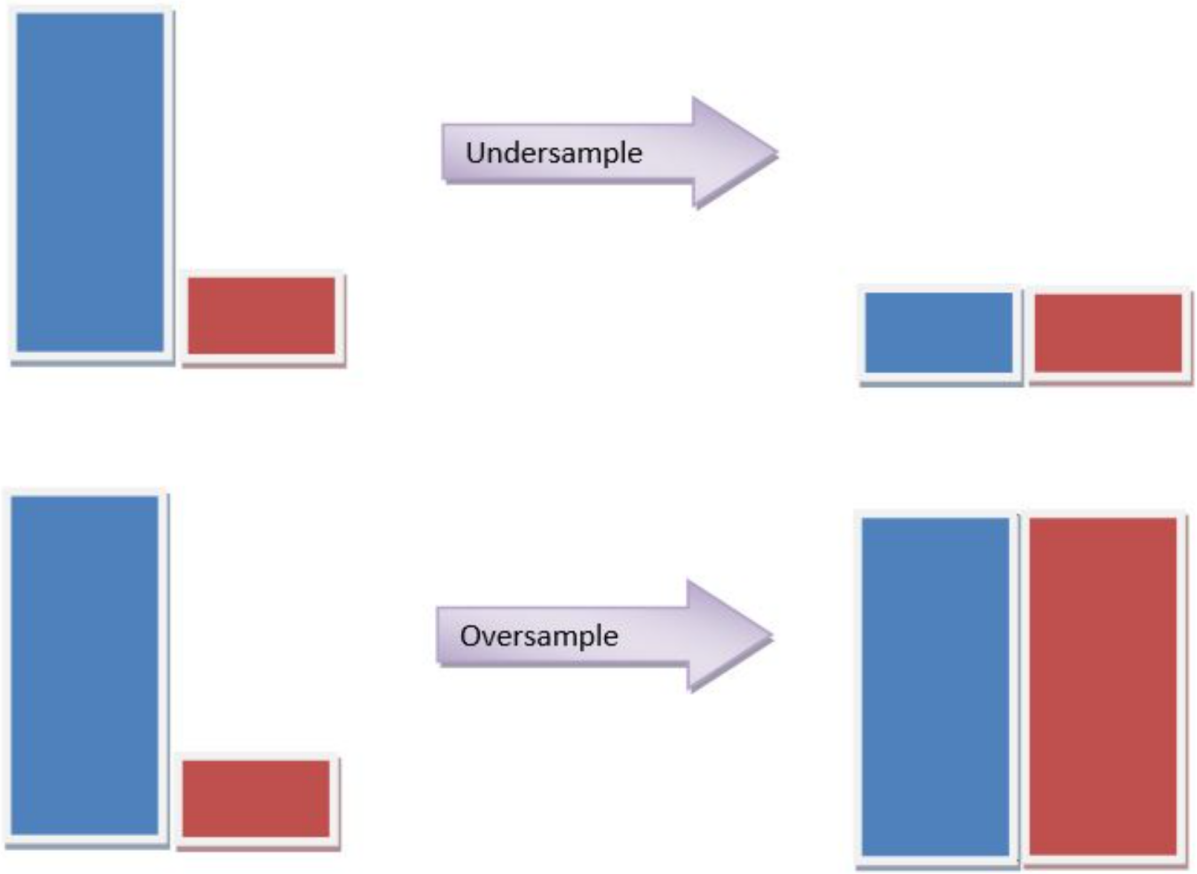
Resampling methods

- correct imbalance by altering the dataset to have a more balanced class distribution
- Data-centric AI vs algorithm-centric AI

Resampling methods

- **oversampling the minority class**: increasing the number of instances in the minority class(es) in a dataset
- **undersampling the majority class**: reducing the number of instances in the majority class(es)

Oversampling vs undersamping



Exercise: manual resampling

- **Dataset**:
  - two classes: Positive (P) and Negative (N)
  - imbalanced: 8 N and 2 P instances
  - N instances: N1, N2, N3, N4, N5, N6, N7, N8
  - P instances: P1, P2
- **Goal**: manually apply undersampling/oversampling to balance the dataset

Exercise: manual resampling

- **Undersampling**
  - Goal: reduce the number of instances in the majority class (negative) to match the minority class (positive).
  - Instructions: choose 2 instances randomly from the N class to keep. Your new dataset should have 2 Positive instances and 2 Negative instances.

Exercise: manual resampling

- **Example Solution:**
  - Selected N instances: N3, N7
  - New dataset: N3, N7, P1, P2

Exercise: manual resampling

- **Oversampling**
  - Goal: increase the number of instances in the minority class (P) to match the majority class (N)
  - Instructions: duplicate instances from the P class until there are as many P instances as N. The new dataset should have 8 P instances and 8 N instances

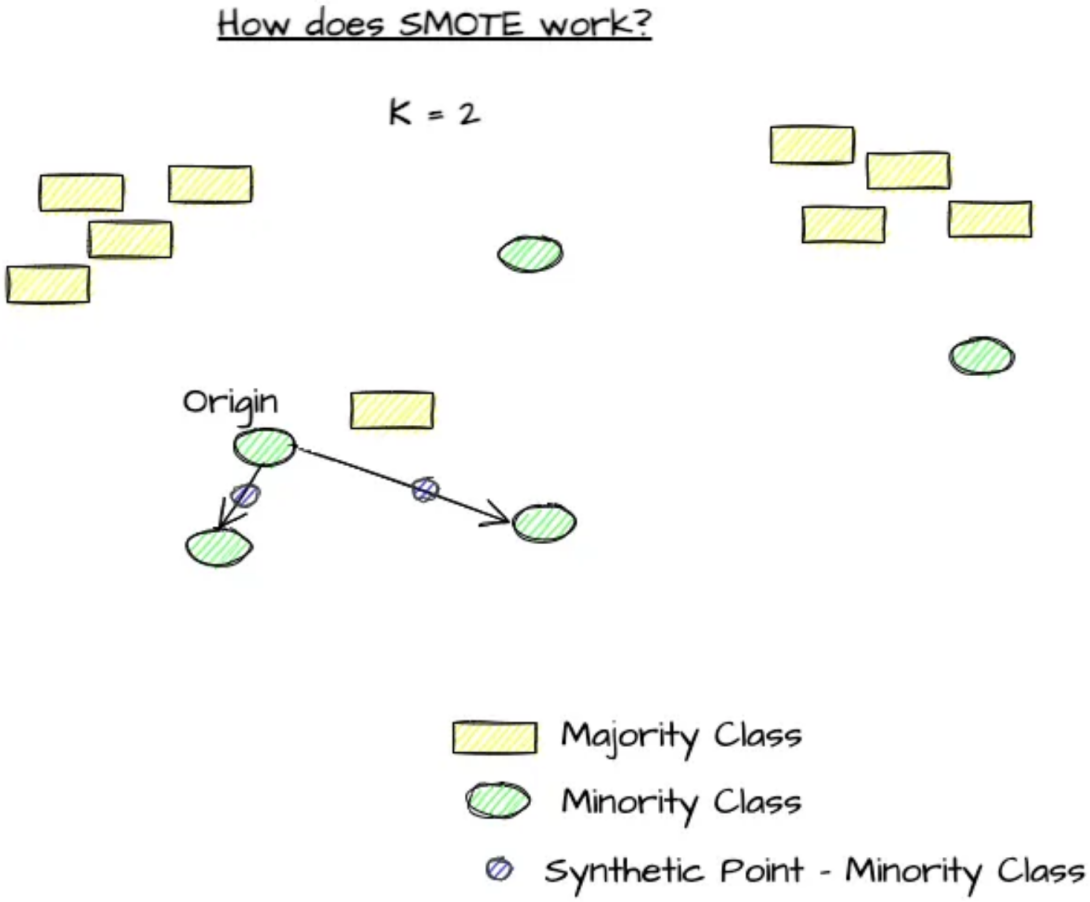
Exercise: manual resampling

- **Solution:**
  - Duplicated P instances: P1, P1, P1, P1, P2, P2, P2, P2
  - New dataset: N1, N2, N3, N4, N5, N6, N7, N8, P1, P1, P1, P1, P2, P2, P2, P2

Synthetic Minority Oversampling Technique (SMOTE)

- advanced oversampling method that generates synthetic examples rather than duplicating existing ones.
- helps address overfitting, which is common under simply duplicating minority class

Basic SMOTE


How does SMOTE work?
K = 2
Origin
Majority Class
Minority Class
Synthetic Point - Minority Class

Basic SMOTE

1. Select a sample, call it O (for Origin), from the minority class randomly
2. Find the K-Nearest Neighbors of O that belong to the same class
3. Connect O to each of these neighbors using a straight line
4. Select a scaling factor z in the range $[0, 1]$ randomly
5. For each new connection, place a new point on the line $(z * 100)\%$ away from O (synthetic samples)
6. Repeat until you get the desired number of synthetic samples

Disadvantage of traditional SMOTE

Disad:

- If the point(s) selected in either steps 1 or 2 are located in a region dominated by majority class samples, the synthetic points might be generated **inside the majority class** region (which may make classification difficult)
- need an improved version of SMOTE

Borderline SMOTE

Similar to basic SMOTE with a few adjustments

To overcome the shortcomings of SMOTE, it identifies two sets of points — noise and border

- **noise** is a point all of whose nearest neighbors belong to a different class (i.e. the majority)
- **border** points have a mix of majority and minority points as their nearest neighbors

Borderline SMOTE

- Choose O as Border
- Find nearest neighbors among points of **any** class (rather than only minority class as in basic SMOTE)
  - this helps select points at risk of misclassification and those closer to the boundary
- everything else same as in basic SMOTE
- borderline SMOTE is not a silver bullet: restricting sampling to border points and relaxing the neighborhood selection criteria need not work in every scenario

K-Means SMOTE

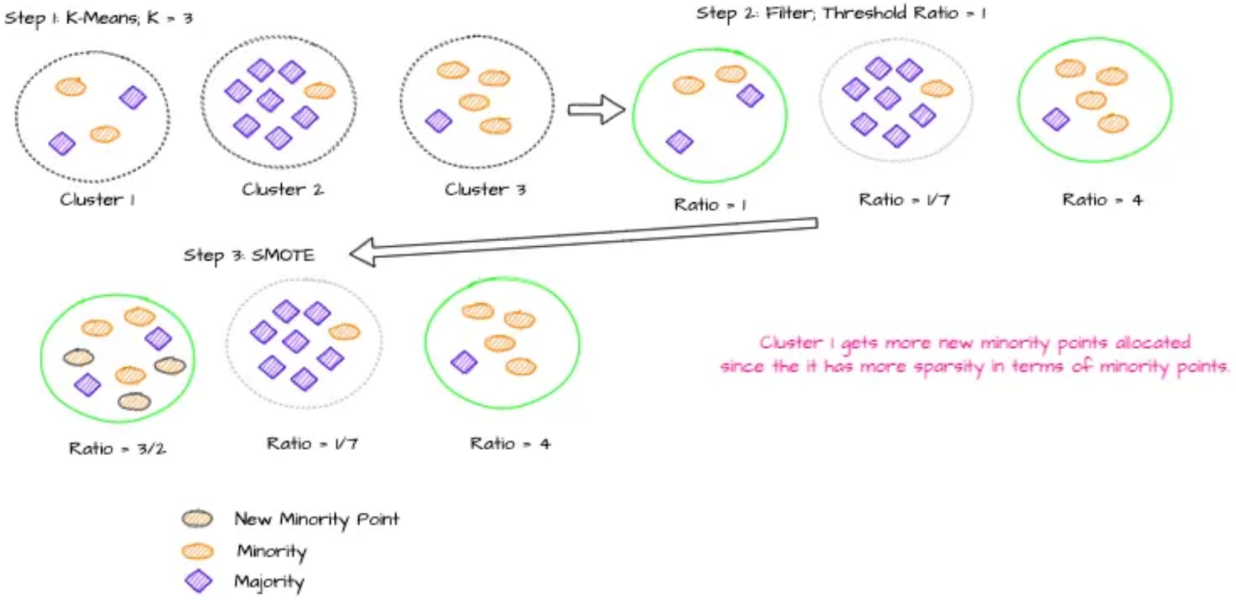
Recent method that aims to eliminate noisy synthetic points generated by other oversampling methods

1. Do K-Means clustering on the data. Select clusters with a high proportion ($>$50% or user-defined) of minority class samples
2. Apply basic SMOTE to these selected clusters. Each cluster will be assigned new synthetic points. The number of these generated points will depend on the sparsity of the minority class in the cluster; the more the sparsity, the more new points

K-means clustering

- unsupervised learning algorithm, i.e., no label y
- idea: group similar data points into K clusters by minimizing the variance within each cluster and maximizing the variance between clusters
- algorithm:
  1. **initialization**: randomly select K centroids, then iteratively adjusts them to minimize the total sum of distances within clusters
  2. **optimization**: iterate by reassigning data points to the nearest centroid and recalculating centroids until they stabilize or max number of iterations is reached
  3. **evaluation**: uses metrics like the silhouette score or the elbow method to determine the optimal number of clusters

K-Means SMOTE



K-Means SMOTE

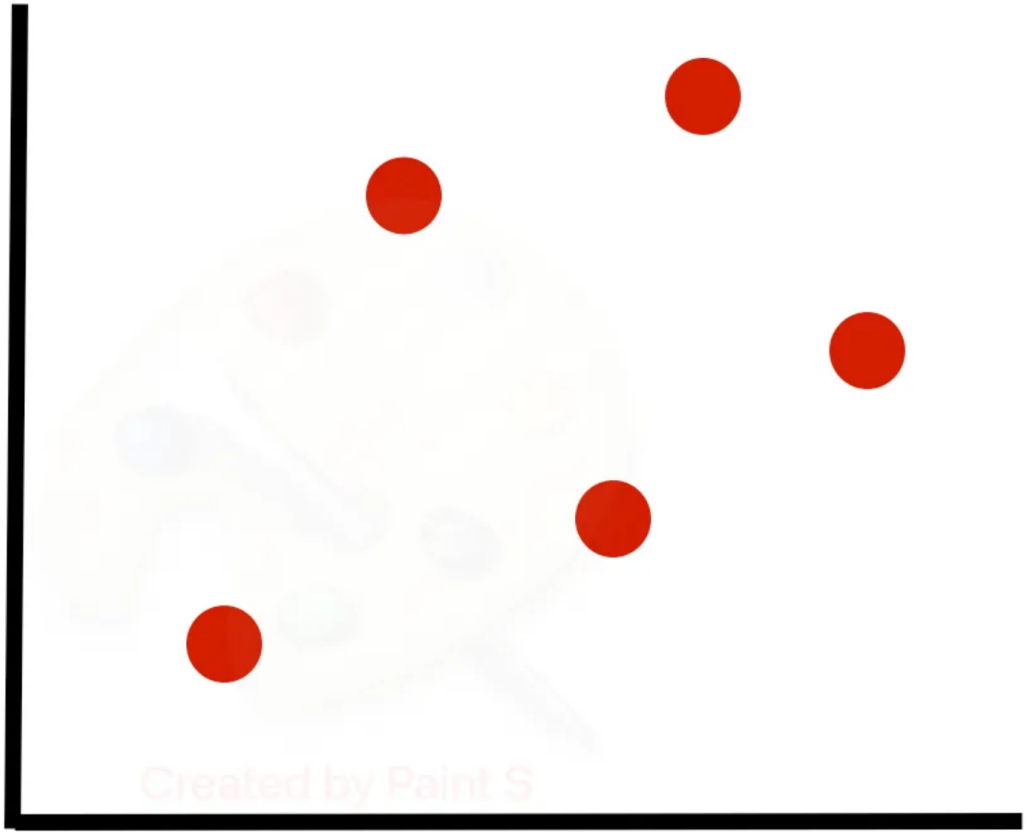
K-Means SMOTE

- Assign new minority samples to cluster 1 not cluster 3 due to a trade-off: the distribution of minority in a cluster should be neither too dense (becomes majority, cluster 3) nor too sparse (not representative to the minority distribution, cluster 2)
- helps create clusters of the minority class (that are not much affected by other classes). This can ultimately benefit the ML model. However, it inherits the weaknesses of K-Means, e.g., choosing K

SMOTE: example

- imbalanced dataset: more rows of a given class than the other
- plot a subset of the rows belonging to the minority class
- we only consider 2 of the many features (x axis / y axis)

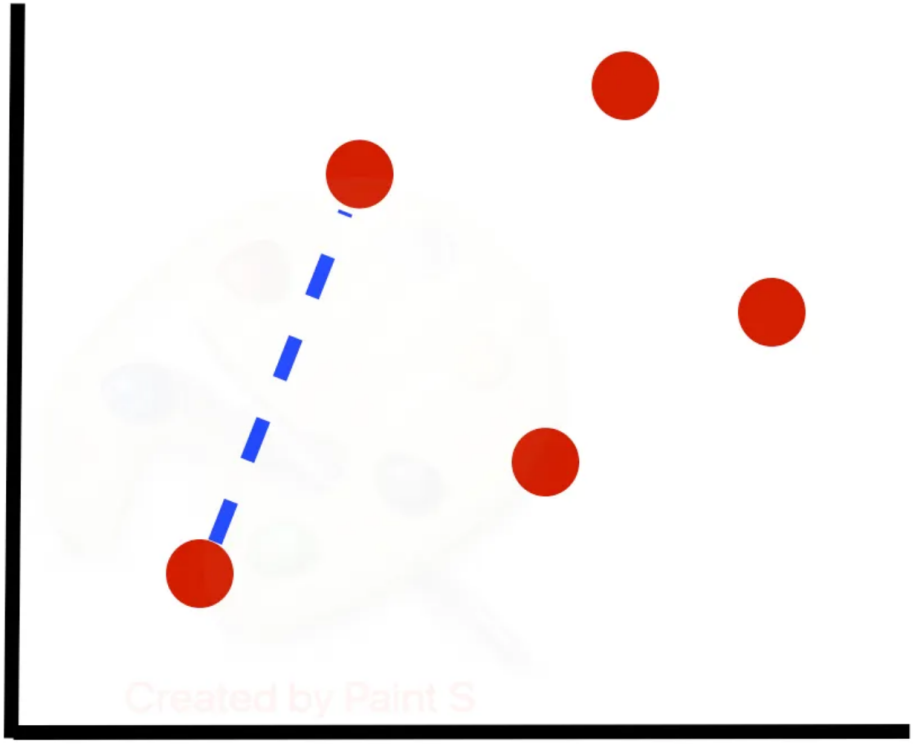
SMOTE: example



SMOTE: example

- consider the first row (or a random row) and compute its k nearest neighbors
- select a random nearest neighbor out of the k nearest neighbors

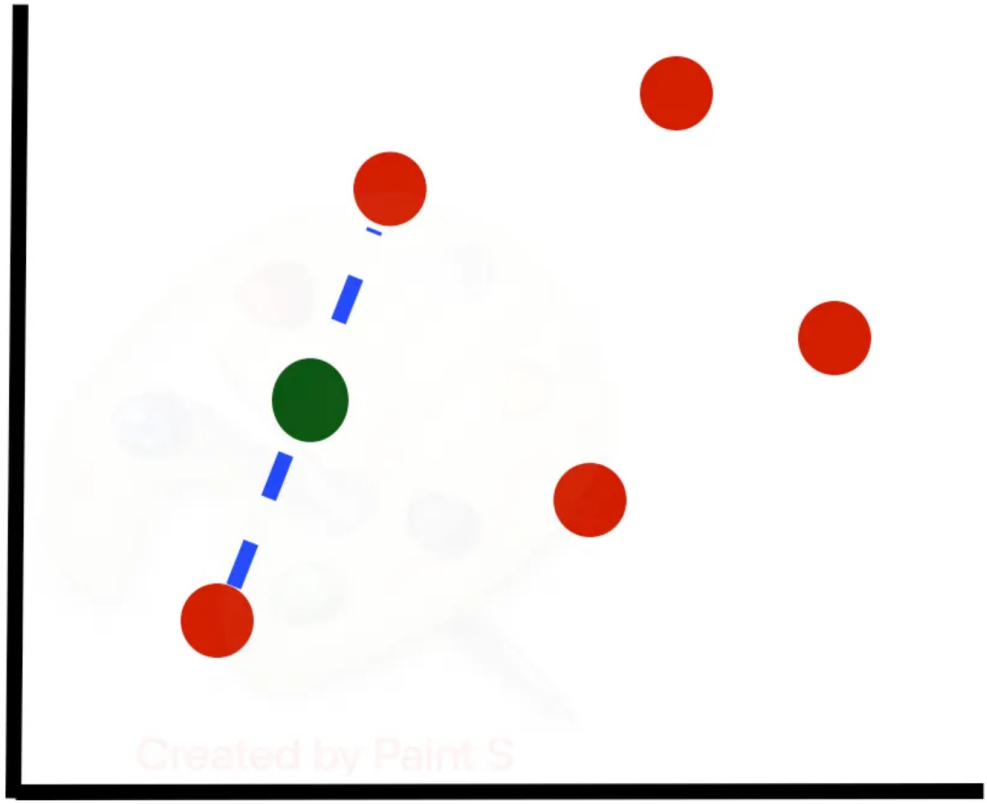
SMOTE: example



SMOTE: example

- compute the difference between the two points and multiply it by a random number between 0 and 1. This gives us a synthetic example along the line between the two points

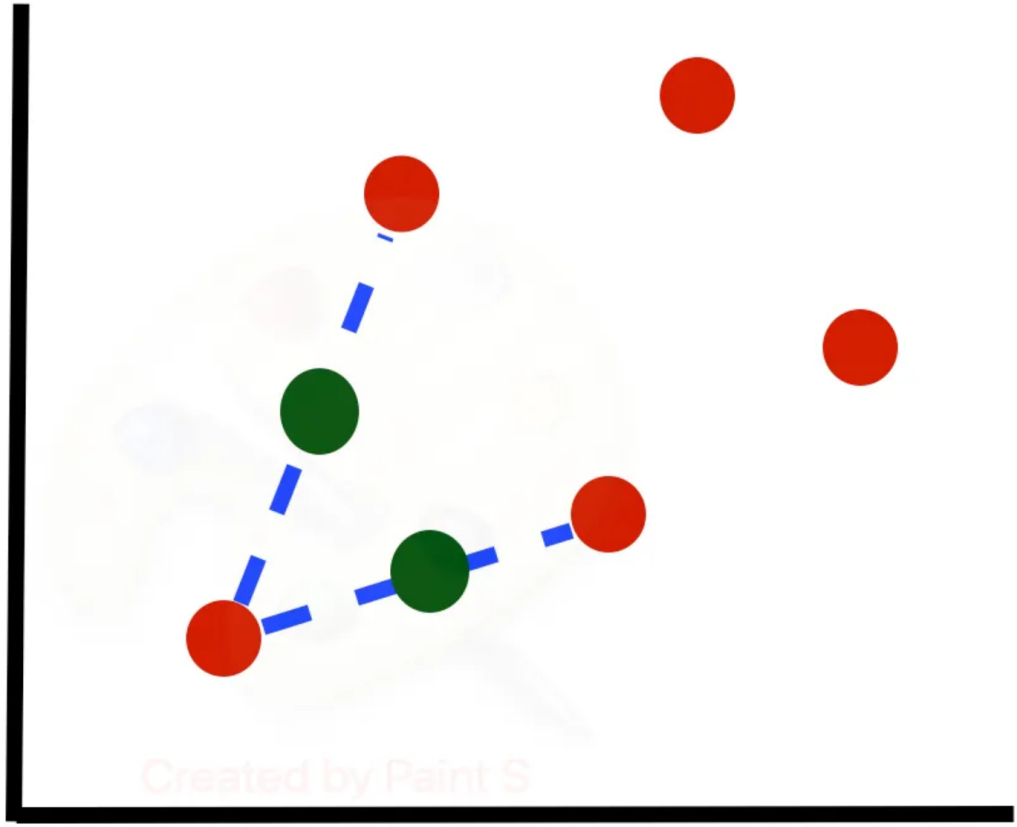
SMOTE: example



SMOTE: example

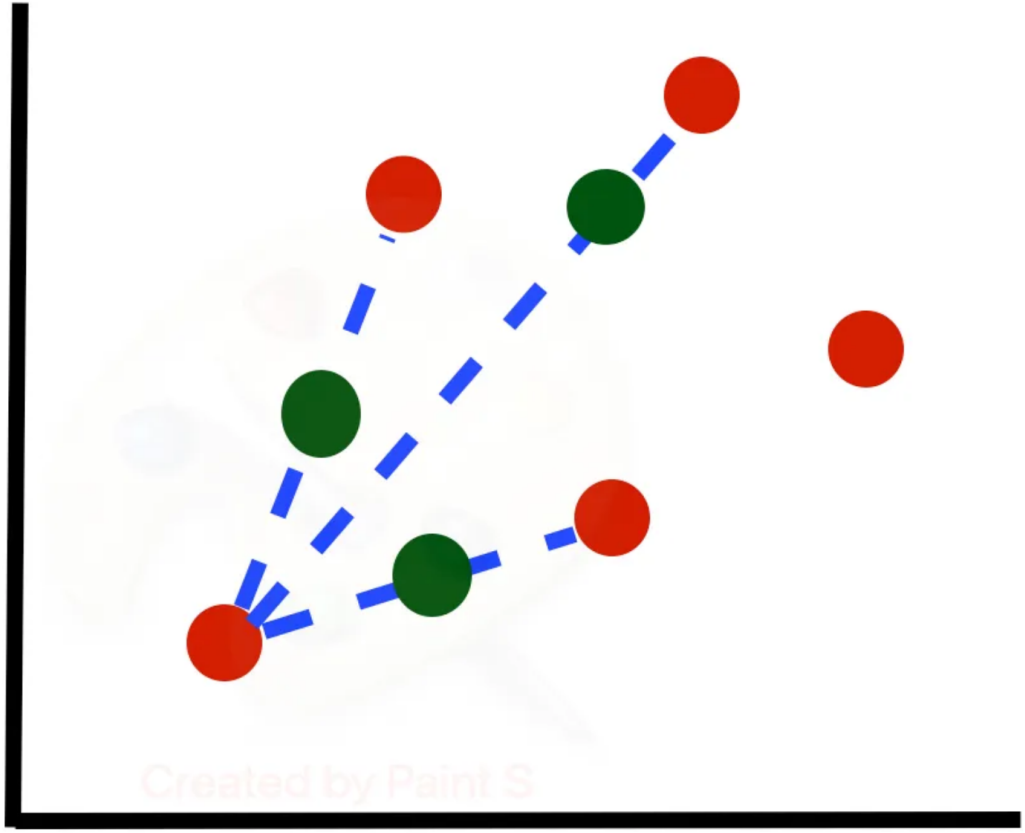
- repeat the process L times
  - L is the over-sampling rate
  - example: if the original minority class has 100 samples and we aim to oversample to achieve 300 samples, then $L = 3$
- In our picture, only $300/100 = 3$ neighbors from the $k = 5$ nearest neighbors are chosen and one sample is generated in the direction of each

SMOTE: example



SMOTE: example

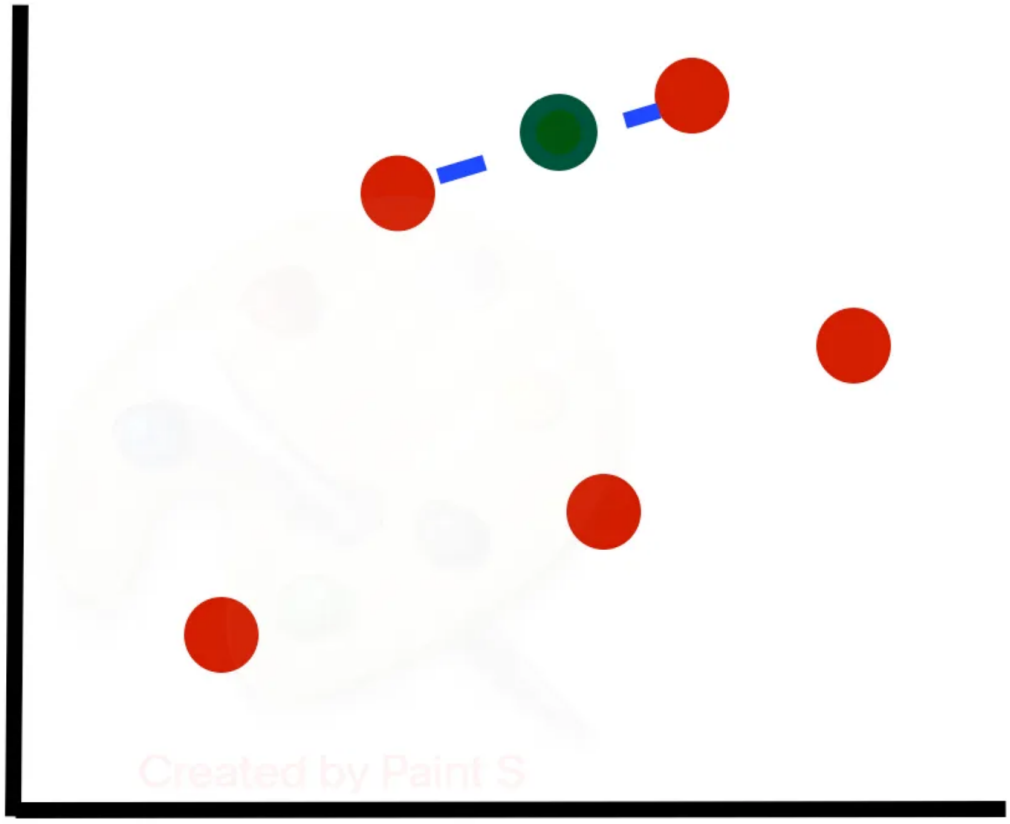
oversample the minority 4 points



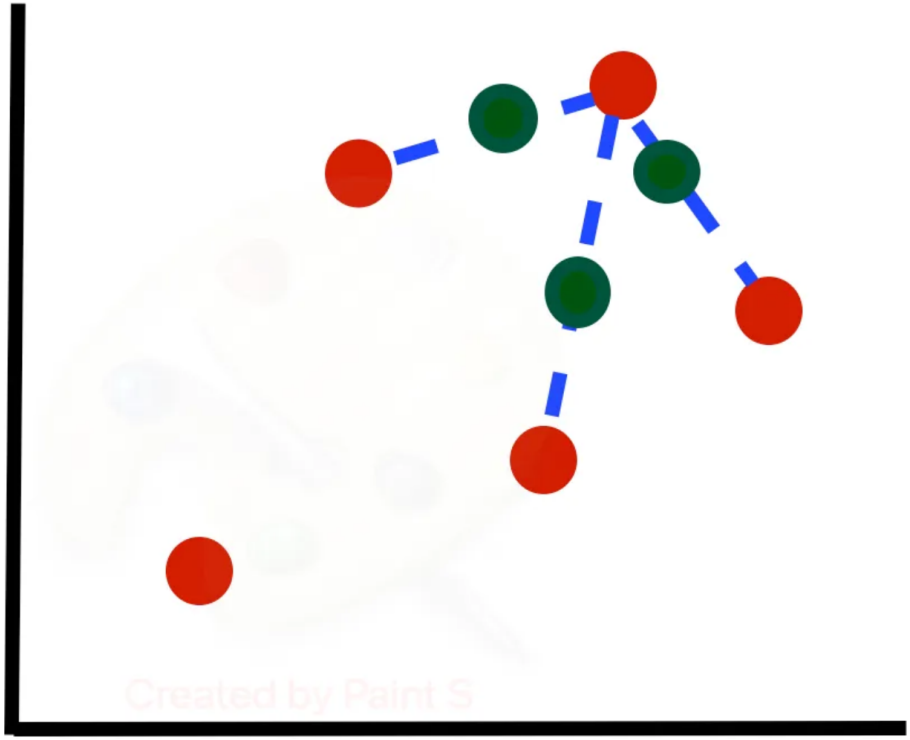
SMOTE: example

- We then move on to the next row, compute its k nearest neighbors, and select $300/100 = 3$ of its nearest neighbors at random to use in generating new synthetic examples

SMOTE: example



SMOTE: example



SMOTE in Python

- let us walk through an example of using SMOTE in Python
- begin by importing the required libraries

SMOTE in Python

```
from random import randrange, uniform
from sklearn.neighbors import NearestNeighbors
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, recall_score
```

SMOTE in Python

- **data**: Credit Card Fraud Detection dataset from Kaggle
- **goal**: train a model to determine whether a given transaction is fraudulent
- read the CSV file and store its contents in a Pandas DataFrame:

SMOTE in Python

```
df = pd.read_csv("creditcard.csv")
```

- no original features due to confidentiality issues, except Time and Amount
- features $V_1, V_2, \cdots, V_{28}$ are the **principal components** obtained with PCA

What is Principal Component Analysis?

- a statistical procedure that uses an **orthogonal transformation** to convert observations of correlated variables into observations of uncorrelated variables called **principal components**
- details later in this class

SMOTE in Python

```
df.head(5)
```

...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

SMOTE in Python

```
df['Class'].value_counts()
Out:
0    284315
1       492
Name: Class, dtype: int64
```

- as we can see, there are many more negative samples (class 0) than positive samples (class 1)

SMOTE in Python

- for simplicity, remove the time dimension

```
df = df.drop(['Time'], axis=1)
```

- split the dataset into features and labels

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

- to evaluate the performance of our model, split the data into training and test sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

SMOTE in Python

- initialize an instance of the RandomForestClassifier class

```
rf = RandomForestClassifier(random_state=42)
```

- fit the model to the training set

```
rf.fit(X_train, y_train)
```

- finally, use the model to predict whether a transaction is fraudulent

```
y_pred = rf.predict(X_test)
```

SMOTE in Python

- confusion matrix (no SMOTE)
- 23 fraudulent transactions misclassified

```
confusion_matrix(y_test, y_pred)
Out:
array([[56862,     2],
       [   23,    75]])
```

SMOTE using imbalanced-learn

- SMOTE is implemented in a separate library imbalanced-learn
- install:

```
pip install imbalanced-learn
```

- import the SMOTE class:

```
from imblearn.over_sampling import SMOTE
```

- to avoid confusion, read the .csv file again:

```
df = pd.read_csv("creditcard.csv")
df = df.drop(['Time'], axis=1)
X = df.drop(['Class'], axis=1)
y = df['Class']
```

- create an instance of the SMOTE class
  - by default, equal number of positive and negative samples

```
sm = SMOTE(random_state=42, k_neighbors=5)
```

- apply SMOTE to data:

```
X_res, y_res = sm.fit_resample(X, y)
```

SMOTE using imbalanced-learn

- again, split the dataset, train the model, and predict whether the samples in the testing dataset are fraudulent

```
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

- confusion matrix: equal number of positive and negative samples and no false negatives
- recall=1

```
confusion_matrix(y_test, y_pred)
Out:
array([[56737,    13],
       [    0, 56976]])
recall_score(y_test, y_pred)
Out: 1.0
```

Empirical comparison of techniques for imbalanced data

- Empirically compare **undersampling, oversampling, and SMOTE**
- Wongvorachan, T., He, S., and Bulut, O. *"A comparison of undersampling, oversampling, and smote methods for dealing with imbalanced classification in educational data mining"*, Information, 14(1):54, 2023

Empirical comparison of techniques for imbalanced data

Recommended procedure:

- **Step 1: prepare the dataset**
  - start with an imbalanced dataset
    - example (educational data mining): predict dropout where the number of dropouts (minority class) is much smaller than the number of students who continue (majority class)
- **Step 2: apply resampling**
  - undersampling: randomly remove samples from the majority class to match the number of samples in the minority class
  - oversampling: randomly duplicate samples in the minority class to match the number of samples in the majority class.
  - SMOTE: generate synthetic samples for the minority class by interpolating between existing minority samples

Empirical comparison of techniques for imbalanced data

- **Step 3: train a classifier**
  - use a common classifier for all three datasets
  - logistic regression, decision trees, or random forests are popular choices
- **Step 4: evaluate model performance**
  - Precision measures the accuracy of positive predictions
  - Recall measures the ability to find positive samples
  - F1 score provides a balance between precision and recall
  - AUC represents the likelihood of distinguishing between classes

Example: educational data mining (Wongvoracha et al. 2023)

Table 4. Classification results for each resampling condition.

	Performance Metrics	Baseline Mean (SD)	ROS Mean (SD)	RUS Mean (SD)	SMOTE-NC + RUS Mean (SD)
Moderately Imbalanced	Accuracy	0.736 (0.009)	0.877 (0.006)	0.705 (0.014)	0.779 (0.011)
	Precision	0.773 (0.006)	0.926 (0.006)	0.724 (0.014)	0.796 (0.013)
	Recall	0.888 (0.007)	0.819 (0.010)	0.662 (0.024)	0.749 (0.018)
	ROC-AUC	0.763 (0.013)	0.968 (0.003)	0.763 (0.015)	0.868 (0.010)
	F1	0.827 (0.005)	0.870 (0.006)	0.692 (0.017)	0.772 (0.013)
Extremely Imbalanced	Accuracy	0.887 (0.004)	0.984 (0.002)	0.731 (0.021)	0.905 (0.006)
	Precision	0.665 (0.058)	0.971 (0.003)	0.736 (0.024)	0.911 (0.008)
	Recall	0.193 (0.028)	0.999 (0.001)	0.721 (0.031)	0.898 (0.008)
	ROC-AUC	0.801 (0.016)	0.999 (0.0003)	0.800 (0.020)	0.967 (0.002)
	F1	0.299 (0.038)	0.985 (0.002)	0.728 (0.022)	0.904 (0.006)

Example: takeaways

- undersampling can effectively balance the classes but may lead to **loss of information**, hampering generalization
- oversampling can improve detection of the minority class but may lead to **overfitting**
- SMOTE provides **good balance** by generating synthetic samples, potentially improving generalization without losing information or overly increasing the risk of overfitting