



UNIVERSITAS AMIKOM
YOGYAKARTA

Program Studi Sistem Informasi | Fakultas Ilmu Komputer

PEMROSESAN DAN VISUALISASI DATA

“Data Imbalanced dan Feature Selection”

Yoga Pristyanto, S.Kom., M.Eng.



UNIVERSITAS AMIKOM
Y O G Y A K A R T A

OUTLINE

- Data Imbalanced
- Feature Selection



UNIVERSITAS AMIKOM
YOGYAKARTA

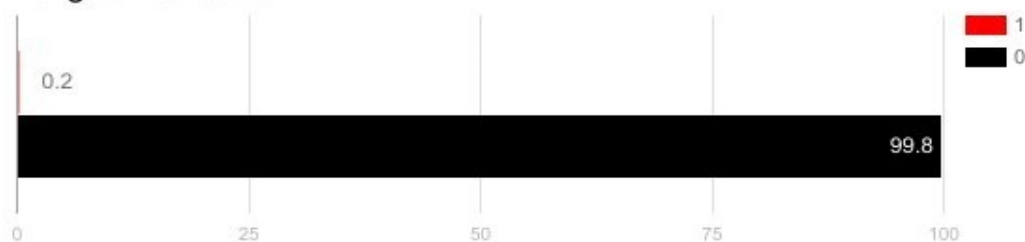
DATA IMBALANCED



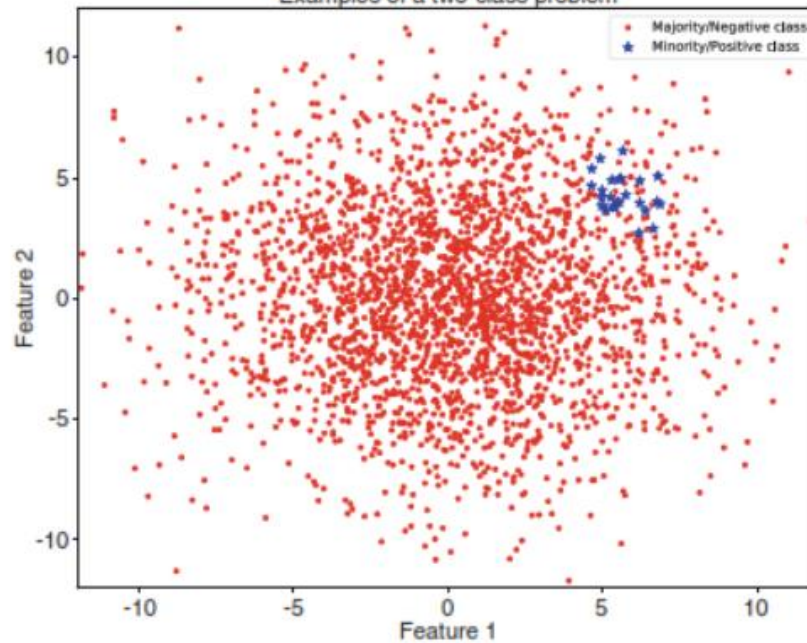
APA ITU IMBALANCED DATA ?

- Imbalanced Dataset adalah sebuah kondisi dataset dalam classification task dimana proporsi dari label (biasa juga disebut kelas atau target) yang dimiliki sangat timpang jauh.
- Misalkan dalam kasus binary classification label 0 dan 1 masing-masing memiliki proporsi 10% dan 90%.
- Dalam contoh tersebut, label yang memiliki proporsi besar disebut label mayor (majority classes) dan lainnya disebut label minor (minority classes).
- Terdapat selisih proporsi yang sangat jauh dari contoh kasus tersebut.
- Idealnya, dataset dalam classification task hendaklah memiliki proporsi label yang berimbang.

Target Distribution



Examples of a two-class problem





MENGAPA IMBALANCED DATA PENTING UNTUK DIPERHATIKAN ?

Imbalanced class dapat memiliki dampak serius pada analisis data dan pembelajaran mesin:

- **Bias Model:** Model pembelajaran mesin dapat menjadi bias terhadap kelas mayoritas karena memiliki lebih banyak contoh untuk mempelajari pola dari kelas tersebut. Model cenderung memprediksi mayoritas, mengorbankan akurasi kelas minoritas.
- **Kinerja Model Buruk:** Kinerja model dalam mengidentifikasi kelas minoritas (misalnya, deteksi penipuan) mungkin buruk karena kurangnya data pelatihan yang mencukupi. Model mungkin gagal mengenali kasus penting.
- **Real World Problem:** Dalam banyak kasus nyata, kelas minoritas seringkali adalah yang paling penting (misalnya, deteksi penyakit langka). Kesalahan dalam mengidentifikasi kelas minoritas dapat memiliki konsekuensi serius.



KATEGORI IMBALANCED DATA

Berdasarkan proporsinya, ada tiga tingkatan mengenai imbalanced dataset diukur dari minority class-nya.

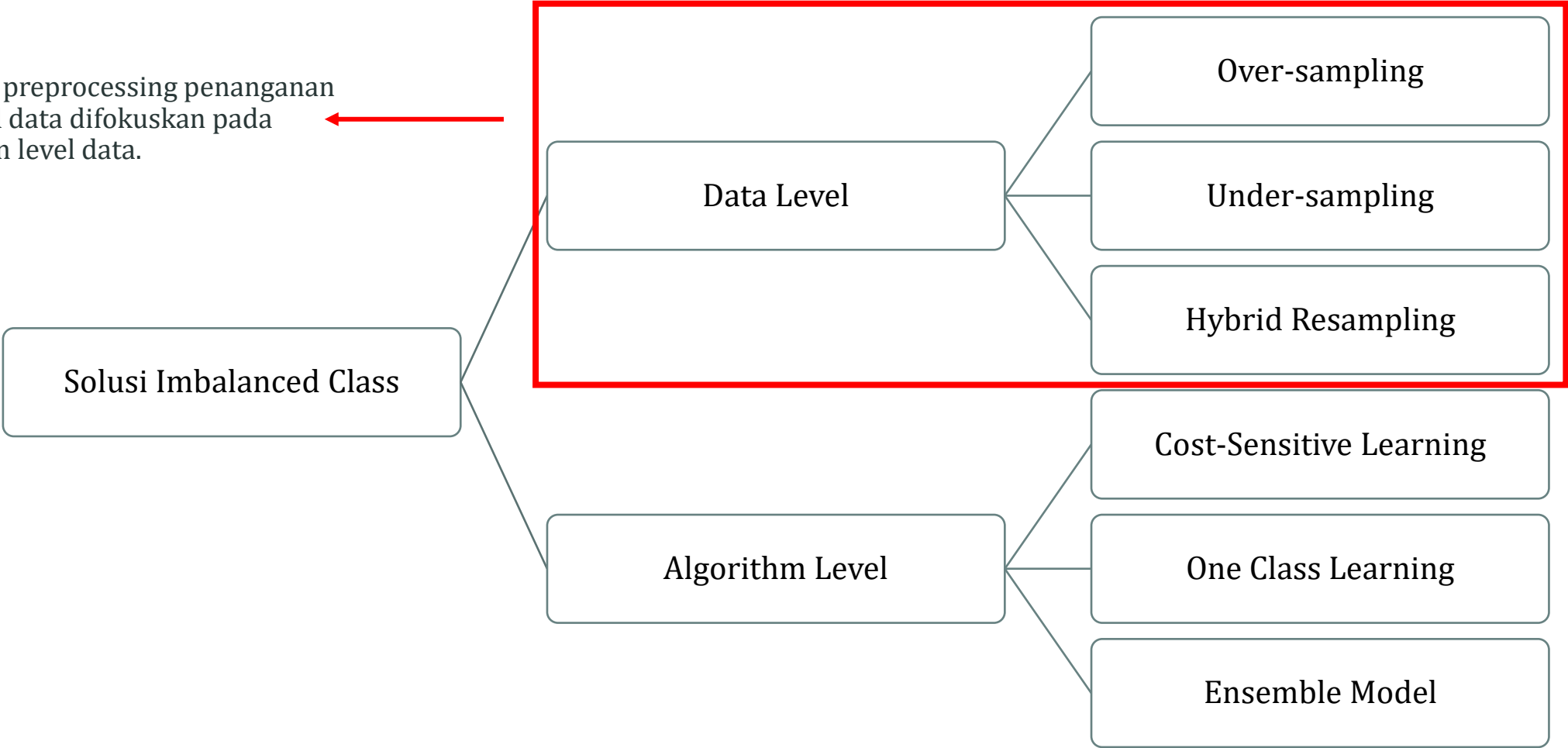
Degree of imbalance	Proportion of Minority Class
Mild	20-40% of the data set
Moderate	1-20% of the data set
Extreme	<1% of the data set

<https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data>



PENANGANAN IMBALANCED DATA

Pada tahap preprocessing penanganan imbalanced data difokuskan pada penggunaan level data.





PENANGANAN IMBALANCED DATA : DATA LEVEL

- Pendekatan level data biasanya sering dikenal dengan istilah resampling.
- Resampling adalah teknik dalam analisis data yang melibatkan proses pengubahan atau manipulasi kumpulan data asli untuk menghasilkan dataset baru yang memiliki karakteristik yang berbeda.
- Teknik ini sering digunakan dalam konteks mengatasi masalah ketidakseimbangan kelas (imbalance class) dalam pembelajaran mesin atau dalam pengujian statistik.
- Resampling dibagi menjadi tiga jenis utama yaitu:
 1. Oversampling
 2. Undersampling
 3. Hybrid Resampling



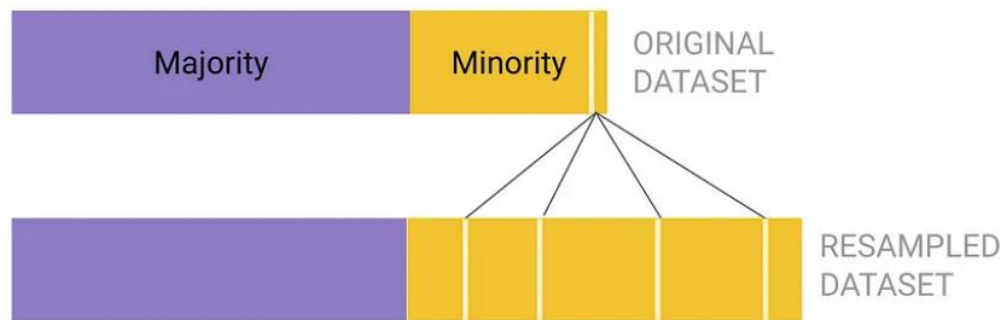
TEKNIK OVERSAMPLING

Definisi:

- Oversampling adalah salah satu teknik resampling yang digunakan dalam analisis data untuk mengatasi masalah ketidakseimbangan kelas pada dataset.
- Teknik ini melibatkan peningkatan jumlah sampel dalam kelas minoritas dengan beberapa cara, sehingga memungkinkan model pembelajaran mesin untuk mempelajari pola dengan lebih baik dari kelas minoritas.

Tujuan Oversampling:

- Tujuan utama dari oversampling adalah menciptakan keseimbangan antara jumlah sampel antara kelas mayoritas (kelas yang lebih banyak) dan kelas minoritas (kelas yang lebih sedikit) dalam dataset.
- Dengan demikian, oversampling dapat membantu mencegah bias yang mungkin muncul dalam model pembelajaran mesin yang cenderung memprediksi mayoritas.





TEKNIK OVERSAMPLING

Pro:

- Meningkatkan kinerja model dalam mengklasifikasikan kelas minoritas.
- Menghindari bias dalam model yang disebabkan oleh ketidakseimbangan kelas.

Kontra:

- Dapat mengakibatkan overfitting, terutama jika oversampling dilakukan secara berlebihan.
- Mengakibatkan peningkatan jumlah data, yang dapat mengakibatkan waktu pelatihan yang lebih lama dan penggunaan sumber daya yang lebih besar.



TEKNIK OVERSAMPLING

Metode Oversampling: Ada tiga metode oversampling yang umum dan dapat digunakan, antara lain:

1. Random Oversampling (ROS)
2. Synthetic Minority Oversampling Techniques (SMOTE)
3. Adaptive Synthetic Sampling (Adasyn)



RANDOM OVERSAMPLING (ROS)

Random Oversampling (ROS):

Konsep dasar di balik random oversampling adalah meningkatkan jumlah sampel dalam kelas minoritas dengan cara menggandakan (duplikat) sampel-sampel yang sudah ada dari kelas tersebut.

Dengan kata lain, teknik ini secara acak memilih dan menyalin sejumlah sampel dari kelas minoritas sehingga jumlahnya seimbang dengan kelas mayoritas.

Tahapan Random Oversampling (ROS):

- **Tahap pertama** pemilihan dataset kemudian hitung kelas mayoritas dan kelas minoritas.
- **Tahap kedua** menghitung nilai selisih antara kelas mayoritas dengan kelas minoritas, setelah diperoleh nilai selisih antara kelas mayoritas dan kelas minoritas, kelompokkan record kelas minoritas.
- **Tahap ketiga** tambahkan record kelas minoritas satu per satu secara acak ke dalam data latih kelas minoritas dengan cara menduplikasi record data kelas minoritas. Penambahan record ini diulang sesuai dengan jumlah dari selisih, sehingga jumlah data kelas minoritas akan sama dengan jumlah data kelas mayoritas.



HANDS-ON: RANDOM OVERSAMPLING (ROS)

```
✓ 1s ▶ from collections import Counter  
from sklearn.datasets import make_classification  
from imblearn.over_sampling import RandomOverSampler
```

→ Import library yang diperlukan.

```
▶ X, y = make_classification(n_samples=10000, weights=[0.85], flip_y=0)  
print("Distribusi kelas:", Counter(y))
```

→ Membuat dataset sederhana sejumlah 10000 dataset dengan kondisi imbalanced dengan proporsi 85% : 15%.

```
Distribusi kelas: Counter({0: 8500, 1: 1500})
```

→ Output

```
▶ # Distribusi kelas sebelum dilakukan oversampling  
print("Sebelum oversampling:", Counter(y))  
  
# Metode Random Oversampling  
oversample = RandomOverSampler(sampling_strategy='minority')  
X_over, y_over = oversample.fit_resample(X, y)  
  
# Distribusi kelas setelah dilakukan oversampling  
print("Setelah oversampling:", Counter(y_over))
```

→ Proses resampling menggunakan Random Oversampling

```
Sebelum oversampling: Counter({0: 8500, 1: 1500})  
Setelah oversampling: Counter({0: 8500, 1: 8500})
```

→ Output



SYNTHETIC MINORITY OVERSAMPLING TECHNIQUES (SMOTE)

Synthetic Minority Over-sampling Technique (SMOTE) adalah teknik oversampling data untuk mengatasi masalah ketidakseimbangan kelas dalam dataset. SMOTE bekerja dengan cara menciptakan sampel sintetis untuk kelas minoritas sehingga bisa seimbang dengan kelas mayoritas.

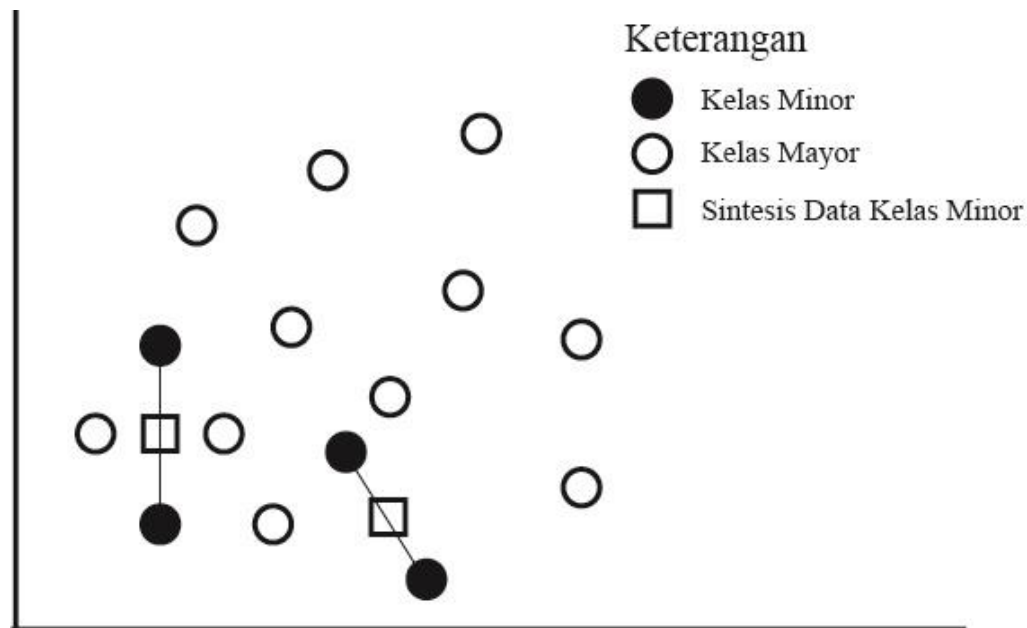
Tahapan SMOTE:

- ***Memilih Sampel:*** SMOTE memilih secara acak satu contoh dari kelas minoritas.
- ***Mencari Tetangga:*** Kemudian mengidentifikasi tetangga terdekat k (umumnya $k=3$ atau $k=5$) dari contoh yang dipilih di antara sampel-sampel kelas minoritas.
- ***Menciptakan Sampel Sintetis:*** SMOTE menghasilkan sampel sintetis dengan interpolasi antara contoh yang dipilih dan tetangganya. Ini dilakukan dengan memilih nilai-nilai acak untuk setiap fitur dan mengaplikasikannya pada nilai-nilai fitur contoh yang dipilih. setelah itu dibuat sintetis data sebanyak persentase duplikasi yang diinginkan antara data minor dan k -nearest neighbors yang dipilih secara random.

Pembangkitan data sintetis yang berskala numerik berbeda dengan kategorik. Data numerik diukur jarak kedekatannya dengan jarak Euclidean sedangkan data kategorik lebih sederhana yaitu dengan nilai modus.



SYNTHETIC MINORITY OVERSAMPLING TECHNIQUES (SMOTE)



Sumber:

- Y. Pristyanto and A. Dahlan, "Hybrid Resampling for Imbalanced Class Handling on Web Phishing Classification Dataset," 2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, 2019, pp. 401-406, doi: 10.1109/ICITISEE48480.2019.9003803.
- Y. Pristyanto, I. Pratama and A. F. Nugraha, "Data level approach for imbalanced class handling on educational data mining multiclass classification," 2018 International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 2018, pp. 310-314, doi: 10.1109/ICOIACT.2018.8350792.



HANDS-ON: SYNTHETIC MINORITY OVERSAMPLING TECHNIQUES (SMOTE)

Install Library

```
! pip install imbalanced-learn
```

Import Library

```
from collections import Counter  
from sklearn.datasets import make_classification  
from imblearn.over_sampling import SMOTE
```

Generate Data

```
X, y = make_classification(n_samples=10000, weights=[0.95], flip_y=0)  
print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({0: 9500, 1: 500})
```

Implementasi SMOTE

```
# Distribusi kelas sebelum dilakukan oversampling  
print("Sebelum oversampling:", Counter(y))  
  
# Metode Random Oversampling  
oversample = SMOTE()  
X_smote, y_smote = oversample.fit_resample(X, y)  
  
# Distribusi kelas setelah dilakukan oversampling  
print("Sesudah oversampling SMOTE:", Counter(y_smote))
```

Output

```
Sebelum oversampling: Counter({0: 9500, 1: 500})  
Sesudah oversampling SMOTE: Counter({0: 9500, 1: 9500})
```




ADAPTIVE SYNTHETIC SAMPLING (ADASYN)

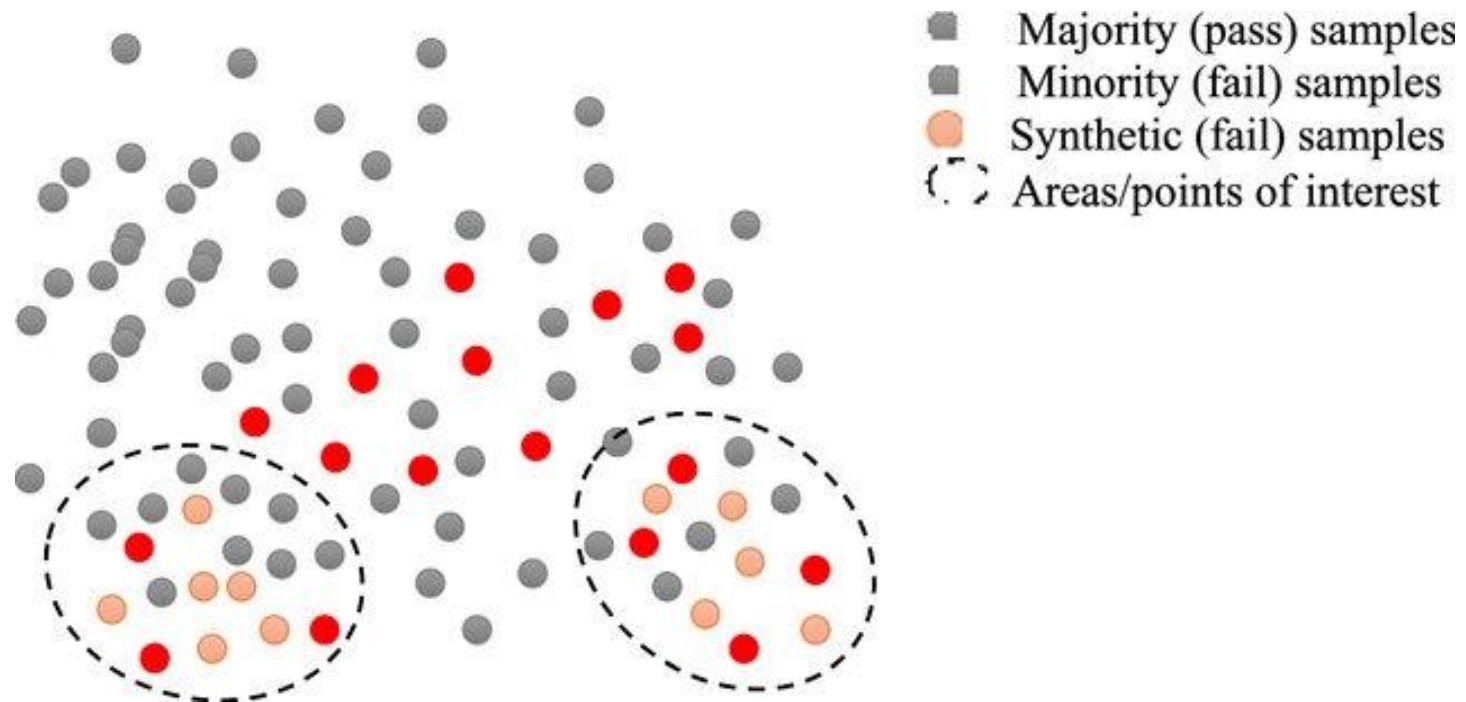
Adaptive Synthetic Sampling (ADASYN) adalah teknik yang digunakan untuk mengatasi masalah ketidakseimbangan kelas dalam dataset pada tugas klasifikasi. Teknik ini secara adaptif menghasilkan sampel sintetis untuk kelas minoritas dengan fokus pada sampel-sampel yang lebih sulit untuk diklasifikasikan (borderline sample).

Tahapan ADASYN:

- ***Menghitung Tingkat Kesulitan (Difficulty Level):*** ADASYN pertama-tama menghitung tingkat kesulitan (difficulty level) untuk setiap sampel dalam kelas minoritas. Tingkat kesulitan ini mencerminkan sejauh mana sampel tertentu berjarak dari batas keputusan (decision boundary) yang memisahkan kelas minoritas dan mayoritas. Sampel-sampel yang lebih dekat dengan batas keputusan dianggap lebih sulit.
- ***Menentukan Faktor Oversampling:*** Berdasarkan tingkat kesulitan, ADASYN menentukan faktor oversampling untuk setiap sampel dalam kelas minoritas. Faktor ini menunjukkan seberapa banyak sampel sintetis yang harus dihasilkan untuk contoh tersebut. Sampel-sampel yang lebih sulit mendapatkan faktor oversampling yang lebih tinggi.
- ***Menghasilkan Sampel Sintetis:*** ADASYN menghasilkan sampel sintetis untuk setiap sampel dalam kelas minoritas. Sampel sintetis ini dibuat dengan cara menginterpolasi antara sampel yang dipilih dan beberapa sampel tetangganya. Ini berarti nilai-nilai fitur dari sampel sintetis dihasilkan dengan mengambil rata-rata tertimbang dari nilai-nilai fitur sampel asli dan tetangganya.



ADAPTIVE SYNTHETIC SAMPLING (ADASYN)



Sumber:

- Nuhu, A.A., Zeeshan, Q., Safaei, B. et al. Machine learning-based techniques for fault diagnosis in the semiconductor manufacturing process: a comparative study. J Supercomput 79, 2031–2081 (2023). <https://doi.org/10.1007/s11227-022-04730-x>.
- Y. Pristyanto, A. F. Nugraha, A. Dahlan, L. A. Wirasakti, A. Ahmad Zein and I. Pratama, "Multiclass Imbalanced Handling using ADASYN Oversampling and Stacking Algorithm," 2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM), Seoul, Korea, Republic of, 2022, pp. 1-5, doi: 10.1109/IMCOM53663.2022.9721632.



HANDS-ON: ADAPTIVE SYNTHETIC SAMPLING (ADASYN)

Import Library

```
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import ADASYN
```

Generate Data

```
X, y = make_classification(n_samples=10000, weights=[0.75], flip_y=0)
print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({0: 7500, 1: 2500})
```

Implementasi ADASYN

```
# Distribusi kelas sebelum dilakukan oversampling
print("Sebelum oversampling:", Counter(y))

# Metode Random Oversampling
oversample = ADASYN()
X_adasyn, y_adasyn = oversample.fit_resample(X, y)

# Distribusi kelas setelah dilakukan oversampling
print("Sesudah oversampling ADASYN:", Counter(y_adasyn))
```

Output

```
Sebelum oversampling: Counter({0: 7500, 1: 2500})
Sesudah oversampling ADASYN: Counter({0: 7500, 1: 7226})
```



TEKNIK UNDERSAMPLING

Definisi:

- Undrrsampling adalah salah satu teknik resampling yang digunakan dalam analisis data untuk mengatasi masalah ketidakseimbangan kelas pada dataset.
- Teknik ini melibatkan pengurangan jumlah sampel dalam kelas mayoritas dengan beberapa cara untuk menyamakan jumlahnya dengan kelas minoritas (kelas yang memiliki lebih sedikit sampel).

Tujuan Undersampling:

- Tujuan utama undersampling adalah menyamakan jumlah sampel antara kelas minoritas dan mayoritas.
- Hal ini dilakukan dengan mengurangi jumlah sampel dari kelas mayoritas sehingga kelas-kelas tersebut memiliki distribusi yang lebih seimbang.





TEKNIK UNDERSAMPLING

Pro:

- Mengurangi beban komputasi: Dengan mengurangi jumlah sampel, pelatihan model dapat menjadi lebih cepat karena model harus memproses lebih sedikit data.
- Mengurangi risiko overfitting: Mengurangi jumlah sampel kelas mayoritas dapat membantu menghindari overfitting, terutama jika dataset sangat tidak seimbang.

Kontra:

- Kehilangan informasi: Menghapus sejumlah besar sampel dari kelas mayoritas dapat menghilangkan informasi yang berharga yang mungkin diperlukan oleh model untuk melatih dengan baik.
- Potensi bias: Jika tidak dilakukan dengan hati-hati, pengurangan sampel kelas mayoritas dapat menyebabkan model menjadi bias terhadap kelas minoritas.



TEKNIK UNDERSAMPLING

Metode Undersampling: Ada empat metode undersampling yang umum dan dapat digunakan, antara lain:

1. Random Undersampling (RUS)
2. Condensed Nearest Neighbor Rule
3. Tomek Links
4. One Sided-Selection (OSS)



RANDOM UNDERSAMPLING (RUS)

Random Undersampling (RUS):

Konsep dasar di balik random undersampling adalah menurunkan jumlah sampel dalam kelas mayoritas dengan mengurangi sampel-sampel yang sudah ada dari kelas tersebut secara acak.

Tahapan Random Undersampling (RUS):

- **Identifikasi Ketidakseimbangan:** Langkah pertama adalah mengidentifikasi ketidakseimbangan kelas dalam dataset. Hal ini diperlukan untuk mengetahui berapa banyak sampel yang ada dalam kelas mayoritas dan kelas minoritas.
- **Penentuan Jumlah Sampel yang Harus Dihapus:** Perlu ditentukan berapa banyak sampel dari kelas mayoritas yang harus dihapus agar jumlahnya seimbang dengan kelas minoritas. Ini dapat dilakukan dengan berbagai cara, seperti menghapus sejumlah sampel secara acak atau dengan mempertimbangkan rasio antara kelas mayoritas dan kelas minoritas.
- **Pemilihan Sampel yang Akan Dihapus:** Setelah jumlah sampel yang akan dihapus ditentukan, sampel-sampel dari kelas mayoritas dipilih secara acak untuk dihapus. Pemilihan ini harus dilakukan secara acak untuk memastikan representasi yang adil dari kelas mayoritas yang tersisa.
- **Penghapusan Sampel:** Sampel-sampel yang dipilih secara acak dari kelas mayoritas dihapus dari dataset. Ini akan mengurangi jumlah sampel dalam kelas mayoritas.



HANDS-ON: RANDOM UNDERSAMPLING (RUS)

Import Library

```
✓ 0s ▶ from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
```

Generate Data

```
✓ 0s ▶ X, y = make_classification(n_samples=10000, weights=[0.85], flip_y=0)

print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({0: 8500, 1: 1500})
```

Implementasi RUS

```
✓ 0s ▶ # Distribusi kelas sebelum dilakukan undersampling
print("Sebelum undersampling:", Counter(y))

# Metode Random Undersampling
undersample = RandomUnderSampler(sampling_strategy='majority')
X_rus, y_rus = undersample.fit_resample(X, y)

# Distribusi kelas setelah dilakukan undersampling
print("Sesudah undersampling:", Counter(y_rus))
```

Output

```
Sebelum undersampling: Counter({0: 8500, 1: 1500})
Sesudah undersampling: Counter({0: 1500, 1: 1500})
```




CONDENSED NEAREST NEIGHBOR RULE

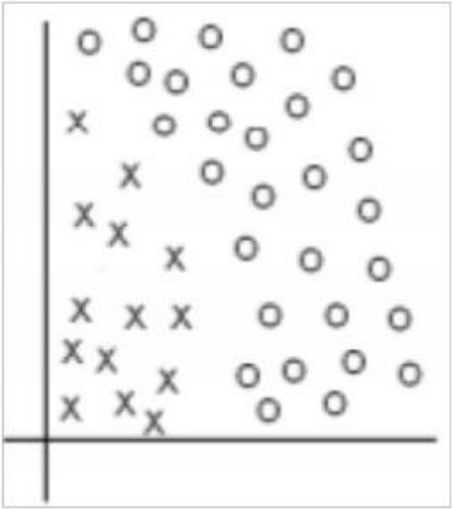
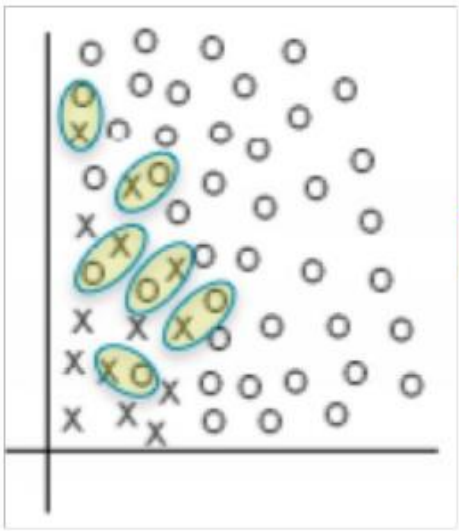
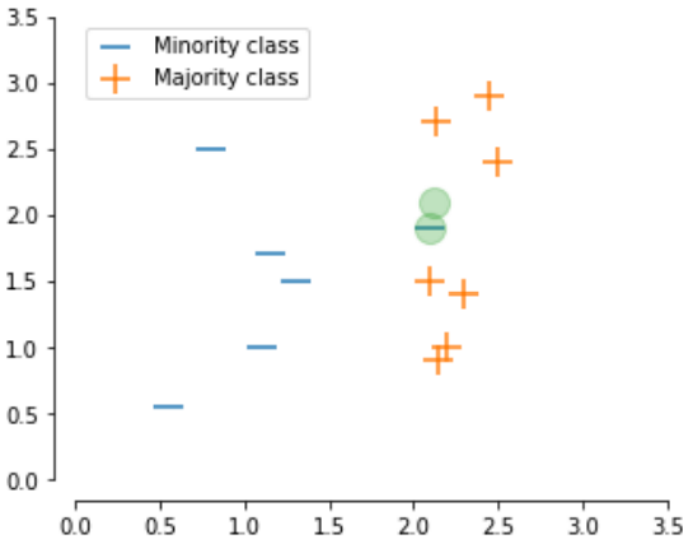
Condensed Nearest Neighbor undersampling (CNN) adalah salah satu teknik pengurangan data yang digunakan dalam pengolahan data dan pembelajaran mesin. Tujuannya adalah mengurangi jumlah sampel dalam dataset sambil mempertahankan representasi yang baik dari semua kelas yang ada.

Tahapan CNN Rule:

- **Inisialisasi:** Langkah pertama adalah menginisialisasi dataset dengan dataset yang kosong atau hanya berisi satu sampel.
- **Iterasi:** CNN Rule bekerja dalam serangkaian iterasi. Pada setiap iterasi, satu sampel dari dataset asli dipilih secara acak.
- **Prediksi:** Sampel yang dipilih di atas digunakan untuk memprediksi kelasnya dengan menggunakan algoritma klasifikasi, seperti K-Nearest Neighbors (K-NN).
- **Pertimbangan:** Jika sampel tersebut diprediksi dengan benar dan dapat dijelaskan oleh sampel-sampel yang sudah ada dalam dataset yang telah dikondensasi sebelumnya, maka sampel tersebut diabaikan (tidak dimasukkan ke dataset yang dikondensasi). Jika sampel tersebut salah diprediksi atau tidak dapat dijelaskan oleh sampel-sampel yang sudah ada, maka sampel tersebut dimasukkan ke dalam dataset yang dikondensasi sebagai representasi tambahan.
- **Iterasi Lanjutan:** Langkah 3 hingga 4 diulang untuk beberapa iterasi sampai tidak ada sampel tambahan yang dapat memperbaiki representasi kelas dalam dataset yang dikondensasi.
- **Hasil Akhir:** Dataset yang dikondensasi adalah hasil akhir dari proses ini. Ini akan menjadi subset yang lebih kecil dari dataset asli, tetapi seharusnya mempertahankan sebagian besar variasi kelas.



CONDENSED NEAREST NEIGHBOR RULE





HANDS-ON: CONDENSED NEAREST NEIGHBOR RULE

Import Library

```
✓ 1s ▶ from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import NeighbourhoodCleaningRule
```

Generate Data

```
✓ 0s ▶ X, y = make_classification(n_samples=10000, weights=[0.75], flip_y=0)

print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({0: 7500, 1: 2500})
```

Implementasi Condensed Nearest Neighbor

```
✓ 1s ▶ # Distribusi kelas sebelum dilakukan undersampling
print("Sebelum undersampling:", Counter(y))

# Metode Random Undersampling
undersample = NeighbourhoodCleaningRule(n_neighbors=5, threshold_cleaning=0.75)
X_cnnr, y_cnnr = undersample.fit_resample(X, y)

# Distribusi kelas setelah dilakukan undersampling
print("Sesudah undersampling CNNR:", Counter(y_cnnr))
```

Output

```
Sebelum undersampling: Counter({0: 7500, 1: 2500})
Sesudah undersampling CNNR: Counter({0: 7055, 1: 2500})
```



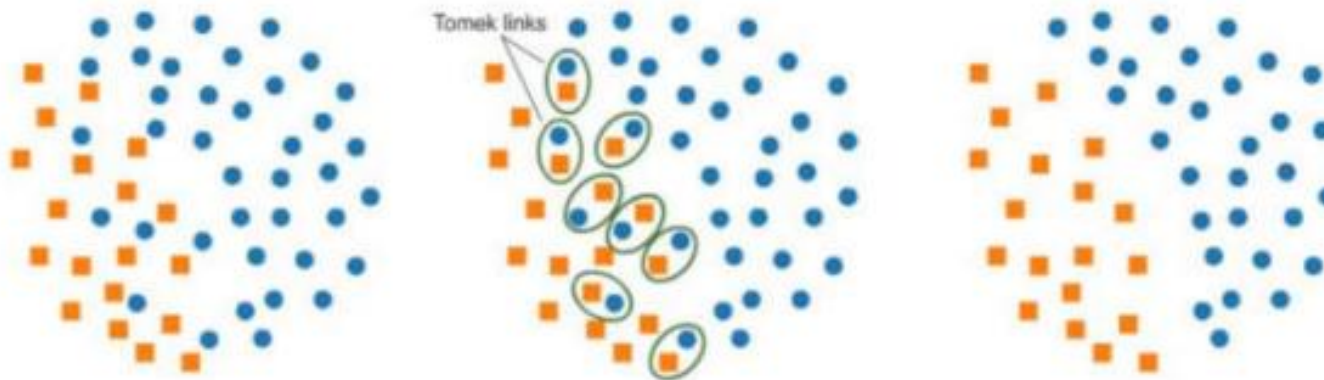
TOMEK LINKS

- Tomek Links adalah pasangan dari dua sampel yang berbeda kelas dan memiliki jarak antara keduanya yang sangat kecil. Mereka mewakili situasi di mana dua sampel dari kelas yang berbeda sangat dekat satu sama lain dalam ruang fitur.
- Konsep utama di balik Tomek Links Undersampling adalah mengidentifikasi dan menghapus sampel-sampel yang terlibat dalam pasangan Tomek Links ini dari kelas mayoritas. Ini bertujuan untuk mengurangi ketidakseimbangan kelas dengan menghilangkan sebagian sampel yang mungkin mengganggu model.



TOMEK LINKS

Cara kerja Tomek Link adalah dengan menghapus data minor ataupun mayor yang memiliki kesamaan karakteristik. Untuk setiap data, jika satu tetangga yang paling dekat memiliki kelas label yang berbeda dengan data tersebut maka kedua data akan dihapus karena dianggap sebagai noise atau misclassify.





HANDS-ON: TOMEK LINKS

Import Library

```
✓ 0s ▶ from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import TomekLinks
```

Generate Data

```
✓ 0s ▶ X, y = make_classification(n_samples=10000, weights=[0.85], flip_y=0)

print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({0: 8500, 1: 1500})
```

Implementasi Tomek Links

```
✓ 0s ▶ # Distribusi kelas sebelum dilakukan undersampling
print("Sebelum undersampling:", Counter(y))

# Metode Random Undersampling
undersample = TomekLinks(sampling_strategy='majority')
X_tl, y_tl = undersample.fit_resample(X, y)

# Distribusi kelas setelah dilakukan undersampling
print("Sesudah undersampling Tomek Links:", Counter(y_tl))
```

Output

```
Sebelum undersampling: Counter({0: 8500, 1: 1500})
Sesudah undersampling Tomek Links: Counter({0: 8378, 1: 1500})
```

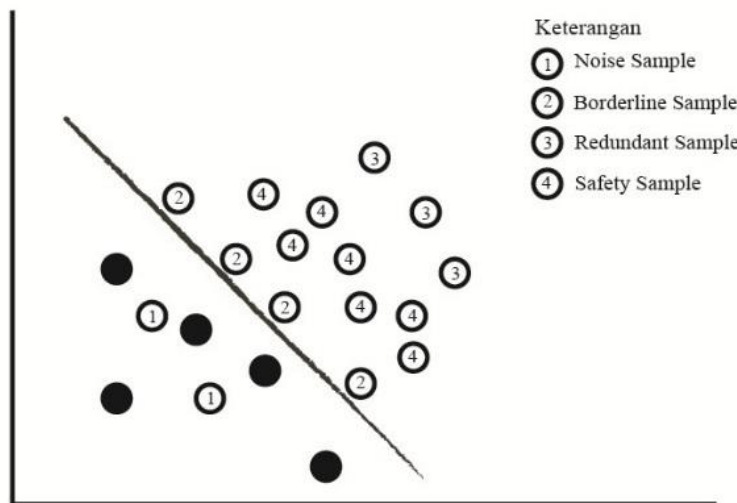


ONE SIDED-SELECTION (OSS)

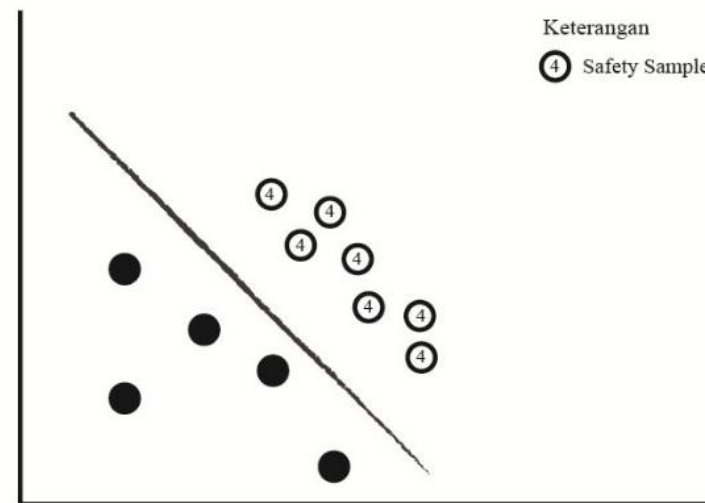
- One Sided Selection (OSS) diperkenalkan pertama kali oleh Kubat pada tahun 1997. Prinsip kerja algoritme OSS berbeda dengan Random Undersampling (RUS) dimana pengurangan kelas mayoritas memperhatikan letak dari sampel kelas tersebut. Hal ini dilakukan untuk menghindari penghapusan pada data kelas mayoritas yang memiliki informasi penting.
- Dalam prosesnya algoritme OSS membagi kelas mayoritas menjadi 4 bagian yaitu noise sample, borderline sample, redundant sample, dan safety sample.
 - 1) *Noise Sample* merupakan sampel kelas mayoritas yang letaknya dikelilingi oleh sampel kelas minoritas.
 - 2) *Borderline Sample* merupakan sampel kelas mayoritas yang terletak ditengah-tengah antara kedua kelas.
 - 3) *Redundant Sample* merupakan sampel kelas mayoritas yang terletak jauh dari batas antara kedua kelas.
 - 4) *Safety Sample* merupakan sampel kelas mayoritas yang dianggap memiliki informasi penting. Safety sample terletak tidak terlalu dekat dan tidak terlalu jauh dari garis batas antara kedua kelas.
- Selanjutnya algoritme OSS akan menghapus *noise sample*, *borderline sample*, serta mengurangi jumlah dari *redundant sample* dalam melakukan *undersampling*.



ONE SIDED-SELECTION (OSS)



(a)



(b)


Sumber:

- M. Kubat and S. Matwin, "Addressing the Curse of Imbalanced Training Sets: One Sided Selection," in International Conference on Machine Learning, 1997, vol. 97, pp. 179-186.
- Y. Pristyanto and A. Dahlan, "Hybrid Resampling for Imbalanced Class Handling on Web Phishing Classification Dataset," 2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, 2019, pp. 401-406, doi: 10.1109/ICITISEE48480.2019.9003803.
- Y. Pristyanto, I. Pratama and A. F. Nugraha, "Data level approach for imbalanced class handling on educational data mining multiclass classification," 2018 International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 2018, pp. 310-314, doi: 10.1109/ICOIACT.2018.8350792.




HANDS-ON: ONE SIDED-SELECTION (OSS)

Import Library

```
0s ✓  from collections import Counter  
from sklearn.datasets import make_classification  
from imblearn.under_sampling import OneSidedSelection
```


Generate Data

```
0s ✓  X, y = make_classification(n_samples=10000, weights=[0.65], flip_y=0)  
  
print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({0: 6500, 1: 3500})
```

Implementasi OSS

```
1s ✓  # Distribusi kelas sebelum dilakukan undersampling  
print("Sebelum undersampling:", Counter(y))  
  
# Metode Random Undersampling  
undersample = OneSidedSelection(n_neighbors=3, sampling_strategy='majority')  
X_oss, y_oss = undersample.fit_resample(X, y)  
  
# Distribusi kelas setelah dilakukan undersampling  
print("Sesudah undersampling OSS:", Counter(y_oss))
```

Output

```
Sebelum undersampling: Counter({0: 6500, 1: 3500})  
Sesudah undersampling OSS: Counter({0: 6291, 1: 3500})
```



HYBRID SAMPLING

Hybrid Sampling adalah pendekatan yang menggabungkan lebih dari satu teknik oversampling atau undersampling untuk mengatasi masalah ketidakseimbangan kelas dalam dataset. Pendekatan ini bertujuan untuk memaksimalkan keuntungan dari masing-masing teknik dan mencapai hasil yang lebih baik dalam mengatasi ketidakseimbangan kelas.

Ada dua metode hybrid sampling dapat digunakan, antara lain:

- SMOTE-TomekLinks
- SMOTE-ENN



SMOTE-TOMEK LINKS

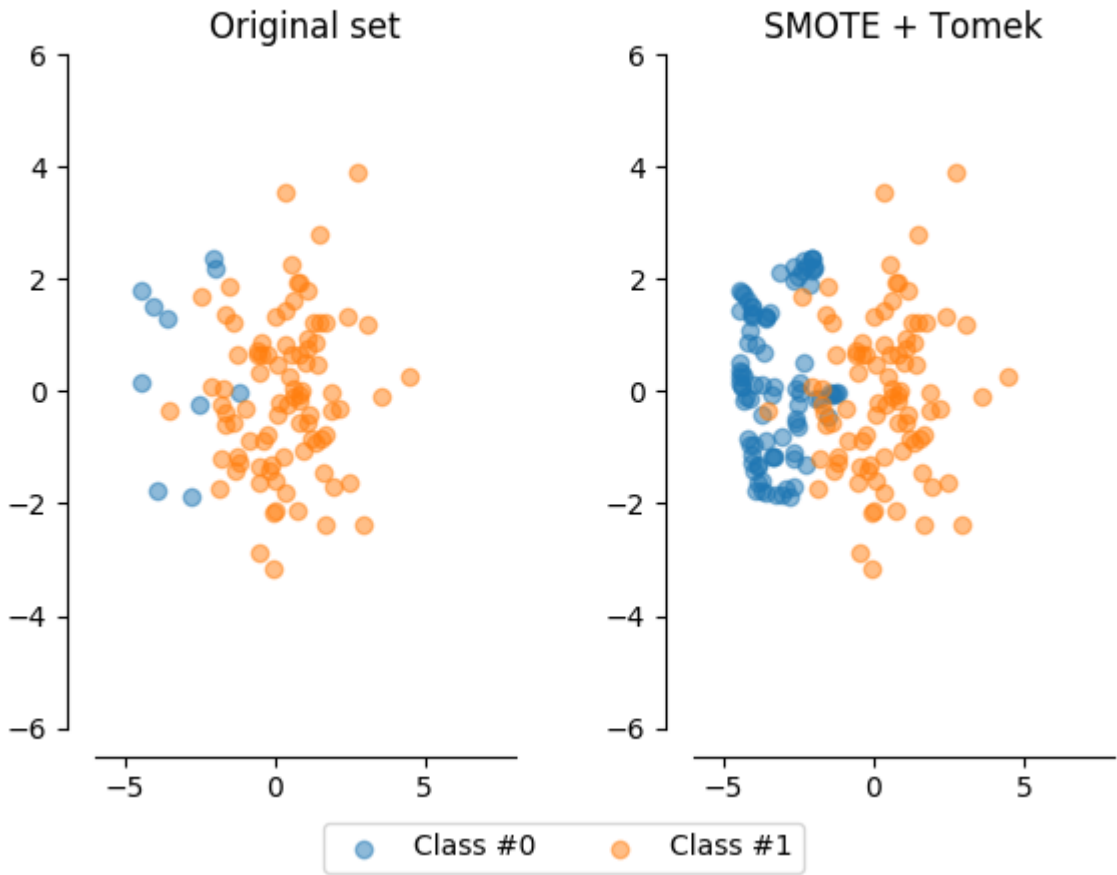
SMOTE-Tomek Links adalah pendekatan yang menggabungkan dua teknik untuk mengatasi masalah ketidakseimbangan kelas dalam dataset, yaitu SMOTE (Synthetic Minority Over-sampling Technique) dan Tomek Links Undersampling. Tujuan utama dari SMOTE-Tomek Links adalah untuk meningkatkan jumlah sampel kelas minoritas dengan menghasilkan sampel sintetis menggunakan SMOTE dan kemudian menghapus sampel yang tidak diperlukan dari kelas mayoritas menggunakan Tomek Links.

Tahapan SMOTE-Tomek Links:

- ***Penerapan SMOTE:*** Tahap pertama adalah mengimplementasikan SMOTE pada dataset Anda. Ini melibatkan pemilihan sampel-sampel dari kelas minoritas dan menghasilkan sampel-sampel sintetis yang mirip dengan sampel-sampel yang ada.
- ***Identifikasi Pasangan Tomek Links:*** Setelah penerapan SMOTE, langkah selanjutnya adalah mengidentifikasi pasangan-pasangan Tomek Links dalam dataset yang telah diubah. Ini melibatkan perhitungan jarak antara semua pasangan sampel dalam dataset.
- ***Pemilihan Sampel untuk Penghapusan:*** Sampel-sampel dari kelas mayoritas yang terlibat dalam pasangan Tomek Links dipilih untuk dihapus dari dataset. Ini adalah sampel-sampel yang mendekati kelas minoritas dan dapat memperkenalkan kebingungan dalam pemodelan.
- ***Penghapusan Sampel:*** Sampel-sampel yang telah dipilih pada langkah sebelumnya dihapus dari dataset. Ini mengurangi jumlah sampel kelas mayoritas.
- ***Hasil Akhir:*** Dataset yang telah diubah setelah menerapkan SMOTE-Tomek Links adalah hasil akhir dari proses ini. Ini adalah dataset yang lebih seimbang yang telah meningkatkan jumlah sampel kelas minoritas dan mengurangi sampel-sampel kelas mayoritas yang tidak perlu.



SMOTE-TOMEK LINKS





HANDS ON: SMOTE-TOMEK LINKS

Import Library

```
✓ 0s ▶ from collections import Counter
from sklearn.datasets import make_classification
from imblearn.combine import SMOTETomek
```

Generate Data

```
✓ 0s ▶ X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                             n_redundant=0, n_repeated=0, n_classes=3,
                             n_clusters_per_class=1,
                             weights=[0.01, 0.05, 0.94],
                             class_sep=0.8, random_state=0)

print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({2: 4674, 1: 262, 0: 64})
```

Implementasi SMOTE-Tomek Links

```
✓ 0s ▶ # Distribusi kelas sebelum dilakukan oversampling
print("Sebelum SMOTE-TomekLinks:", Counter(y))

# Metode Random Oversampling
smote_tomek = SMOTETomek(random_state=0)
X_st, y_st = smote_tomek.fit_resample(X, y)

# Distribusi kelas setelah dilakukan oversampling
print("Sesudah SMOTE-TomekLinks:", Counter(y_st))
```

Output

```
Sebelum SMOTE-TomekLinks: Counter({2: 4674, 1: 262, 0: 64})
Sesudah SMOTE-TomekLinks: Counter({1: 4566, 0: 4499, 2: 4413})
```



SMOTE - ENN

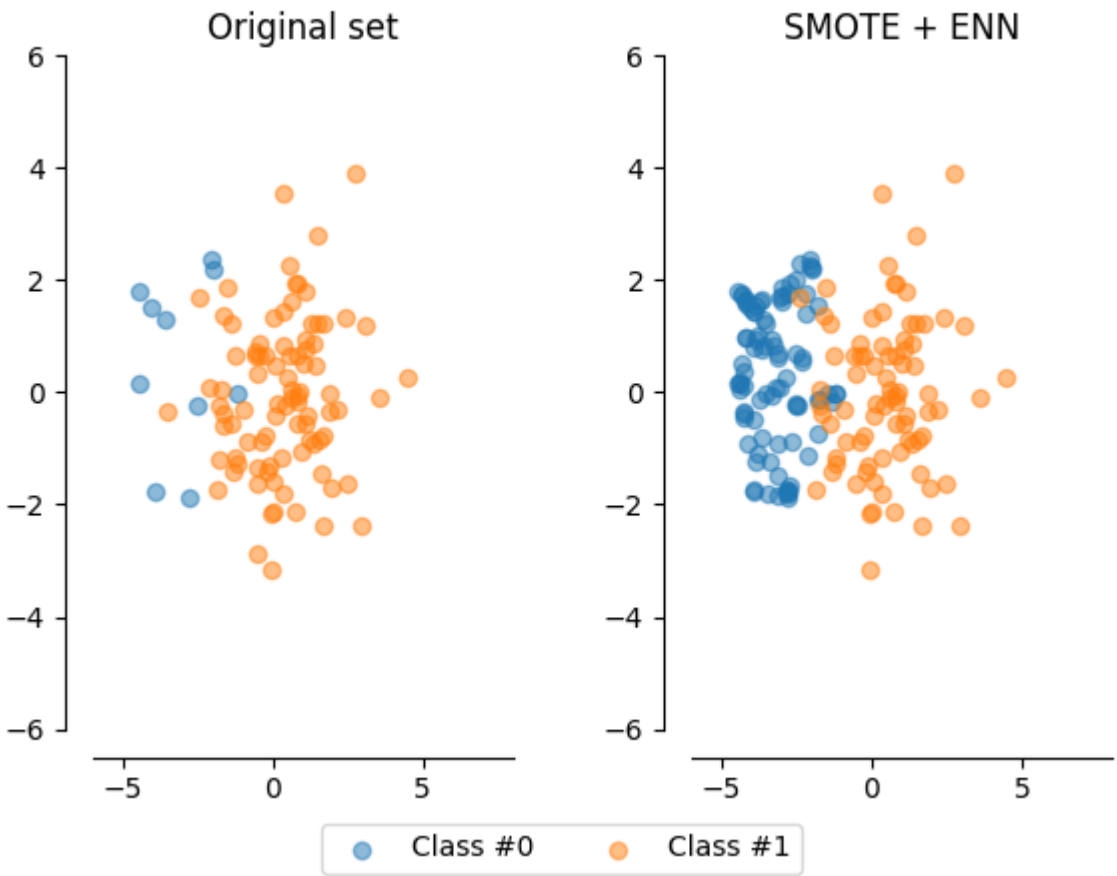
SMOTE-ENN adalah pendekatan yang menggabungkan dua teknik untuk mengatasi masalah ketidakseimbangan kelas dalam dataset, yaitu SMOTE (Synthetic Minority Over-sampling Technique) dan ENN (Edited Nearest Neighbors). Tujuan utama dari SMOTE-ENN adalah untuk meningkatkan jumlah sampel kelas minoritas dengan menghasilkan sampel sintetis menggunakan SMOTE dan kemudian menghapus sampel yang mungkin mengganggu dari kelas mayoritas menggunakan ENN.

Tahapan SMOTE-ENN:

- ***Penerapan SMOTE:*** Tahap pertama adalah menerapkan SMOTE pada dataset Anda. Ini melibatkan pemilihan sampel-sampel dari kelas minoritas dan menghasilkan sampel-sampel sintetis yang mirip dengan sampel-sampel yang sudah ada.
- ***Penerapan ENN:*** Setelah penerapan SMOTE, langkah selanjutnya adalah menerapkan ENN pada dataset yang telah diubah. ENN bekerja dengan mengidentifikasi sampel-sampel kelas mayoritas yang mungkin salah klasifikasi (mengganggu) dan menghapusnya.
- ***Hasil Akhir:*** Dataset yang telah diubah setelah menerapkan SMOTE-ENN adalah hasil akhir dari proses ini. Ini adalah dataset yang lebih seimbang yang telah meningkatkan jumlah sampel kelas minoritas dan mengurangi sampel-sampel kelas mayoritas yang mungkin mengganggu pemodelan.



SMOTE-ENN





HANDS ON: SMOTE-ENN

Import Library

```
0s ✓ from collections import Counter
from sklearn.datasets import make_classification
from imblearn.combine import SMOTEENN
```

Generate Data

```
1s ✓ X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_classes=3,
                           n_clusters_per_class=1,
                           weights=[0.01, 0.05, 0.94],
                           class_sep=0.8, random_state=0)

print("Distribusi kelas:", Counter(y))
```

Output

```
Distribusi kelas: Counter({2: 4674, 1: 262, 0: 64})
```

Implementasi SMOTE-ENN

```
0s ✓ # Distribusi kelas sebelum dilakukan oversampling
print("Sebelum SMOTE-ENN:", Counter(y))

# Metode Random Oversampling
smote_enn = SMOTEENN(random_state=0)
X_se, y_se = smote_enn.fit_resample(X, y)

# Distribusi kelas setelah dilakukan oversampling
print("Sesudah SMOTE-ENN:", Counter(y_se))
```

Output

```
Sebelum SMOTE-ENN: Counter({2: 4674, 1: 262, 0: 64})
Sesudah SMOTE-ENN: Counter({1: 4381, 0: 4060, 2: 3502})
```




RESOURCE IMBALANCED DATA

- Modul Imbalanced IPYNB

<https://drive.google.com/file/d/1LxgNEdCoSnbzVEEm69b9syHYE1lSewgo/view?usp=sharing>

- Link Dataset CSV

<https://drive.google.com/file/d/1Ltny2P35xRksP7nNeJYr3TkU0AlJLDzw/view?usp=sharing>

- More Resource About Imbalanced Data

<https://machinelearningmastery.com/resources-for-imbalanced-classification/>

<http://glemaitre.github.io/imbalanced-learn/index.html>

- More Dataset Resource Can Used For Imbalanced Data Experiment:

<https://machinelearningmastery.com/standard-machine-learning-datasets-for-imbalanced-classification/>



UNIVERSITAS AMIKOM
YOGYAKARTA

FEATURE SELECTION



APA ITU FEATURE SELECTION ?

- Seleksi fitur merupakan konsep inti dalam Machine Learning yang berdampak besar bagi kinerja model prediksi.
- Fitur data yang tidak/sebagian saja relevan dapat berdampak negative terhadap kinerja model.
- Definisi Seleksi Fitur itu sendiri adalah proses otomatis atau manual memilih fitur data yang paling berkontribusi terhadap variabel prediksi atau output yang diinginkan.

	K1	K2	K3	K4	K5	K6
R1						
R2						
R3						
R4						

Fitur:

Kolom yang dipilih Untuk sesuai tujuan



TUJUAN FEATURE SELECTION

→ *Cara lain untuk mengurangi dimensi data*

→ *Menghilangkan Atribut yang Redundant:*

- Menduplikasi sebagian besar atau seluruh informasi yang terkandung dalam satu atau lebih atribut lainnya.
- Misalnya, harga pembelian suatu produk dan jumlah pajak penjualan yang dibayarkan

→ *Mengurangi Atribut yang tidak relevan:*

- Tidak berisi informasi yang berguna untuk tugas penambangan data yang ada
- Misalnya, ID siswa seringkali tidak relevan dengan tugas memprediksi IPK siswa



MANFAAT FEATURE SELECTION

Reduksi Overfitting:

Semakin kecil data redundant maka keputusan berdasarkan noise semakin berkurang

Meningkatkan Akurasi:

Semakin kecil data misleading maka akurasi model lebih baik

Reduksi Waktu Training:

Semakin kecil titik data (data point) maka kompleksitas algoritma berkurang dan latih algoritma lebih cepat



METODE FEATURE SELECTION

Sejumlah pendekatan yang diusulkan untuk pemilihan fitur secara umum dapat dikategorikan ke dalam dua klasifikasi berikut: *wrapper* dan *filter*.

Filtering

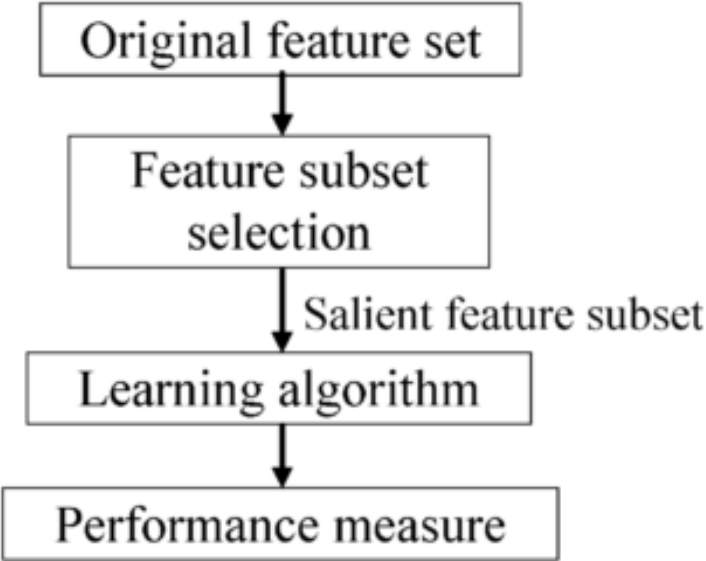
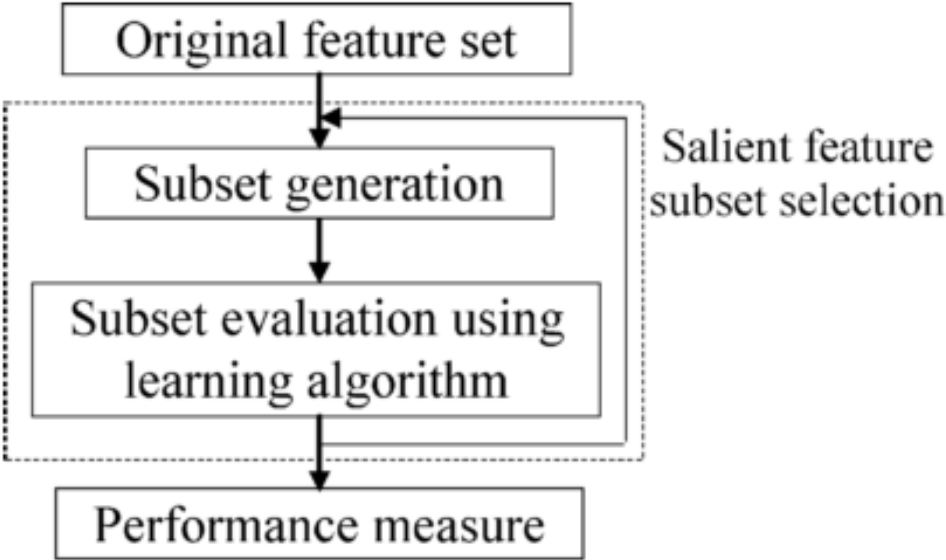
Merupakan salah satu metode seleksi fitur yang menggunakan statistik atau metrik yang tidak melibatkan model pembelajaran mesin untuk menilai kualitas fitur-fitur dalam dataset.

Wrapper

Merupakan salah satu metode seleksi fitur yang menggunakan algoritma pembelajaran mesin untuk menilai kualitas fitur-fitur dalam dataset. Ini adalah salah satu metode yang lebih kuat tetapi juga lebih komputasi-intensif dibandingkan dengan metode seleksi fitur lainnya.



WRAPPER DAN FILTER





WRAPPER DAN FILTER

Filter Approach:

- Information Gain
- Correlation-based Feature Selection



Pada tahap preprocessing proses fitur seleksi difokuskan pada pendekatan filtering.

Wrapper Approach:

- Forward Selection
- Recursive Feature Elimination



CORRELATION-BASED FEATURE SELECTION FEATURE FILTERING

- Correlation-based feature selection yang selanjutnya disebut seleksi atribut berbasis korelasi atau CBFS adalah sebuah algoritma filter sederhana yang meranking subset berdasarkan fungsi evaluasi heuristik berbasis korelasi.
- Berdasarkan hipotesis bahwa subset atribut yang bagus berisi atribut yang mempunyai korelasi tinggi terhadap kelas dan tidak saling berkorelasi satu sama lain.
- Korelasi yang tinggi satu sama lain atribut menandakan atribut tersebut redundan. Atribut yang berkorelasi rendah terhadap kelas adalah atribut yang tidak relevan.
- Atribut yang tidak relevan dan redundan harus dihapus.



CORRELATION-BASED FEATURE SELECTION FEATURE FILTERING

Berikut langkah untuk menghitung nilai correlation pada CBFS :

- Hitung nilai coefficient correlation dengan Symmetrical Uncertainty menggunakan persamaan berikut :

$$SU = 2.0 * \left(\frac{Gain}{H(X)+H(Y)} \right)$$

dimana :

H : Entropy Attribute

X : Attribute 1

Y : Attribute 2

- Hitung nilai Merits menggunakan persamaan berikut :

$$Merits = \frac{k \overline{r_{cf}}}{\sqrt{(k+k-1) \overline{r_{ff}}}}$$

dimana :

K : Jumlah Attribute

$\overline{r_{cf}}$: correlation class-attribute

$\overline{r_{ff}}$: intercorrelation attribute-attribute

- Lakukan perhitungan Merits untuk semua kemungkinan kombinasi pemilihan atribut untuk menemukan nilai Merits tertinggi.



INFORMATION GAIN FEATURE FILTERING

- Information gain merupakan salah satu metode seleksi fitur sederhana dan banyak dipakai oleh peneliti melakukan reduksi dimensi pada dataset.
- Nilai information gain diperoleh dari nilai entropy sebelum pemisahan dikurangi dengan nilai entropy setelah pemisahan.
- Pengukuran nilai ini hanya digunakan sebagai tahap awal untuk penentuan atribut yang nantinya akan digunakan atau dibuang.
- Atribut yang memenuhi kriteria pembobotan yang nantinya akan digunakan dalam proses klasifikasi sebuah algoritma.



INFORMATION GAIN FEATURE FILTERING

→ Pemilihan fitur dengan information gain dilakukan dalam 3 tahapan yaitu:

- Menghitung nilai information gain untuk setiap atribut dalam dataset original.
- Tentukan batas (treshold) yang diinginkan. Hal ini akan memungkinkan atribut yang berbobot sama dengan batas atau lebih besar akan dipertahankan serta membuang atribut yang berada dibawah batas.
- Dataset diperbaiki dengan pengurangan atribut berdasarkan ranking nilai information gain.

→ Berikut ini persamaan untuk menghitung nilai Entropy dan Information Gain:

$$\text{Entropy}(S) = \sum_{i=1}^n -p_i * \log_2 p_i$$

dimana,

S : Himpunan kasus

n : Jumlah partisi S

p_i : Proporsi dari S_i terhadap S

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{i=1}^n \frac{|S_i|}{|S|} * \text{Entropy}(S_i)$$

dimana,

S : Himpunan kasus

A : Atribut

n : Jumlah partisi atribut A

$|S_i|$: Jumlah kasus pada partisi ke- i

$|S|$: Jumlah kasus dalam S



HANDS-ON: FEATURE SELECTION

- Modul Imbalanced IPYNB

<https://drive.google.com/file/d/1MNkKj2QZly-pbjZSgmm6MRu8By8at3AI/view?usp=sharing>

APAKAH ADA PERTANYAAN ???



TUGAS

- Download dataset pada link ini:
<https://drive.google.com/file/d/1Ur20rDb6P4T0cxQHfV5BDvY0e02D2g8y/view?usp=sharing>
- Lakukanlah resampling menggunakan : One Sided-Selection (OSS), SMOTE, dan SMOTE-TomekLinks pada dataset tersebut.
- Lakukanlah fitur seleksi dan pilih 5 fitur terbaik menggunakan : CBFS dan Information Gain pada dataset tersebut.
- Kumpulkan file **.ipynb** dari resampling dan fitur seleksi.
- Penamaan file yang harus dikumpulkan : **NIM_Nama.ipynb**
- Kumpulkan file **.ipynb** anda pada link berikut : <https://ungu.in/SubmitDatViz2023>
- Batas pengumpulan : **Selasa 10 Oktober 2023 Pukul 23.59 WIB.**

TERIMA KASIH