ITADATAhack 2025

Team Six

Team Members

Alexandre Cambier

Politecnico di Milano

Pavel Savinov

Università degli Studi
Di Padova

Omar Touil

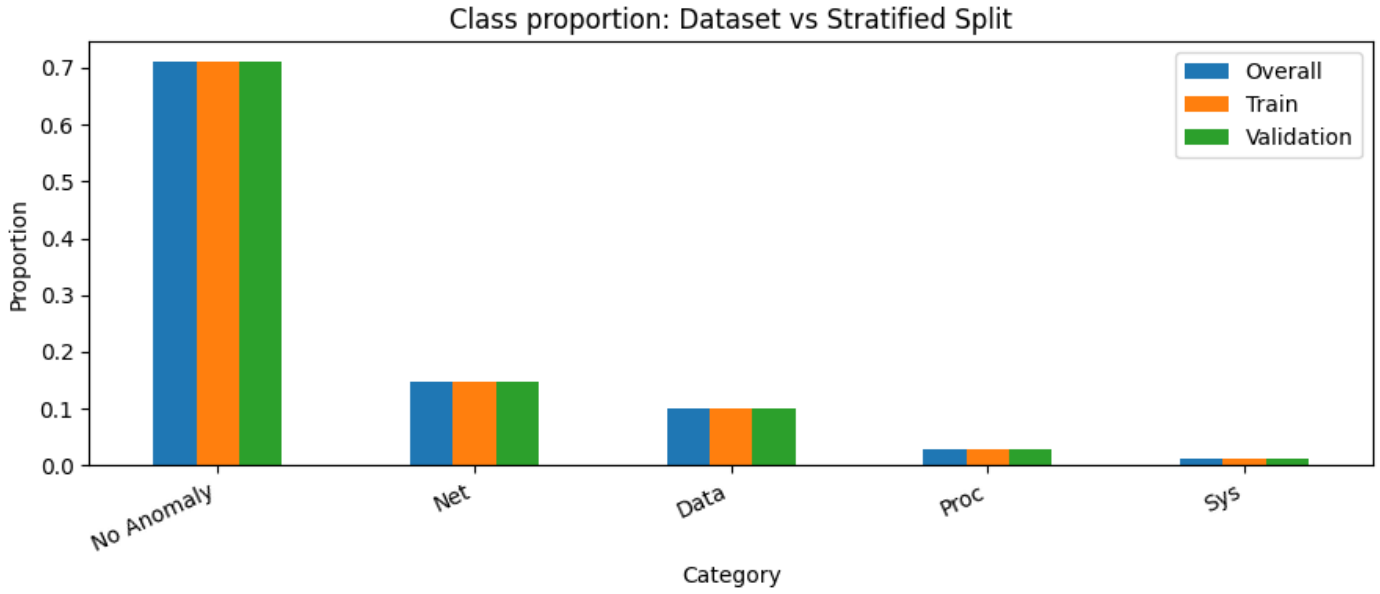
Università degli Studi
del Piemonte Orientale

Task 1 – The Problem

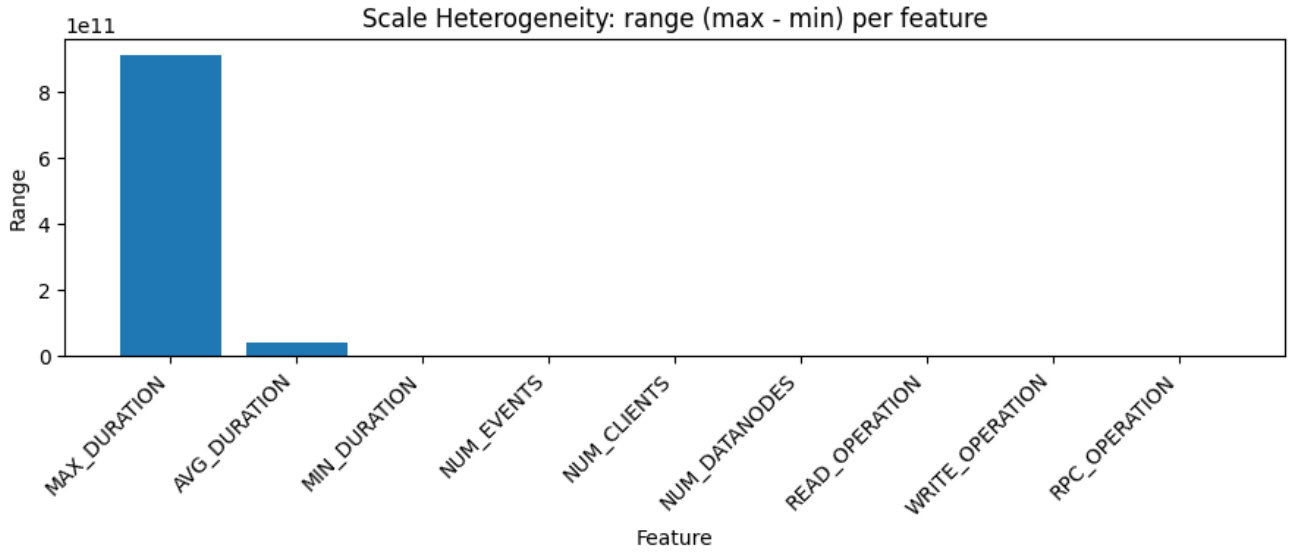
- Data: HDFS traces, 12 features, train 119,860, test 29,999.
- Task: Predict anomaly category per trace — sys, proc, net, data, no_anomaly.
- Metric: Weighted G-Mean from per-class precision and recall.

Task 1 – Same Proportion in Train and Validation

```
train = pd.read_csv("training_categoria.dsv", sep=";")
test = pd.read_csv("test_categoria.dsv", sep=";")

X = train.drop(["TRACE_ID", "ANOMALY", "ANOMALY_CATEGORY"], axis=1)
y = train["ANOMALY_CATEGORY"]

le = LabelEncoder()
y = le.fit_transform(y)

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

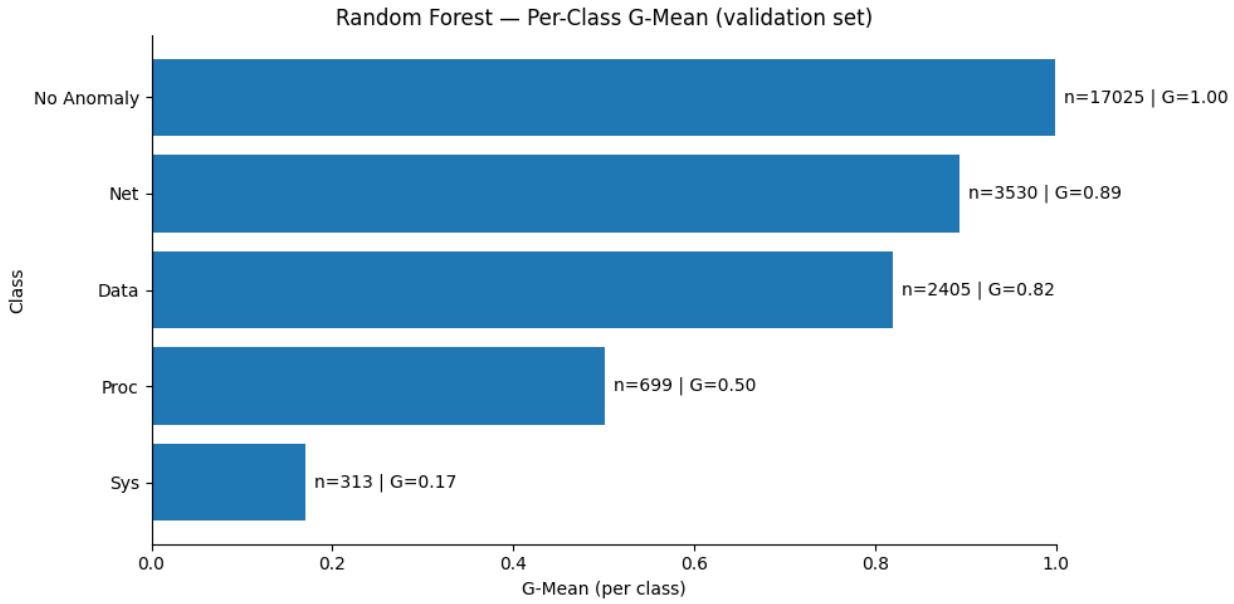


Task – Putting Variables on the Same Scale




```
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(test.drop("TRACE_ID", axis=1))
```

Task 1 – Random Forest and G Mean




```python
clf = RandomForestClassifier(
    n_estimators=300,
    class_weight="balanced",
    random_state=42
)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_val)

gmean = geometric_mean_score(y_val, y_pred,
                             average='weighted')
print("G-Mean Weighted:", gmean)

y_test_pred = clf.predict(X_test)
y_test_pred_labels = le.inverse_transform(y_test_pred)
```

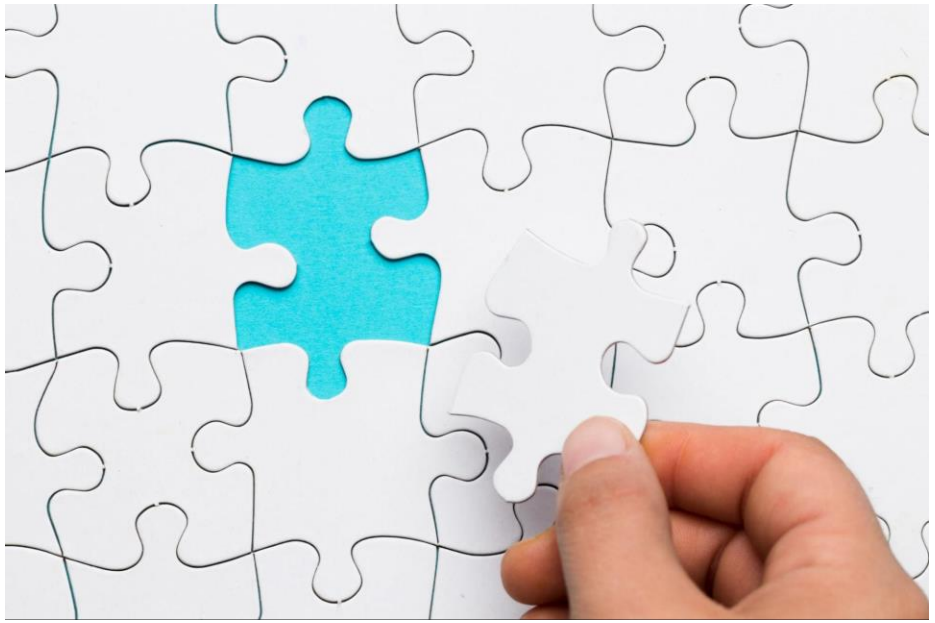
Task 2 – The Problem

- Data: HDFS traces, train 140,824, test 30,001.
- Task: Predict anomaly type per trace across Data, Proc, Net, Sys, plus no anomaly — 15 classes.
- Metric: Weighted G-Mean from per-class precision and recall.

Task 2 – Feature Engineering

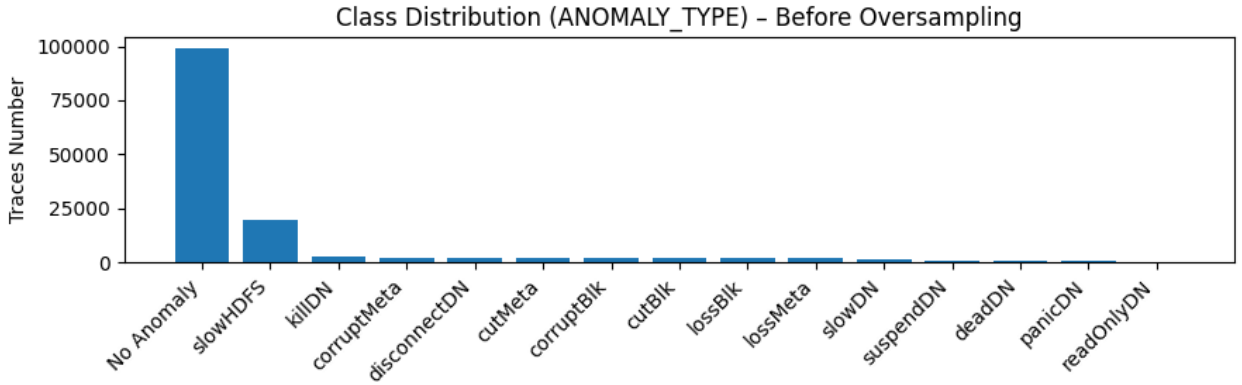
```
# --------------------------
# Load data
# --------------------------
train = pd.read_csv("training_tipologia.dsv", sep=";")
test = pd.read_csv("test_tipologia.dsv", sep=";")
test_ids = test["TRACE_ID"]

feature_cols = [
    'NUM_EVENTS', 'MAX_DURATION', 'MIN_DURATION', 'AVG_DURATION',
    'NUM_CLIENTS', 'NUM_DATANODES', 'READ_OPERATION', 'WRITE_OPERATION', 'RPC_OPERATION',
    'MAX_NODES_PER_LEVEL', 'MIN_NODES_PER_LEVEL', 'AVG_NODES_PER_LEVEL',
    'MAX_CHILDREN_PER_NODE', 'MIN_CHILDREN_PER_NODE', 'AVG_CHILDREN_PER_NODE'
]

X = train[feature_cols].copy()
y = train["ANOMALY_TYPE"].copy()
X_test = test[feature_cols].copy()

# --------------------------
# Feature Engineering
# --------------------------
for df in [X, X_test]:
    df['AVG_DURATION_PER_EVENT'] = df['AVG_DURATION'] / (df['NUM_EVENTS'] + 1e-6)
    df['READ_PER_DN'] = df['READ_OPERATION'] / (df['NUM_DATANODES'] + 1e-6)
    df['WRITE_PER_DN'] = df['WRITE_OPERATION'] / (df['NUM_DATANODES'] + 1e-6)
    df['RPC_PER_EVENT'] = df['RPC_OPERATION'] / (df['NUM_EVENTS'] + 1e-6)
    df['EVENTS_CLIENTS'] = df['NUM_EVENTS'] * df['NUM_CLIENTS']
```

Task 2 – Filling the Missing Values

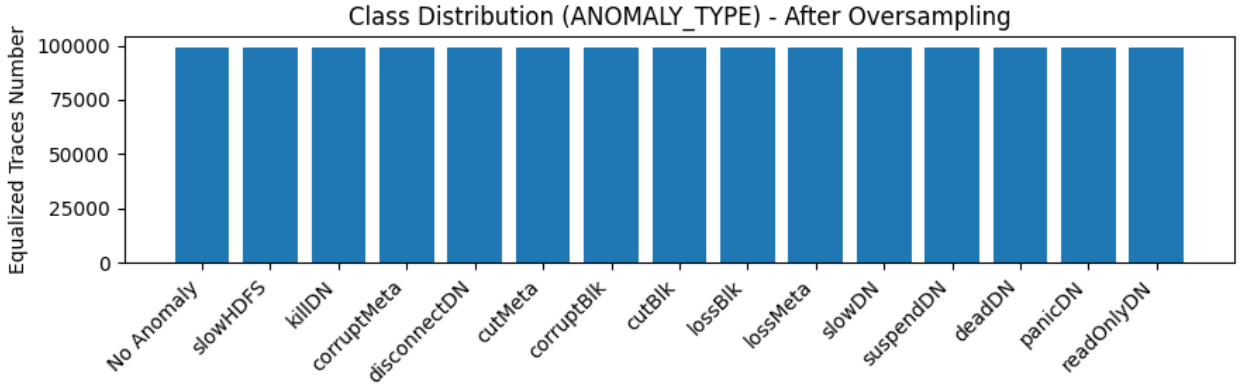


```
# ---------------------------
# Preprocessing
# ---------------------------
imputer = SimpleImputer(strategy='median')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
X_test = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)

# Encode labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)
num_classes = len(le.classes_)
```

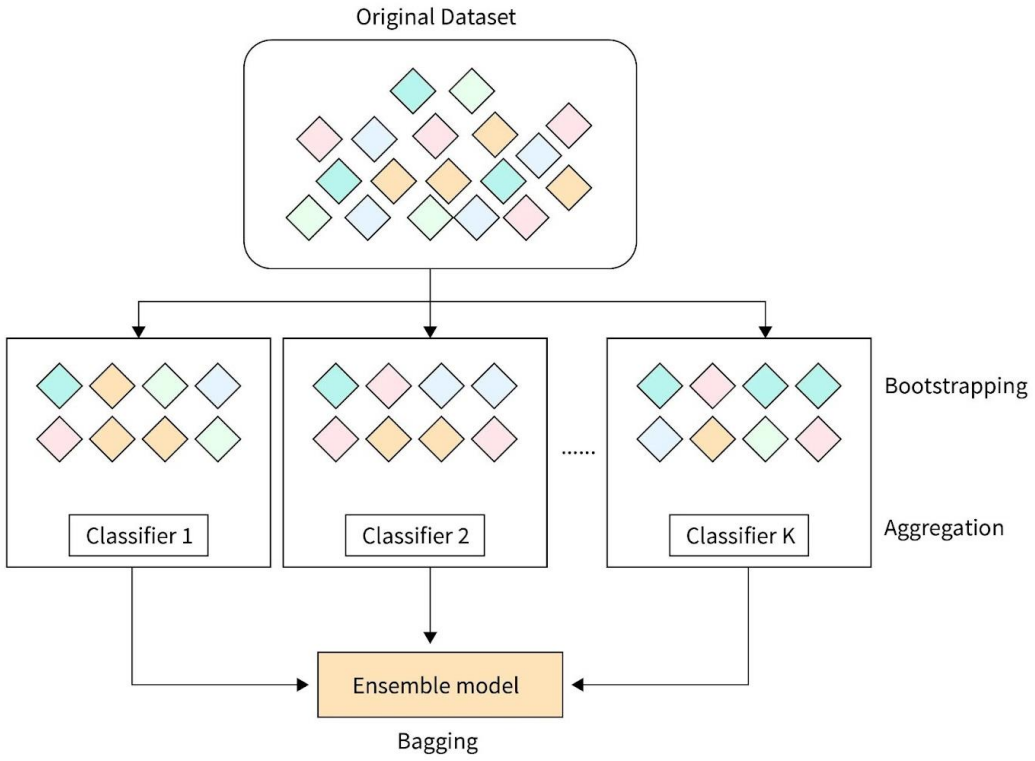
Task 2 – Oversampling




```python
# ---------------------------
# Oversample rare classes in subset
# ---------------------------
df_subset = X_subset.copy()
df_subset['target'] = y_subset
max_count = df_subset['target'].value_counts().max()
df_list = []

for cls in df_subset['target'].unique():
    df_cls = df_subset[df_subset['target'] == cls]
    if len(df_cls) < max_count:
        df_cls_resampled = resample(df_cls, replace=True,
                                    n_samples=max_count, random_state=42+i)
        df_list.append(df_cls_resampled)
    else:
        df_list.append(df_cls)

df_balanced = pd.concat(df_list)
X_balanced = df_balanced.drop('target', axis=1)
y_balanced = df_balanced['target']
```

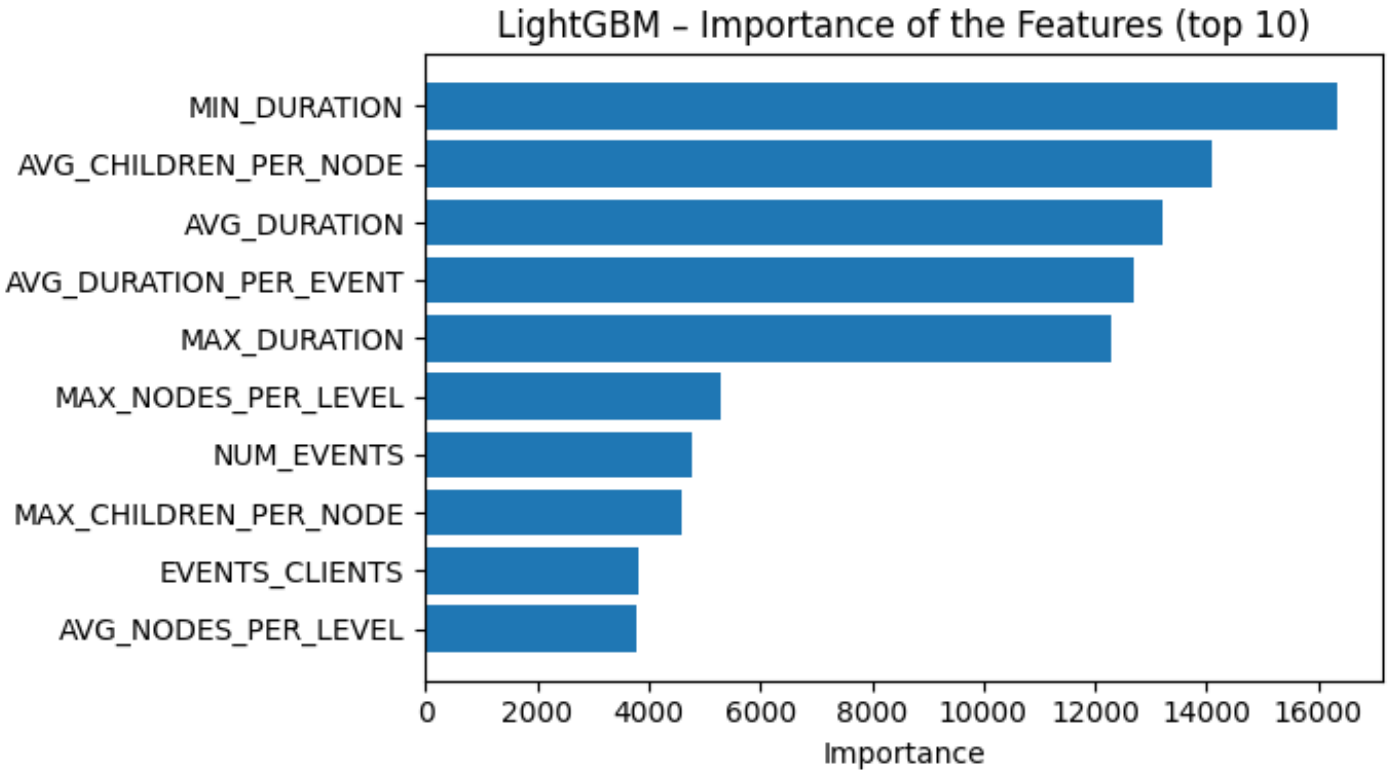


Task 2 - Bagging




```
# ---------------------------
# Bagging with Oversampling
# ---------------------------
n_models = 5
subset_size = 45000  # ~1/3 of training data per model
sss = StratifiedShuffleSplit(n_splits=n_models,
                             train_size=subset_size, random_state=42)

pred_probs = np.zeros((X_test.shape[0], num_classes))

for i, (train_idx, _) in enumerate(sss.split(X, y_encoded)):
    print(f"Training model {i+1}/{n_models}")

    X_subset = X.iloc[train_idx]
    y_subset = y_encoded[train_idx]
```

Task 2 – LightGBM Model




```
# --------------------------
# Train LightGBM
# --------------------------
model = LGBMClassifier(
    n_estimators=300,
    learning_rate=0.05,
    num_leaves=32,
    class_weight='balanced',
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42+i,
    n_jobs=-1,
    verbose=-1
)
model.fit(X_balanced, y_balanced)
```

Thank you for your attention!