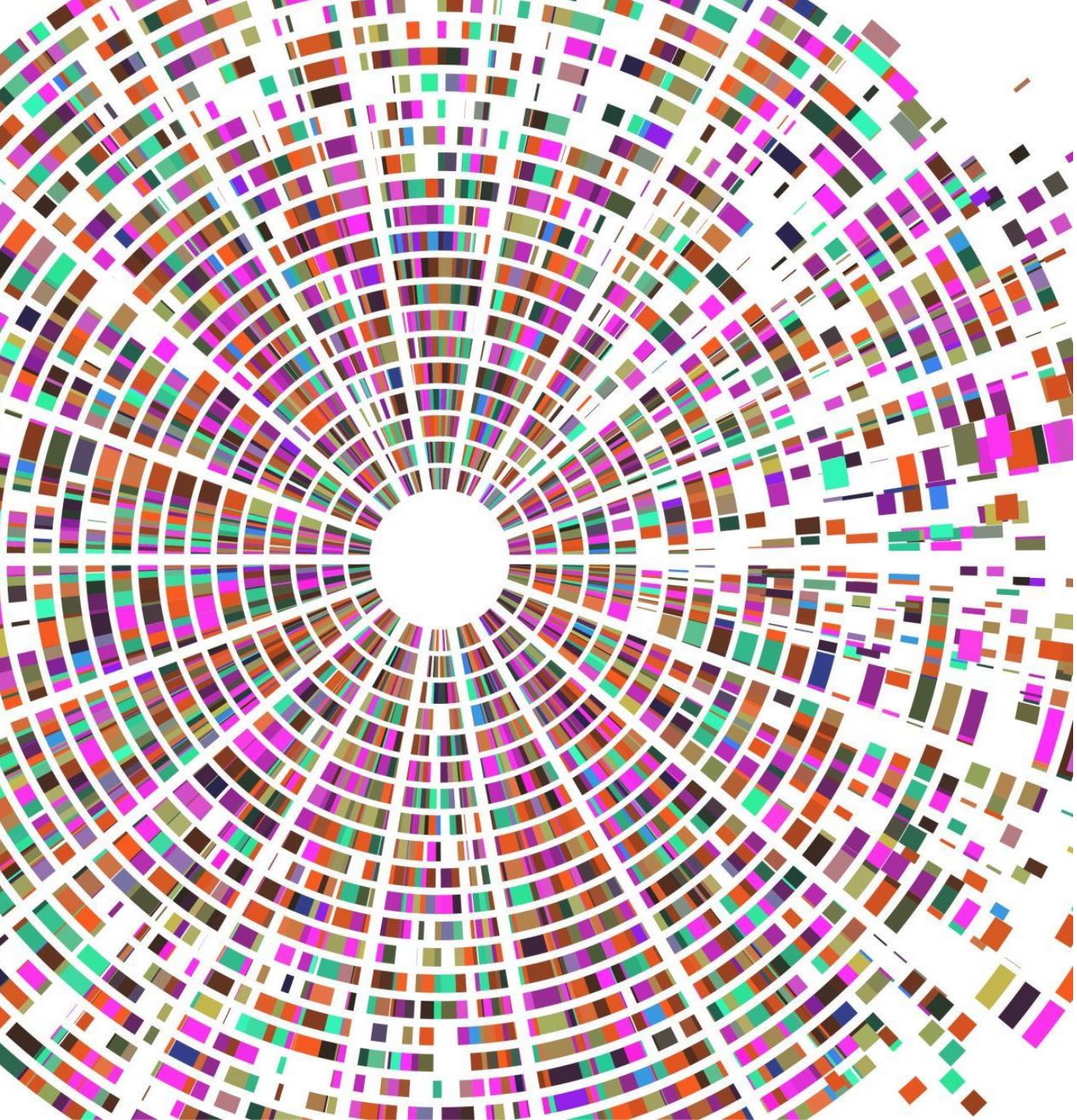
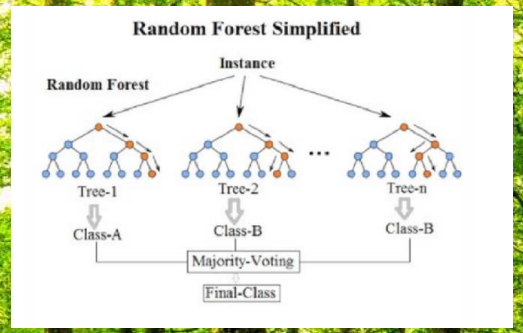


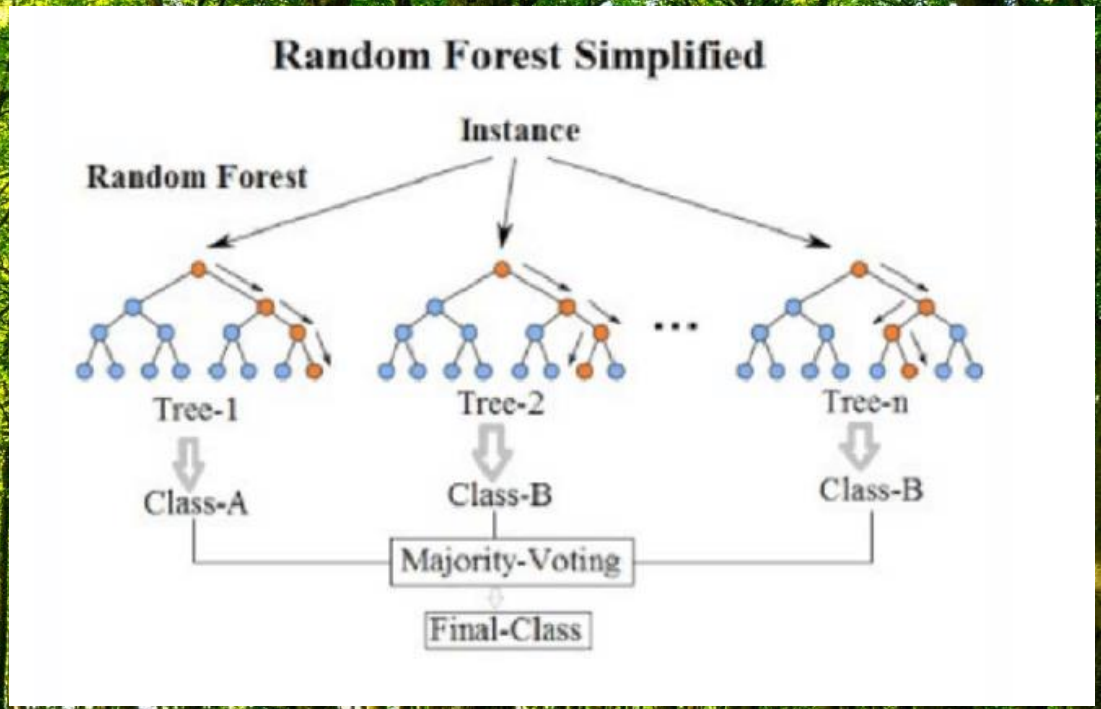
Klasyfikacja Metagenomiczna: MinHash

Piotr Kupidura

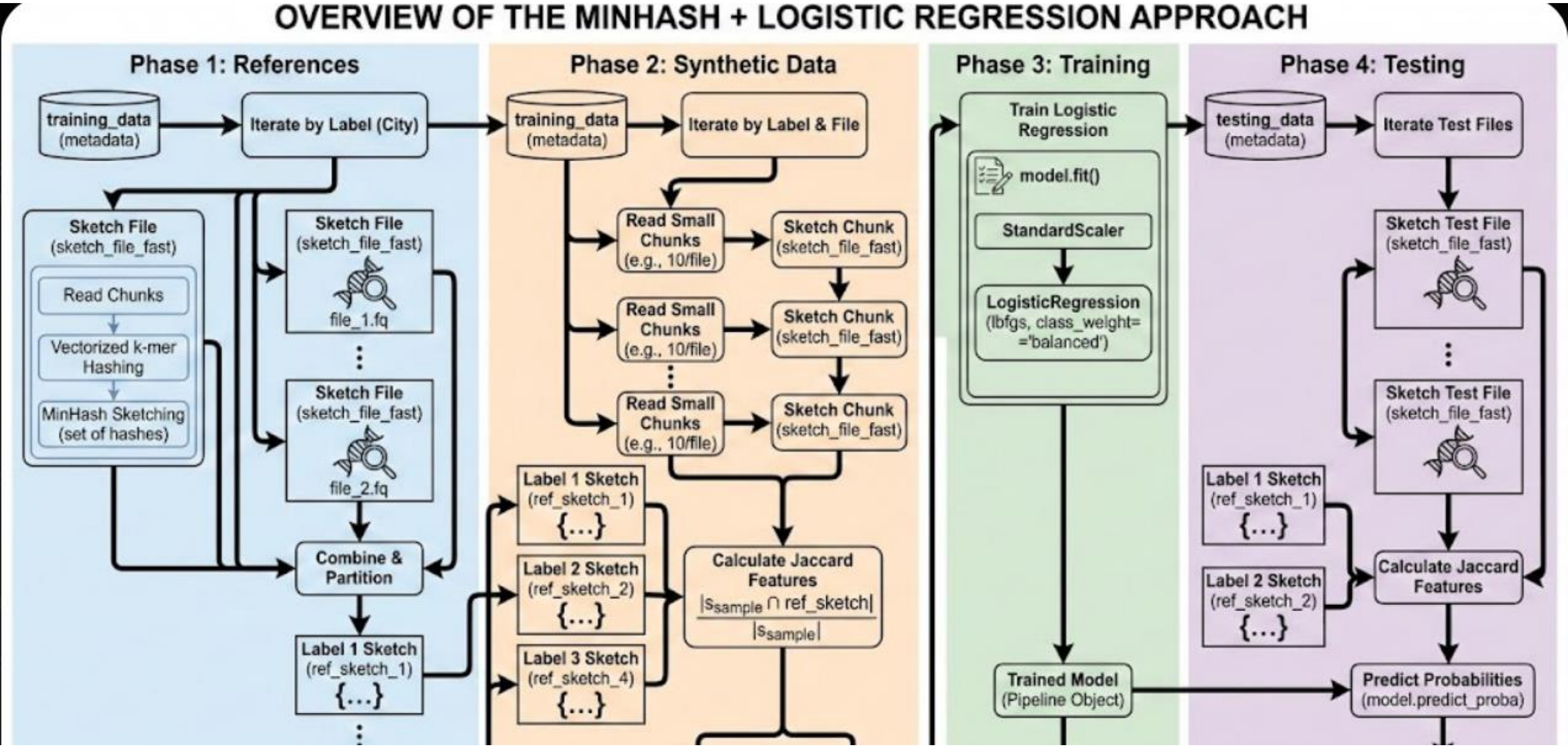
Michał Stanowski

Jakub Giezgała





RANDOM FOREST (AUC: 0.78)


OVERVIEW OF THE MINHASH + LOGISTIC REGRESSION APPROACH
Phase 1: References
training_data (metadata)
Iterate by Label (City)
Sketch File (sketch_file_fast)
Read Chunks
Vectorized k-mer Hashing
MinHash Sketching (set of hashes)
Sketch File (sketch_file_fast)
file_1.fq
Sketch File (sketch_file_fast)
file_2.fq
Combine & Partition
Label 1 Sketch (ref_sketch_1)
{...}
Phase 2: Synthetic Data
training_data (metadata)
Iterate by Label & File
Read Small Chunks (e.g., 10/file)
Sketch Chunk (sketch_file_fast)
Read Small Chunks (e.g., 10/file)
Sketch Chunk (sketch_file_fast)
Read Small Chunks (e.g., 10/file)
Sketch Chunk (sketch_file_fast)
Label 1 Sketch (ref_sketch_1)
{...}
Label 2 Sketch (ref_sketch_2)
{...}
Label 3 Sketch (ref_sketch_4)
{...}
Calculate Jaccard Features
|s_sample ∩ ref_sketch| / |s_sample|
Phase 3: Training
Train Logistic Regression
model.fit()
StandardScaler
LogisticRegression (lbfgs, class_weight='balanced')
Trained Model (Pipeline Object)
Phase 4: Testing
testing_data (metadata)
Iterate Test Files
Sketch Test File (sketch_file_fast)
Sketch Test File (sketch_file_fast)
Label 1 Sketch (ref_sketch_1)
{...}
Label 2 Sketch (ref_sketch_2)
{...}
Calculate Jaccard Features
Predict Probabilities (model.predict_proba)

FASTER & FASTER (Wang & NumPy)

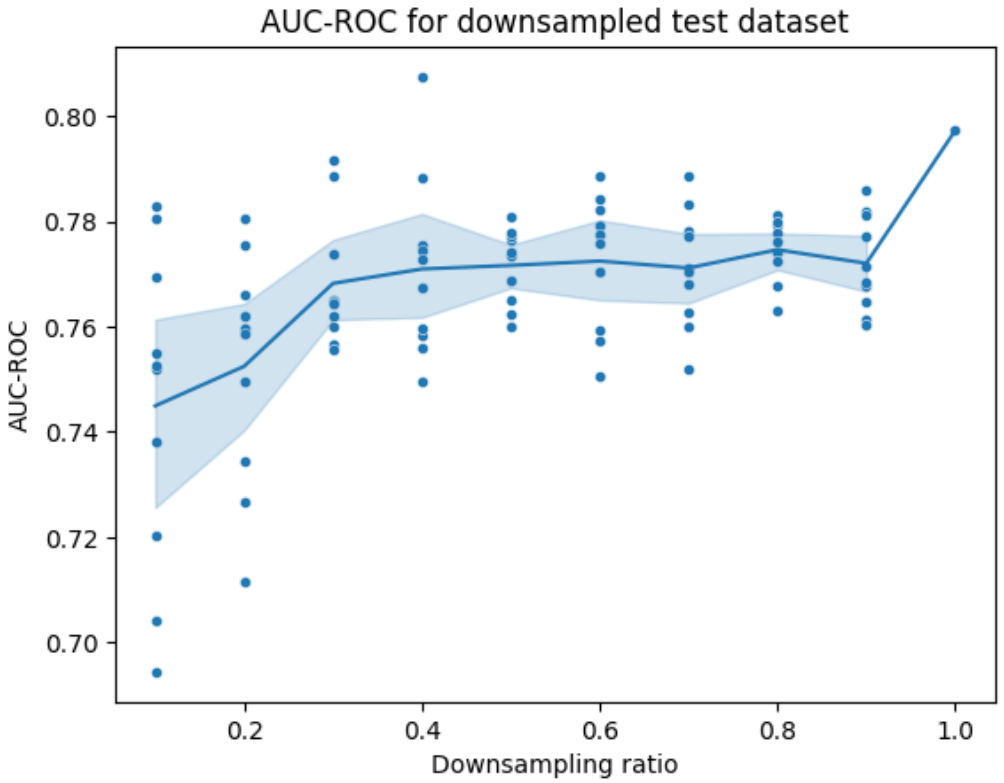
```python
def vectorized_hash(arr: np.ndarray) -> np.ndarray:
    """ Invertible integer mixing function (Wang Hash) for randomization.
    # MurmurHash3 64-bit a bit faster but get worse AUC
    key = arr.astype(np.uint64)
    key ^= key >> np.uint64(33)
    key *= np.uint64(0xff51afd7ed558ccd)
    key ^= key >> np.uint64(33)
    key *= np.uint64(0xc4ceb9fe1a85ec53)
    key ^= key >> np.uint64(33)
    """
    key = arr.copy()
    key = (~key) + (key << 21)
    key = key ^ (key >> 24)
    key = (key + (key << 3)) + (key << 8)
    key = key ^ (key >> 14)
    key = (key + (key << 2)) + (key << 4)
    key = key ^ (key >> 28)
    key = key + (key << 31)
    return key
```

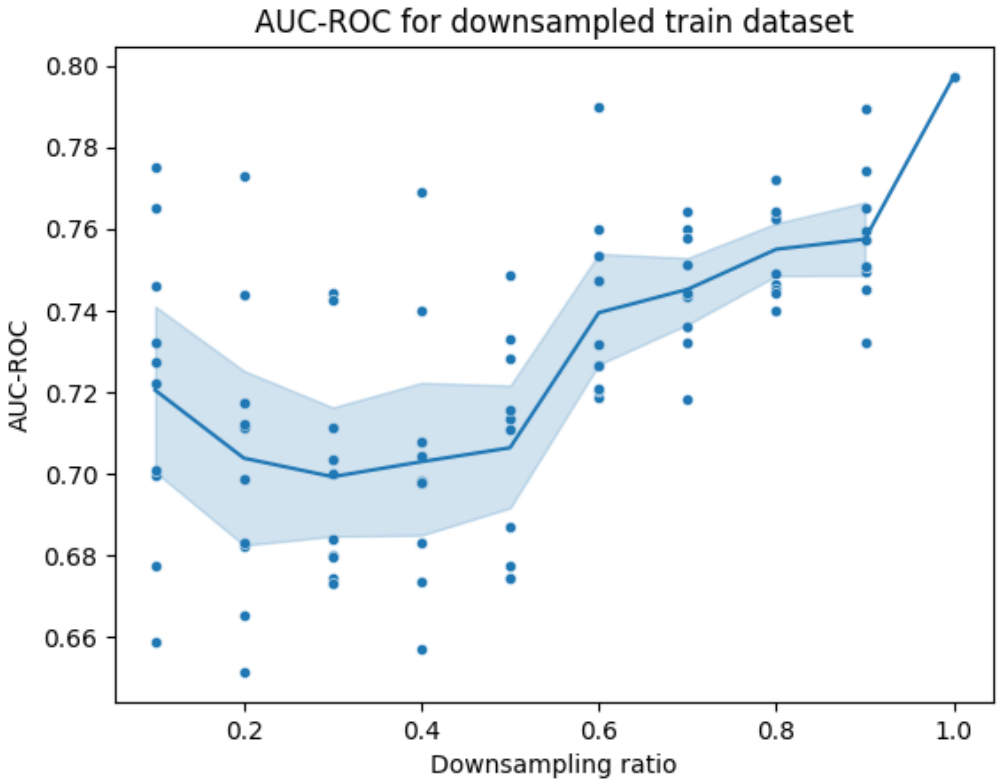
My friend showing me how his 1000 lines of C++ code is 100x faster than my 10 line Python code.

GRID SEARCH

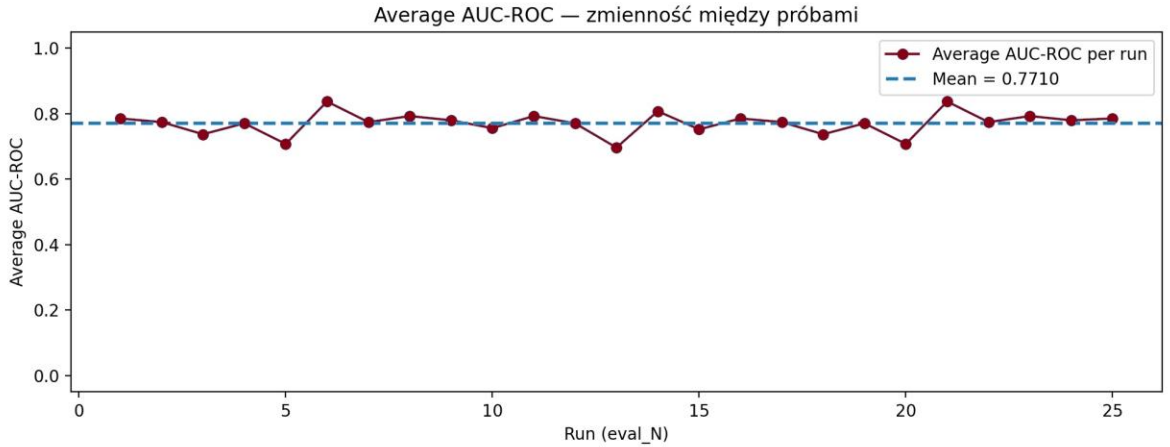
- **`DEFAULT_K = 25 ->`** Długość k-meru, na którym operuje algorytm. Wysoka wartość pozwala na dużą specyficzność biologiczną. Pozwala na precyzyjne odróżnianie sygnatur, minimalizując ryzyko przypadkowych dopasowań.
- **`DEFAULT_SEED = 424242 ->`** `Zapewnia determinizm i powtarzalność przy inicjacji` modelu klasyfikatora (Regresji Logistycznej).
- **`CHUNK_SIZE_KB = 75 ->`** Rozmiar fragmentu (w kilobajtach), na jakie dzielone są pliki treningowe w fazie generowania syntetycznego podziału, w celu uczenia na mniejszym podzbiorze.
- **`DEFAULT_SKETCH_SIZE = 7500 ->`** Liczba najmniejszych wartości funkcji skrótu (hashy) przechowywanych w "szkicu" dla każdej próbki. Szkic zajmuje w pamięci dokładnie 7500×8 bajtów (dla 64-bitowych liczb całkowitych) ≈60 KB na próbkę. Im większy rozmiar, tym mniejszy błąd statystyczny przy porównywaniu próbek (Error≈1/SketchSize)

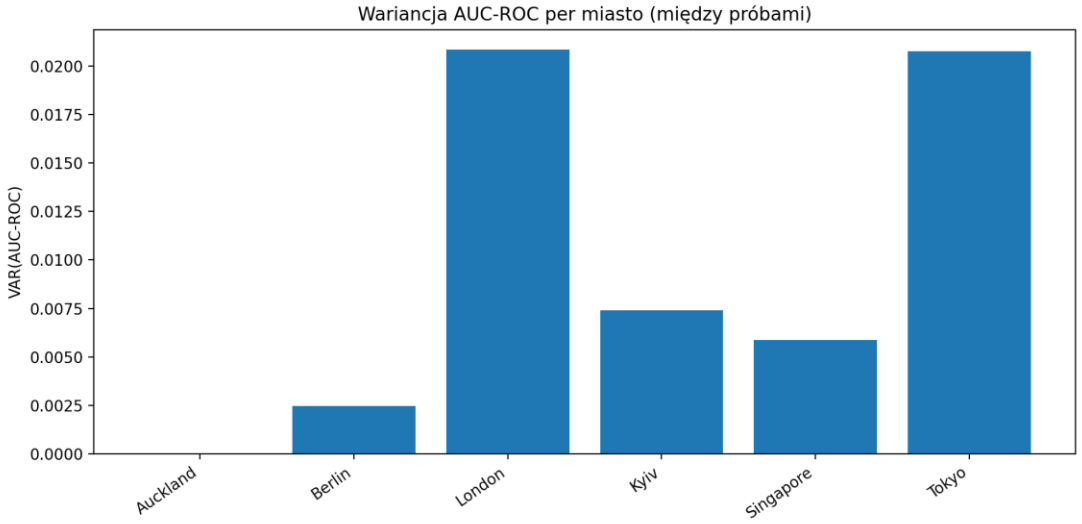
DOWNSAMPLING

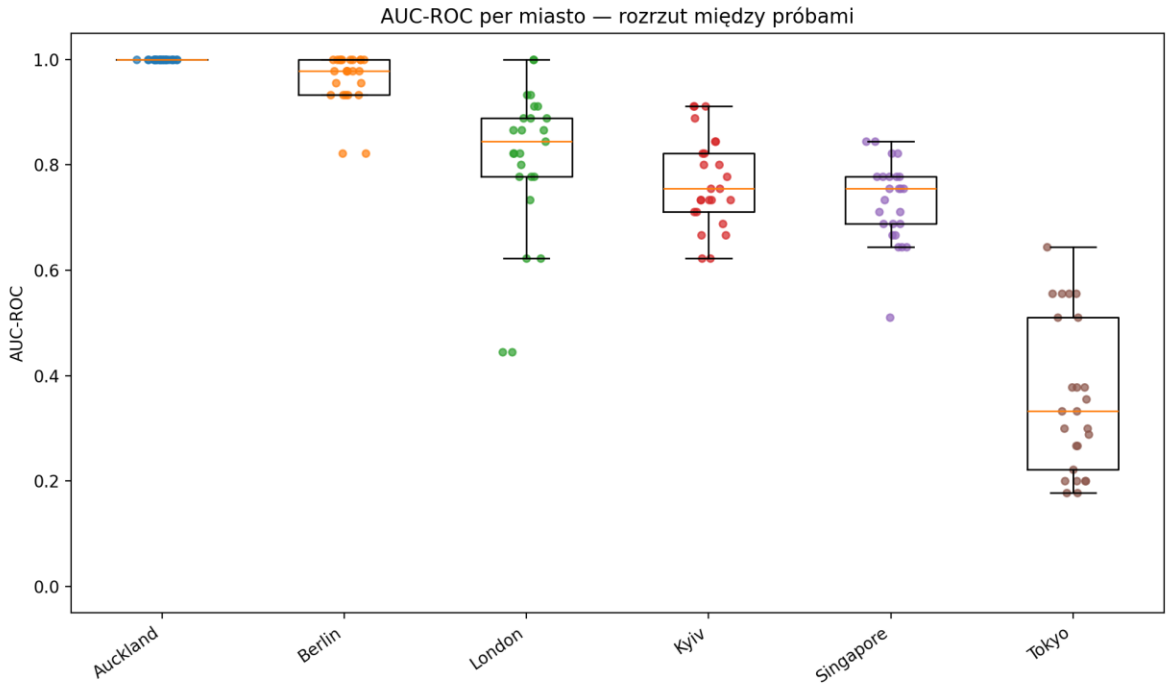




EVALUATION ON DIFFERENT SUBSETS







REFERENCES

Mourad Azhari, Altaf Alooui, Zakia Acharoui, Badia Ettaki - "*Adaptation of the random forest method: solving the problem of pulsar search*".

Nicholas A. Bokulich - "*Integrating sequence composition information into microbial diversity analyses with k-mer frequency counting*".

Debopriya Ghosh, Javier Cabrera - "*Enriched Random Forest for High Dimensional Genomic Data*".

Kenneth S Katz, Oleg Shutov, Richard Lapoint, Michael Kimelman, J Rodney Brister, Christopher O'Sullivan - "*STAT: a fast, scalable, MinHash-based k-mer tool to assess Sequence Read Archive next-generation sequence submissions".*