Risultati

Francesco Fiaschi

November 2024

Contents

1 Risultato Paper OPS-SAT

1.1 Supervised

Modello	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Score
LinearSVC	0.926	0.911	0.726	0.808	0.771	0.949	0.976	0.867
LogisticRegression	0.924	0.92	0.708	0.8	0.764	0.949	0.976	0.867
FCNN	0.96	0.926	0.885	0.905	0.88	0.963	0.982	0.903
AdaBoost	0.934	0.89	0.788	0.836	0.797	0.923	0.962	0.841
RF+ICCS	0.964	0.98	0.85	0.91	0.891	0.949	0.976	0.867
Linear+L2	0.902	0.969	0.558	0.708	0.69	0.889	0.95	0.814
XGBOD	0.968	0.953	0.894	0.922	0.903	0.969	0.99	0.912

1.2 Unsupervised

Modello	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Score
MO-GAAL	0.896	0.939	0.549	0.693	0.669	0.771	0.849	0.699
AnoGAN	0.594	0.296	0.655	0.408	0.19	0.403	0.651	0.239
SO-GAAL	0.89	0.937	0.522	0.67	0.649	0.858	0.919	0.761
IForest	0.701	0.297	0.292	0.295	0.105	0.347	0.635	0.301
KNN	0.849	0.78	0.407	0.535	0.489	0.658	0.852	0.593
OCSVM	0.837	0.721	0.389	0.506	0.447	0.659	0.788	0.655
ABOD	0.845	0.782	0.381	0.512	0.472	0.644	0.843	0.584
INNE	0.83	0.689	0.372	0.483	0.418	0.624	0.801	0.646
ALAD	0.819	0.667	0.301	0.415	0.361	0.537	0.7	0.451
LMDD	0.822	1.0	0.168	0.288	0.37	0.624	0.765	0.663
SOD	0.826	0.611	0.513	0.558	0.453	0.621	0.797	0.549
COF	0.834	0.667	0.442	0.532	0.449	0.603	0.774	0.593
LODA	0.83	0.689	0.372	0.483	0.418	0.549	0.692	0.522
LUNAR	0.819	0.743	0.23	0.351	0.313	0.541	0.797	0.46
CBLOF	0.802	0.569	0.292	0.386	0.304	0.45	0.574	0.372
DIF	0.788	1.0	0.009	0.018	0.084	0.494	0.805	0.522
VAE	0.794	0.532	0.292	0.377	0.283	0.446	0.687	0.513
GMM	0.783	0.482	0.239	0.32	0.225	0.426	0.713	0.389
DeepSVDD	0.788	0.509	0.239	0.325	0.241	0.344	0.55	0.336
PCA	0.779	0.464	0.23	0.308	0.21	0.373	0.612	0.363
COPOD	0.767	0.4	0.177	0.245	0.147	0.328	0.627	0.257
SOS	0.758	0.364	0.177	0.238	0.125	0.308	0.524	0.274
ECOD	0.767	0.396	0.168	0.236	0.14	0.34	0.637	0.345

2 Risultati XGBOD su OPS-SAT

Modello	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Score
+Modelli e Param	0.97	0.945	0.912	0.928	0.909	0.973	0.992	0.92
EarlyStop (M+P)	0.97	0.971	0.885	0.926	0.909	0.969	0.99	0.912
Più Modelli	0.968	0.944	0.903	0.923	0.903	0.974	0.91	0.92
Con Parametri	0.964	0.943	0.885	0.913	0.891	0.972	0.991	0.912
Senza Parametri	0.962	0.935	0.885	0.909	0.886	0.977	0.992	0.912
Con Grid	0.947	0.989	0.761	0.86	0.839	0.898	0.945	0.969

3 Risultati Rocket su OPS-SAT

Modalità	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Score
Unsupervised	0.834	0.963	0.23	0.371	0.424	0.726	0.772	0.646
Supervised								
Rocket+RidgeCV	0.977	0.972	0.92	0.945	0.932	0.962	0.984	0.929
Rocket+LogisticReg	0.974	0.963	0.912	0.936	0.92	0.964	0.986	0.92
Rocket+RidgeReg	0.864	0.936	0.389	0.55	0.554	0.871	0.94	0.92
Rocket+KNN	0.834	0.963	0.23	0.371	0.424	0.726	0.772	0.646

`RidgeClassifierCV` → è il modello usato nella demo del paper di rocket ed ho riscontrato metriche migliori

4 Riesecuzione del Paper

Dataset	Accuracy
Coffee	1.0
Computers	0.64
Adiac	0.634
ArrowHead	0.788
BeetleFly	0.8
CinCECGTorso	0.7898
CBF	0.971
ChlorineConcentration	0.590
GunPoint	0.9866
Ham	0.771
HandOutlines	0.935
InlineSkate	0.367
Lightning2	0.623
Mallat	0.928
MiddlePhalanxTW	0.558

5 Risultati Rocket su Dataset BakeOff (paper)

Sono i risultati della mia implementazione (presa dallo stesso paper ma leggermente modificata) con alcuni asset dei dataset di ”BakeOff”.

Dataset	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC
Coffee	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Computers	0.704	0.713	0.704	0.701	0.417	0.358	0.753
Adiac	0.777	0.823	0.777	0.761	0.773	0.79	0.978
ArrowHead	0.771	0.809	0.771	0.756	0.69	0.88	0.942
BeetleFly	0.85	0.885	0.85	0.847	0.834	0.331	1.0
CinCECGTorso	0.823	0.831	0.823	0.822	0.767	0.914	0.963
CBF	0.992	0.992	0.992	0.992	0.988	1.0	1.0
ChlorineConcentration	0.782	0.778	0.782	0.774	0.633	0.813	0.902
GunPoint	1.0	1.0	1.0	1.0	1.0	0.315	1.0
Ham	0.657	0.658	0.657	0.655	0.314	0.37	0.734
HandOutlines	0.916	0.917	0.916	0.915	0.817	0.963	0.96
InlineSkate	0.705	0.708	0.705	0.7	0.404	0.865	0.838
Lightning2	0.738	0.743	0.738	0.733	0.473	0.829	0.805
Mallat	0.943	0.951	0.943	0.944	0.936	0.977	0.994
MiddlePhalanxTW	0.552	0.516	0.552	0.523	0.419	0.438	0.822

6 Codici

6.1 XGBOD

6.1.1 Più Modelli Unsupervised

```python
from pyod.models.xgbod import XGBOD
from pyod.models.knn import KNN
from pyod.models.iforest import IForest
from pyod.models.lof import LOF
from pyod.models.abod import ABOD
from pyod.models.ocsvm import OCSVM

    # Definizione dei modelli unsupervised
unsupervised_models = [ KNN(),
                        LOF(),
                        ABOD(),
                         OCSVM()
                     ]
# Inizializza e addestra XGBOD
model = XGBOD(estimator_list=unsupervised_models)

model.fit(X_train_scaled, y_train)

# Prevedi gli outlier nel dataset di test
y_pred = model.predict(X_test_scaled)
y_predicted_score = model.decision_function(X_test_scaled)
# Eseguiamo la valutazione delle metriche
metrics = evaluate_metrics(y_test, y_pred, y_predicted_score)

# Stampa i risultati
print(model, metrics)
```

6.1.2 Più Modelli e Parametri

```python
from pyod.models.xgbod import XGBOD
from pyod.models.knn import KNN
from pyod.models.lof import LOF
from pyod.models.abod import ABOD
from pyod.models.ocsvm import OCSVM

# Definizione dei modelli unsupervised
unsupervised_models = [ KNN(),
                        LOF(),
                        ABOD(),
                         OCSVM()
                     ]

# Inizializza e addestra XGBOD
model = XGBOD(estimator_list=unsupervised_models,
              n_estimators=100,
              max_depth=3,
              learning_rate=0.2,
              n_jobs=-1,
              random_state=SEED
            )

model.fit(X_train_scaled, y_train)

# Prevedi gli outlier nel dataset di test
y_pred = model.predict(X_test_scaled)
y_predicted_score = model.decision_function(X_test_scaled)

# Eseguiamo la valutazione delle metriche
metrics = evaluate_metrics(y_test, y_pred, y_predicted_score)
print("")
print(metrics)
```

6.1.3 EarlyStopping (M+P)

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from pyod.models.xgbod import XGBOD
from pyod.models.knn import KNN
from pyod.models.iforest import IForest
```

```python
from pyod.models.lof import LOF
from pyod.models.abod import ABOD
from pyod.models.ocsvm import OCSVM

# Definizione dei modelli unsupervised
unsupervised_models = [ KNN(),
                        LOF(),
                        ABOD(),
                         OCSVM()
                     ]

# Divisione del dataset di allenamento per avere un set di validazione
X_train_sub, X_val, y_train_sub, y_val = train_test_split(X_train_scaled, y_train, test_size=0.2, random_state=SEED)

# Inizializzazione del modello
model = XGBOD(estimator_list=unsupervised_models, n_estimators=50, max_depth=3, learning_rate=0.2, n_jobs=-1, random_state=SEED)

best_score = -np.inf
patience = 10       # Numero di volte che il modello cercher  di migliorarsi
patience_counter = 0
n_iterations = 100      # Numero massimo di cicli dell'allenamento

for i in range(n_iterations):  # Numero massimo di iterazioni
    model.fit(X_train_sub, y_train_sub)

    # Predizione sul set di validazione
    y_val_pred = model.predict(X_val)
    val_score = accuracy_score(y_val, y_val_pred)

    # Controllo early stopping
    if val_score > best_score:
        best_score = val_score
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print(f"Early stopping at iteration {i}")
            break
    model.n_estimators += 1  # Incrementa il numero di stimatori per la prossima iterazione

# Predizione sul set di test
y_pred = model.predict(X_test_scaled)
y_predicted_score = model.decision_function(X_test_scaled)

# Eseguiamo la valutazione delle metriche
metrics = evaluate_metrics(y_test, y_pred, y_predicted_score)
print("")
print(metrics)
```

6.1.4 Ricerca Grid

```python
from sklearn.model_selection import RandomizedSearchCV
from pyod.models.xgbod import XGBOD
import numpy as np

# Definizione della griglia di parametri
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [3, 5],
    'learning_rate': [0.01, 0.1]
}

# Inizializza il modello
model = XGBOD()

# Randomized search con meno iterazioni e parallelizzazione
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=10, cv=3, scoring='roc_auc', random_state=42, n_jobs=-1)
random_search.fit(X_train_scaled, y_train)

# Migliori parametri trovati
best_params = random_search.best_params_
print(f"Best parameters found: {best_params}")

# Riaddestramento del modello con i migliori parametri
model = XGBOD(**best_params)
```

```
model.fit(X_train_scaled, y_train)

# Prevedi gli outlier nel dataset di test
y_pred = model.predict(X_test_scaled)
y_predicted_score = model.decision_function(X_test_scaled)

# Eseguiamo la valutazione delle metriche
metrics = evaluate_metrics(y_test, y_pred, y_predicted_score)

# Stampa i risultati
print(model, metrics)
```

6.2 Rocket

6.2.1 Normale (unsupervised)

```
    import numpy as np
import pandas as pd
from numba import njit, prange
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
    matthews_corrcoef, average_precision_score, roc_auc_score

# Funzioni gi  definite in precedenza
def generate_kernels(input_length, num_kernels):
    candidate_lengths = np.array((7, 9, 11), dtype=np.int32)
    lengths = np.random.choice(candidate_lengths, num_kernels)

    weights = np.zeros(lengths.sum(), dtype=np.float64)
    biases = np.zeros(num_kernels, dtype=np.float64)
    dilations = np.zeros(num_kernels, dtype=np.int32)
    paddings = np.zeros(num_kernels, dtype=np.int32)

    a1 = 0
    for i in range(num_kernels):
        _length = lengths[i]
        _weights = np.random.normal(0, 1, _length)
        b1 = a1 + _length
        weights[a1:b1] = _weights - _weights.mean()
        biases[i] = np.random.uniform(-1, 1)
        dilation = 2 ** np.random.uniform(0, np.log2((input_length - 1) / (_length - 1)))
        dilation = np.int32(dilation)
        dilations[i] = dilation
        padding = ((_length - 1) * dilation) // 2 if np.random.randint(2) == 1 else 0
        paddings[i] = padding
        a1 = b1

    return weights, lengths, biases, dilations, paddings

@njit(fastmath=True)
def apply_kernel(X, weights, length, bias, dilation, padding):
    input_length = len(X)
    output_length = (input_length + (2 * padding)) - ((length - 1) * dilation)
    _ppv = 0
    _max = np.NINF
    _mean_sum = 0  # Per calcolare la media
    end = (input_length + padding) - ((length - 1) * dilation)
    for i in range(-padding, end):
        _sum = bias
        index = i
        for j in range(length):
            if index > -1 and index < input_length:
                _sum += weights[j] * X[index]
            index += dilation
        _mean_sum += _sum  # Aggiungi al totale per la media
        if _sum > _max:
            _max = _sum
        if _sum > 0:
            _ppv += 1
    mean_response = _mean_sum / output_length  # Calcola la media
    return _ppv / output_length, _max, mean_response

@njit("float64[:,:](float64[:,:],Tuple((float64[::1],int32[:],float64[:],int32[:],int32[:])))",
    parallel=True, fastmath=True)
def apply_kernels(X, kernels):
    weights, lengths, biases, dilations, paddings = kernels
    num_examples, _ = X.shape
    num_kernels = len(lengths)
```

```
    _X = np.zeros((num_examples, num_kernels * 3), dtype=np.float64)  # 3 features per kernel
    for i in prange(num_examples):
        a1 = 0  # Per i pesi
        a2 = 0  # Per le caratteristiche
        for j in range(num_kernels):
            b1 = a1 + lengths[j]
            b2 = a2 + 3
            _X[i, a2:b2] = apply_kernel(
                X[i], weights[a1:b1], lengths[j], biases[j], dilations[j], paddings[j]
            )
            a1 = b1
            a2 = b2
    return _X

def detect_anomalies_with_threshold(scores, threshold):
    return (scores > threshold).astype(int)

# Genera kernel convoluzionali casuali
input_length = X_train.shape[1]
num_kernels = 10000
kernels = generate_kernels(input_length, num_kernels)

# Applica i kernel alle serie temporali
features_train = apply_kernels(X_train2, kernels)
features_test = apply_kernels(X_test2, kernels)

# Sintesi delle caratteristiche per esempio
anomaly_scores_train = np.mean(features_train, axis=1)  # Media
anomaly_scores_test = np.mean(features_test, axis=1)  # Media

# Rilevamento delle anomalie
threshold = np.percentile(anomaly_scores_train , 95)
anomaly_labels_train = detect_anomalies_with_threshold(anomaly_scores_train , threshold)
anomaly_labels_test = detect_anomalies_with_threshold(anomaly_scores_test , threshold)

# Visualizzazione dei risultati
print("Anomalie rilevate nel training set:", anomaly_labels_train)
print("Anomalie rilevate nel test set:", anomaly_labels_test)

# Eseguiamo la valutazione delle metriche
metrics = evaluate_metrics(y_test, anomaly_labels_test, y_proba=anomaly_scores_test)
print("Metriche di valutazione sul test set:\n", metrics)
# {'Accuracy': 0.832, 'Precision': 0.962, 'Recall': 0.221, 'F1': 0.36, 'MCC': 0.415, 'AUC_PR':
    0.726, 'AUC_ROC': 0.772, 'PREC_N_SCORES': 0.646}
```

6.2.2 Rockad

```
import numpy as np
import pandas as pd

from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.utils import resample
from sktime.transformations.panel.rocket import Rocket
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.metrics.pairwise import distance_metrics


class NearestNeighborOCC():

    def __init__(self, dist="euclidean"):
        self.scores_train = None
        self.dist = None

        metrics = distance_metrics()

        if type(dist) is str and dist in metrics.keys():
            self.dist = metrics[dist]
        elif dist in metrics.values():
            self.dist = dist
        elif False:
            # TODO: allow time series distance measures such as DTW or Matrix Profile
            pass
        else:
            raise Exception("Distance metric not supported.")
```

```
    def fit(self, scores_train):
        _scores_train = scores_train

        if type(_scores_train) is not np.array:
            _scores_train = np.array(scores_train.copy())

        if len(_scores_train.shape) == 1:
            _scores_train = _scores_train.reshape(-1, 1)

        self.scores_train = _scores_train

        return self


    def predict(self, scores_test):
        """
        Per definition (see [1]): 0 indicates an anomaly, 1 indicates normal.
        Here : -1 indicates an anomaly, 1 indicates normal.
        """

        predictions = []
        for score in scores_test:
            predictions.append(self.predict_score(score))
        return np.array(predictions)


    def predict_score(self, anomaly_score):
        prediction = None

        anomaly_score_arr = np.array([anomaly_score for i in range(len(self.scores_train))])

        _scores_train = self.scores_train.copy().reshape(-1, 1)
        anomaly_score_arr = anomaly_score_arr.reshape(-1, 1)
        nearest_neighbor_idx = np.argmin(self.dist(anomaly_score_arr, _scores_train))

        _scores_train = np.delete(_scores_train, nearest_neighbor_idx).reshape(-1, 1)

        nearest_neighbor_score = self.scores_train[nearest_neighbor_idx]
        neares_neighbot_score_arr = np.array([nearest_neighbor_score for i in range(len(
            _scores_train))])
        nearest_neighbor_score_arr = neares_neighbot_score_arr.reshape(-1, 1)

        nearest_nearest_neighbor_idx = np.argmin(self.dist(nearest_neighbor_score_arr, _scores_train
            ))
        nearest_nearest_neighbor_score = _scores_train[nearest_nearest_neighbor_idx][0]

        prediction = self.indicator_function(
            anomaly_score, nearest_neighbor_score, nearest_nearest_neighbor_score)

        return prediction


    def indicator_function(self, z_score, nearest_score, nearest_of_nearest_score):

        # make it an array and reshape it to calculate the distance
        z_score_arr = np.array(z_score).reshape(1, -1)
        nearest_score_arr = np.array(nearest_score).reshape(1, -1)
        nearest_of_nearest_score_arr = np.array(nearest_of_nearest_score).reshape(1, -1)

        numerator = self.dist(z_score_arr, nearest_score_arr)
        denominator = self.dist(nearest_score_arr, nearest_of_nearest_score_arr)

        # error handling for corner cases
        if numerator == 0:
            return 1
        elif denominator == 0:
            return -1
        else:
            return 1 if (numerator/denominator) <= 1 else -1

class NN:

    def __init__(self,
            n_neighbors = 5,
            n_jobs = 1,
            dist = 'euclidean',
            random_state=42,
        ) -> None:

```

```python
        self.n_neighbors = n_neighbors
        self.n_jobs = n_jobs
        self.dist = dist
        self.random_state = random_state


    def fit(self, X):
        self.nn = NearestNeighbors(
            n_neighbors = self.n_neighbors,
            n_jobs = self.n_jobs,
            metric = self.dist,
            algorithm = 'ball_tree',
            )

        self.nn.fit(X)


    def predict_proba(self, X, y=None):
        scores = self.nn.kneighbors(X)
        scores = scores[0].mean(axis=1).reshape(-1,1)

        return scores


class ROCKAD():

    def __init__(self,
            n_estimators=10,
            n_kernels = 100,
            n_neighbors = 5,
            n_jobs = 1,
            power_transform = True,
            random_state = 42,
        ) -> None:
        self.random_state = random_state
        self.power_transform = power_transform

        self.n_estimators = n_estimators
        self.n_kernels = n_kernels
        self.n_neighbors = n_neighbors
        self.n_jobs = n_jobs
        self.n_inf_cols = []

        self.estimator = NN
        self.rocket_transformer = Rocket(num_kernels = self.n_kernels, n_jobs = self.n_jobs,
            random_state = self.random_state)
        self.scaler = StandardScaler()
        self.power_transformer = PowerTransformer(standardize = False)


    def init(self, X):

        # Fit Rocket & Transform into rocket feature space
        Xt = self.rocket_transformer.fit_transform(X)

        self.Xtp = None # X: values, t: (rocket) transformed, p: power transformed

        if self.power_transform is True:

            Xtp = self.power_transformer.fit_transform(Xt)

            self.Xtp = pd.DataFrame(Xtp)

        else:
            self.Xtp = pd.DataFrame(Xt)


    def fit_estimators(self):

        Xtp_scaled = None

        if self.power_transform is True:
            # Check for infinite columns and get indices
            self._check_inf_values(self.Xtp)

            # Remove infinite columns
            self.Xtp = self.Xtp[self.Xtp.columns[~self.Xtp.columns.isin(self.n_inf_cols)]]

            # Fit Scaler
```

```
            Xtp_scaled = self.scaler.fit_transform(self.Xtp)

            Xtp_scaled = pd.DataFrame(Xtp_scaled, columns = self.Xtp.columns)

            self._check_inf_values(Xtp_scaled)

            Xtp_scaled = Xtp_scaled.astype(np.float32).to_numpy()

        else:
            Xtp_scaled = self.Xtp.astype(np.float32).to_numpy()


        self.list_baggers = []

        for idx_estimator in range(self.n_estimators):
            # Initialize estimator
            estimator = self.estimator(
                n_neighbors = self.n_neighbors,
                n_jobs = self.n_jobs,
            )

            # Bootstrap Aggregation
            Xtp_scaled_sample = resample(
                Xtp_scaled,
                replace = True,
                n_samples = None,
                random_state = self.random_state + idx_estimator,
                stratify = None,
            )

            # Fit estimator and append to estimator list
            estimator.fit(Xtp_scaled_sample)
            self.list_baggers.append(estimator)


    def fit(self, X):
        self.init(X)
        self.fit_estimators()

        return self


    def predict_proba(self, X):
        y_scores = np.zeros((len(X), self.n_estimators))

        # Transform into rocket feature space
        Xt = self.rocket_transformer.transform(X)

        Xtp_scaled = None

        if self.power_transform == True:
            # Power Transform using yeo-johnson
            Xtp = self.power_transformer.transform(Xt)
            Xtp = pd.DataFrame(Xtp)

            # Check for infinite columns and remove them
            self._check_inf_values(Xtp)
            Xtp = Xtp[Xtp.columns[~Xtp.columns.isin(self.n_inf_cols)]]
            Xtp_temp = Xtp.copy()

            # Scale the data
            Xtp_scaled = self.scaler.transform(Xtp_temp)
            Xtp_scaled = pd.DataFrame(Xtp_scaled, columns = Xtp_temp.columns)

            # Check for infinite columns and remove them
            self._check_inf_values(Xtp_scaled)
            Xtp_scaled = Xtp_scaled[Xtp_scaled.columns[~Xtp_scaled.columns.isin(self.n_inf_cols)]]
            Xtp_scaled = Xtp_scaled.astype(np.float32).to_numpy()

        else:
            Xtp_scaled = Xt.astype(np.float32)


        for idx, bagger in enumerate(self.list_baggers):
            # Get scores from each estimator
            scores = bagger.predict_proba(Xtp_scaled).squeeze()

            y_scores[:, idx] = scores

```

```
        # Average the scores to get the final score for each time series
        y_scores = y_scores.mean(axis=1)

        return y_scores


    def _check_inf_values(self, X):
        if np.isinf(X[X.columns[~X.columns.isin(self.n_inf_cols)]]).any(axis=0).any() :
            self.n_inf_cols.extend(X.columns.to_series()[np.isinf(X).any()])
            self.fit_estimators()
            return True

# Create the normal dataset (Normal class: Class 1)
#        the anomaly dataset (Anomaly class: Class 2)

RANDOM_STATE = 42
# Initialize and fit ROCKAD

# X_train_array = np.array([x.to_numpy().flatten() for x in X_train.iloc[:, 0]])
# X_test_array = np.array([x.to_numpy().flatten() for x in X_test.iloc[:, 0]])

# Create the normal dataset (Normal class: Class 1)
#        the anomaly dataset (Anomaly class: Class 2)
# print("X_train2: ", X_train2)
# print("X_test: ", X_test)
# print("X_test2: ", X_test2)
# print("y_train: ", y_train)
# print("y_test: ", y_test)

X_normal_train = X_train[y_train == '1']

X_normal_test = X_test[y_test == '1']
X_anomaly_test = X_test[y_test == '2']
y_normal_test = y_test[y_test == '1']
y_anomaly_test = y_test[y_test == '2']

# Merge the test sets
X_test = pd.DataFrame(np.concatenate((X_normal_test, X_anomaly_test), axis=0))
y_test = np.concatenate((y_normal_test, y_anomaly_test), axis=0)

# X_test = np.concatenate((X_normal_test, X_anomaly_test), axis=0)
# y_test = np.concatenate((y_normal_test, y_anomaly_test), axis=0)

rockad = ROCKAD(n_estimators=10, n_kernels=10 ,random_state=RANDOM_STATE)
rockad.fit(X_normal_train)


# Predict anomaly scores
scores = rockad.predict_proba(X_test)

print("Score: ",scores)

# Initialize and fit NearestNeigbor One Class Classifier
decision_func = KNN().fit(scores)

# Predict anomalies
predictions = decision_func.predict(scores)
```

6.2.3 Supervised RegressionCV

```
import numpy as np
import pandas as pd
from sklearn.linear_model import RidgeClassifierCV
from numba import njit, prange
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
    matthews_corrcoef, average_precision_score, roc_auc_score
from pyod.utils.data import precision_n_scores
from scipy.special import softmax

@njit("Tuple((float64[:],int32[:],float64[:],int32[:],int32[:]))(int64,int64)")
def generate_kernels(input_length, num_kernels):

    candidate_lengths = np.array((7, 9, 11), dtype = np.int32)
    lengths = np.random.choice(candidate_lengths, num_kernels)

    weights = np.zeros(lengths.sum(), dtype = np.float64)
    biases = np.zeros(num_kernels, dtype = np.float64)
    dilations = np.zeros(num_kernels, dtype = np.int32)
```

```
    paddings = np.zeros(num_kernels, dtype = np.int32)

    a1 = 0

    for i in range(num_kernels):

        _length = lengths[i]

        _weights = np.random.normal(0, 1, _length)

        b1 = a1 + _length
        weights[a1:b1] = _weights - _weights.mean()

        biases[i] = np.random.uniform(-1, 1)

        dilation = 2 ** np.random.uniform(0, np.log2((input_length - 1) / (_length - 1)))
        dilation = np.int32(dilation)
        dilations[i] = dilation

        padding = ((_length - 1) * dilation) // 2 if np.random.randint(2) == 1 else 0
        paddings[i] = padding

        a1 = b1

    return weights, lengths, biases, dilations, paddings

@njit(fastmath = True)
def apply_kernel(X, weights, length, bias, dilation, padding):

    input_length = len(X)

    output_length = (input_length + (2 * padding)) - ((length - 1) * dilation)

    _ppv = 0
    _max = np.NINF

    end = (input_length + padding) - ((length - 1) * dilation)

    for i in range(-padding, end):

        _sum = bias

        index = i

        for j in range(length):

            if index > -1 and index < input_length:

                _sum = _sum + weights[j] * X[index]

            index = index + dilation

        if _sum > _max:
            _max = _sum

        if _sum > 0:
            _ppv += 1

    return _ppv / output_length, _max

@njit("float64[:,:](float64[:,:],Tuple((float64[::1],int32[:],float64[:],int32[:],int32[:])))",
    parallel = True, fastmath = True)
def apply_kernels(X, kernels):

    weights, lengths, biases, dilations, paddings = kernels

    num_examples, _ = X.shape
    num_kernels = len(lengths)

    _X = np.zeros((num_examples, num_kernels * 2), dtype = np.float64) # 2 features per kernel

    for i in prange(num_examples):

        a1 = 0 # for weights
        a2 = 0 # for features

        for j in range(num_kernels):

            b1 = a1 + lengths[j]
```

```python
            b2 = a2 + 2

            _X[i, a2:b2] = \
            apply_kernel(X[i], weights[a1:b1], lengths[j], biases[j], dilations[j], paddings[j])

            a1 = b1
            a2 = b2

    return _X


# Genera kernel convoluzionali casuali
input_length = X_train.shape[1]
num_kernels = 1000
kernels = generate_kernels(input_length, num_kernels)

# Applica i kernel alle serie temporali
features_train = apply_kernels(X_train2, kernels)
features_test = apply_kernels(X_test2, kernels)


# Addestramento del modello supervisionato
model = RidgeClassifierCV(alphas = np.logspace(-3, 3, 10))
model.fit(features_train, y_train)

# Predizione delle anomalie nei dati di test
y_pred = model.predict(features_test)

if  len(np.unique(y_test)) > 2:
    y_proba = softmax(model.decision_function(features_test), axis=1)
else:
    y_proba = softmax(model.decision_function(features_test), axis=0)

# Visualizzazione dei risultati
print("Predizioni nel test set:", y_pred)

# Eseguiamo la valutazione delle metriche
metrics = evaluate_metrics(y_test, y_pred, y_proba)
print("Metriche di valutazione:\n", metrics)
# {'Accuracy': 0.977, 'Precision': 0.972, 'Recall': 0.92, 'F1': 0.945, 'MCC': 0.932, 'AUC_PR':
    0.962, 'AUC_ROC': 0.984, 'PREC_N_SCORES': 0.929}
```