Zuzanna Górecka, Kamil Szydłowski \ **Nienadzorowana Detekcja Anomalii na Podstawie Niepodobieństwa do Sąsiadów**

Założenia wstępne

1. Nasz projekt polega na samodzielnej implementacji algorytmu nienadzorowanej detekcji anomalii na podstawie niepodobieństwa do sąsiadów wraz z samodzielną implementacją algorytmu *kNN*, a nie na na modyfikacji istniejącej implementacji tego algorytmu (zgodnie z zasadami realizacji projektu wystarczające jest jedno z tych dwóch).
2. Projekt jest zgodny z przedstawiony w dokumentacji wstępnej założeniami, które zostają przypomniane poniżej. Różnice wynikają tylko z innego nazewnictwa atrybutów i metod klas.

Implementacja

Opis implementacji

Opracowana procedura do detekcji anomalii składa się z 3 etapów:

1. Uruchomienia zaimplementowanego przez nas algorytmu k-NN do znalezienia k najbliższych sąsiadów dla każdego przykładu,
2. wyliczenia wskaźnika nieprawidłowości,
3. na podstawie wartości wskaźnika klasyfikacja przykładu jako odstającego lub nie.

Cały algorytm znajduje się w klasie *NNAnomalyDetector*

Parametry:

k: int

liczba sąsiadów

metric: str | Callable

miara niepodobieństwa; może być podana jako 1 z nazw ze słownika sklearn.metrics.pairwise.distance_metrics() albo jako konkretna funkcja

outlier_factor_input: str | Callable

wskaźnik nieprawidłowości; może być podany jako konkretna funkcja lub jedna z nazw zaimplementowanych wskaźników:

- k_distance - niepodobieństwo do k-tego sąsiada
- mean_knn_distance - średnie niepodobieństwo do k sąsiadów
- negative_loc_reachability_density - ujemna lokalna gęstość otoczenia
- lof - Local outlier factor

Atrybuty:

k: int

liczba sąsiadów

metric: Callable

miara niepodobieństwa

outlier_factor_input: Callable

wskaźnik nieprawidłowości

fitted: bool

przuje wartość True jeżeli klasyfikator został nauczony

kNN: KNN

Nauczony algorytm kNN

Metody:

fit(X)

Nauczenie klasyfikatora na podstawie podanych danych uczących X.

> Parametry
>
> - X: np.ndarray\ Dane uczące

predict(X, thresh=None)

Klasyfikacja przykładów jako odstających (klasa 1) lub nie (klasa 0).

> Parametry
>
> - X: np.ndarray\ Dane do klasyfikacji
> - thresh: float\ Parametr thresh przyjmuje wartość punktu odcięcia, dla której przykład klasyfikowany jest jako anomalia. Jeżeli wartość nie jest podana to algorytm zwróci wektor wartości wskaźnika nieprawidłowości dla każdego przykładu. Jeżeli parametr thresh jest podany to algorytm zwróci dodatkowo wektor z informacją o tym, czy dany przykład jest anomalią
>
> Wyjście
>
> - outlier_factor_list: np.ndarray\ wektor wartości wskaźnika nieprawidłowości
> - classes: np.ndarray\ wektor z klasyfikacją czy przykład jest anomalią (klasa 1) czy nie (klasa 0)

Dodatkowo zostanie zaimplementowana klasa do znajdowania k+ najbliższych sąsiadów – *KNN*. W odróżnieniu od algorytmu sklearn.neighbors.NearestNeighbors uwaględniana jest sytuacja, w której algorytm może zwrócić więcej niż k sąsiadów, jeżeli k-ty sąsiad jest tak samo odległy jak k+n-ty sąsiad.

Parametry:

k: int

liczba sąsiadów

metric: string | Callable

miara niepodobieństwa; może być podana jako 1 z nazw ze słownika sklearn.metrics.pairwise.distance_metrics() albo jako konkretna funkcja

Atrybuty

k: int

liczba sąsiadów

metric: string | Callable

miara niepodobieństwa

X_train: np.ndarray

Zbiór uczący

train_distances: List[float]

Lista odległości do k najbliższych sąsiadów dla danych trenujących

train_neigh_idx: List[int]

Wartości indeksów k+ najbliższych sąsiadów dla danych trenujących.

<u>Metody:</u>

fit(X)

Nauczenie algorytmu kNN

> **Parametry**
>
> - **X: np.ndarray\ Dane uczące**

fit(X)

Nauczenie algorytmu kNN

> **Parametry**
>
> - **X: np.ndarray\ Dane uczące**

predict(X)

Zwraca k+ najbliższych sąsiadów.

> **Parametry**
>
> - **X: np.ndarray\ Dane**
>
> **Wyjście**
>
> - **neighbours_dist_list: List[np.ndarray]\ lista wektorów odległości do k najbliższych sąsiadów**
> - **neighbours_idx_list: List[np.ndarray]\ lista wektorów indeksów k+ najbliższych sąsiadów**

fit_predict(X)

Zwraca k+ najbliższych sąsiadów dla danych uczących.

> **Parametry**
>
> - **X: np.ndarray\ Dane uczące**
>
> **Wyjście**
>
> - **neighbours_dist_list: List[np.ndarray]\ lista wektorów odległości do k najbliższych sąsiadów**
> - **neighbours_idx_list: List[np.ndarray]\ lista wektorów indeksów k+ najbliższych sąsiadów**

Kod implementacji

KNN

In [ ]:

```
import numpy as np
from sklearn.metrics.pairwise import distance_metrics
from typing import Callable, List, Tuple
from tqdm import tqdm
```

```
class KNN:
    def __init__(self, k: int, metric: str | Callable):
        self.k = k
        if type(metric)==str:
            distance_metrics_dict = distance_metrics()
            if not metric in distance_metrics_dict.keys():
                raise ValueError("Invalid distance metric name")
            self.metric = distance_metrics_dict[metric]
        elif callable(metric):
            self.metric = metric
        else:
            raise TypeError("Invalid distance metric type")
        self.X_train = None
        self.train_distances = None
        self.train_neigh_idx = None

    def _compute_distances(self, x1: np.ndarray, exclude_index: int=None) -> List[Tuple[
float, int]]:
        dist_idx = []
        for j, x2 in enumerate(self.X_train):
            if exclude_index==j:
                continue
            dist_idx.append((self.metric(x1.reshape(1, -1), x2.reshape(1, -1))[0,0], j)
)
        return dist_idx

    def _choose_kNN(self, dist_idx: List[Tuple[float, int]]) -> Tuple[np.ndarray, np.nda
rray]: # ->???
        dist_idx.sort(key=lambda x: x[0])
        neighbours_dist = []
        neighbours_idx = []
        kth_distance = dist_idx[self.k - 1][0]

        for distance, index in dist_idx:
            if distance <= kth_distance:
                neighbours_dist.append(distance)
                neighbours_idx.append(index)
            else:
                break
        return np.array(neighbours_dist), np.array(neighbours_idx)

    def _fit_predict(self, X: np.ndarray) -> None:
        neighbours_dist_list = []
        neighbours_idx_list = []
        for i, x in enumerate(tqdm(X, desc="Fitting kNN", ncols=100)):
            dist_idx = self._compute_distances(x, exclude_index=i) # main difference bet
ween predict method
            neighbours_dist, neighbours_idx = self._choose_kNN(dist_idx)
            neighbours_dist_list.append(neighbours_dist)
            neighbours_idx_list.append(neighbours_idx)
        self.train_distances = neighbours_dist_list
        self.train_neigh_idx = neighbours_idx_list

    def fit(self, X_train: np.ndarray) -> None:
        self.X_train = X_train
        self._fit_predict(X_train)

    def predict(self, X: np.ndarray) -> tuple[List[np.ndarray], List[np.ndarray]]:
        if len(self.X_train)==0:
            raise ValueError("Model has not been trained yet")
        neighbours_dist_list = []
        neighbours_idx_list = []
        for x in tqdm(X, desc="Predicting kNN", ncols=100):
            dist_idx = self._compute_distances(x)
            neighbours_dist, neighbours_idx = self._choose_kNN(dist_idx)
            neighbours_dist_list.append(neighbours_dist)
            neighbours_idx_list.append(neighbours_idx)
        return neighbours_dist_list, neighbours_idx_list

    def fit_predict(self, X: np.ndarray) -> tuple[List[np.ndarray], List[np.ndarray]]:
        self.fit(X)
```

```
        return self.train_distances, self.train_neigh_idx
```

NN Anomaly Detector

In [ ]:

```
import numpy as np
from sklearn.metrics.pairwise import distance_metrics
from typing import Callable, List, Union


class NNAnomalyDetector:
    def __init__(self, k: int, metric: str | Callable, outlier_factor_input: str | Callable):
        self.k = k
        self.metric = metric

        if callable(outlier_factor_input):
            self.outlier_factor = outlier_factor_input
        elif type(outlier_factor_input)==str:
          outlier_factor_dict = {'k_distance': self._k_distance, 'mean_knn_distance': self._mean_knn_distance, "negative_loc_reachability_density": self._negative_loc_reachability_density, "lof": self._lof}
          outlier_factor = outlier_factor_dict.get(outlier_factor_input)
          if outlier_factor==None:
              raise ValueError("Invalid outlier factor metric name")
          self.outlier_factor = outlier_factor
        else:
            raise TypeError("Invalid outlier_factor_input type")

        self.fitted = False

    def _k_distance(self, distances: np.ndarray, *argv) -> float:
        return distances[-1]

    def _mean_knn_distance(self, distances: np.ndarray, *argv) -> float:
        return distances.mean()

    def _reachability(self, distance: np.ndarray, neighbour_idx: int) -> float:
        neigh_dist = self.kNN.train_distances[neighbour_idx]
        return max(distance, self._k_distance(neigh_dist))

    def _negative_loc_reachability_density(self, distances: np.ndarray, neighbours_idx: np.ndarray) -> float:
        reachability_sum = 0
        for distance, neighbour_idx in zip(distances, neighbours_idx):
            reachability_sum += self._reachability(distance, neighbour_idx)
        return -1/(reachability_sum/len(neighbours_idx))

    def _lof(self, distances: np.ndarray, neighbours_idx: np.ndarray) -> float:
        sum = 0
        for distance, neighbour_idx in zip(distances, neighbours_idx):
            neigh_distances = self.kNN.train_distances[neighbour_idx]
            neigh_neighbours_idx = self.kNN.train_neigh_idx[neighbour_idx]
            sum += self._negative_loc_reachability_density(neigh_distances, neigh_neighbours_idx)
        return sum/self._negative_loc_reachability_density(distances, neighbours_idx)/len(neighbours_idx)

    def fit(self, X: np.ndarray) -> None:
        self.kNN = KNN(self.k, self.metric)
        self.kNN.fit(X)
        self.fitted = True

    def predict(self, X: np.ndarray, thresh: float = None) -> np.ndarray | tuple[np.ndarray, np.ndarray]:
        if not self.fitted:
            raise ValueError("Model has not been trained yet")

        distances, neighbours_idx = self.kNN.predict(X)
```

```
        outlier_factor_list = []
        for example_distances, example_neighbours_idx in zip(distances, neighbours_idx):
            example_outlier_factor = self.outlier_factor(example_distances, example_neig
hbours_idx)
            outlier_factor_list.append(example_outlier_factor)

        if thresh is not None:
            return outlier_factor_list, (np.array(outlier_factor_list) > thresh).astype(
int)
        else:
            return outlier_factor_list
```

Eksperymenty

Opis eksperymentów

Podstawowym celem eksperymentów jest rozstrzygnięcie, czy nasza implementacja algorytmu k-NN w zadaniu detekcji anomalii przewyższa jakość klasyfikacji jednoklasowej gotowych modeli: *LocalOutlierFactor*, *OneClassSVM* i *IsolationForest*. W tym celu trenujemy i przeprowadzamy ocenę podanych modeli.

Podczas eksperymentów badamy wpływ hiperparametru K – liczba sąsiadów w algorytmie k-NN na wynik jego działania. Ponadto wyznaczamy wartości różnych miar niepodobieństwa i wskaźników nieprawidłowości oraz czas działania danej metody.

Ten wynik umieszczamy w kontekście działania gotowych modeli klasyfikacji jednoklasowej z domyślnymi parametrami.

Jako źródła danych wykorzystujemy podane niżej zbiory z repozytorium Outlier Detection Datasets. Zbiory do oceny jakości (Speech, Satelite, ForestCover) będą miały realistyczne rozmiary (więcej niż 1000 przykładów, przynajmniej 10 atrybutów o rozkładzie ciągłym).

1. **Speech - Duża wymiarowość\ Przykłady: 3686 \ Wymiarowość: 400 \ Przykłady odstające: 61 (1,65%)**
2. **Satellite - Stosunkowo duża liczba anomalii \ Przykłady: 6435 \ Wymiarowość: 36 \ Przykłady odstające: 2036 (32%)**
3. **ForestCover - Stosunkowo mała liczba anomalii \ Przykłady: 286048 \ Wymiarowość: 10 \ Przykłady odstające: 2747 (0.9%)**

Ponadto do celów deweloperskich wykorzystujemy zbiór *Wine dataset*, którego rozmiary są znacznie mniejsze od pozostałych. W związku z tym, że nie jest on wystarczająco reprezentacyjny, nie przedstawimy wniosków na podstawie uzyskanych na nim wyników.

Wstępne przygotowanie danych obejmuje ich normalizację, aby średnia wynosiła 0 a odchylenie standardowe 1.

Porównanie algorytmów k-NN ze względu na różne wartości k przeprowadzamy przy użyciu wartości pola pod krzywą ROC (AUC) z uwagi na to, że wskaźnik nieprawidłowości na wyjściu ma postać zmiennej jednowymiarowej ciągłej.

Następnie wskaźnik nieprawidłowości przekształcamy na wartość binarną przy użyciu punktu odcięcia (threshold). Optymalny punkt odcięcia to taki, który znajduje się najbliżej lewego górnego rogu wykresu (maksymalizując wartość AUC).

Na tej podstawie porównujemy między sobą nasz model i gotowe klasyfikatory, używając miary F1, która uwzględnia zarówno precyzję, jak i czułość.

Korzystamy z dostepnej w zbiorach danych etykiety: 0 - wartość nieodstająca; 1 - anomalia. \ Zbiór danych dzielimy na:

- **Zbiór treningowy - tylko wartości nieodstające (klasa 0) - używany do uczenia modeli**
- **Zbiór walidacyjny - obie klasy - używany do wyznaczenia punktu odcięcia (thresholdu)**
- **Zbiór testowy - obie klasy - użymany do oceny jakości modeli (AUROC, F1)**

Ze względu na dużą liczbę przykładów w zbiorach danych, do treningu używamy wyłącznie 0.3 danych. Ponadto ze zbioru *Forest Cover* losujemy najpierw próbkę 0.015 danych. Pozwala to przeprowadzić zadowalającą liczbę eksperymentów w rozsądnym, aczkolwiek nadal długim (kiludziesięciominutowym), czasie.

Kod eksperymentów

Przygotowanie

In [ ]:

```
import numpy as np
import pandas as pd
import scipy.io
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, f1_score, roc_curve
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from sklearn.ensemble import IsolationForest
import time
```

In [ ]:

```
def prepare_data(path, val_size, test_size, sample_fraction=None):
    data = scipy.io.loadmat(path)

    if sample_fraction is not None:
        sample_size = int(sample_fraction * data['X'].shape[0])
        indices = np.random.choice(range(data['X'].shape[0]), size=sample_size, replace=
False)
        data['X'] = data['X'][indices]
        data['y'] = data['y'][indices]

    # Training dataset should contain only inliers samples
    data_X_0 = data['X'][data['y'].ravel() == 0]
    data_y_0 = data['y'][data['y'].ravel() == 0]

    X_train, X_val_test, y_train, y_val_test = train_test_split(data_X_0, data_y_0, test
_size=val_size+test_size, random_state=42)

    # After that, val and test datasets should constist of both inliers and outliers samp
les
    X_val_test = np.append(X_val_test, data['X'][data['y'].ravel() == 1], axis=0)
    y_val_test = np.append(y_val_test, data['y'][data['y'].ravel() == 1], axis=0)

    relative_test_size = test_size / (val_size + test_size)
    X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test, test_size=re
lative_test_size, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    return X_train, X_val, X_test, y_train, y_val, y_test
```

In [ ]:

```
def use_NNAnomalyDetector(X_train, X_val, X_test, k, metric, outlier_factor_input):
    nn_detector = NNAnomalyDetector(k=k, metric=metric, outlier_factor_input=outlier_fac
tor_input)
    start_time = time.time()
    nn_detector.fit(X_train)
    factor_val = nn_detector.predict(X_val)
    factor_test = nn_detector.predict(X_test)
    end_time = time.time()
```

```
    return factor_val, factor_test, end_time - start_time
```

In [ ]:

```
def use_LocalOutlierFactor(X_train, X_val, X_test, k, metric):
    lof = LocalOutlierFactor(n_neighbors=k, metric=metric, novelty=True)
    start_time = time.time()
    lof.fit(X_train)
    factor_val = -(lof.decision_function(X_val) + lof.offset_)
    factor_test = -(lof.decision_function(X_test) + lof.offset_)
    end_time = time.time()

    return factor_val, factor_test, end_time - start_time
```

In [ ]:

```
def use_OneClassSVM(X_train, X_val, X_test, kernel='rbf', degree=3, nu=0.5, gamma='scale
'):
    svm_detector = OneClassSVM(kernel=kernel, degree=degree, nu=nu, gamma=gamma)
    start_time = time.time()
    svm_detector.fit(X_train)
    factor_val = -svm_detector.decision_function(X_val)
    factor_test = -svm_detector.decision_function(X_test)
    end_time = time.time()

    return factor_val, factor_test, end_time - start_time
```

In [ ]:

```
def use_IsolationForest(X_train, X_val, X_test, n_estimators=100, max_samples='auto', co
ntamination='auto', random_state=None):
    iso_forest = IsolationForest(n_estimators=n_estimators, max_samples=max_samples, con
tamination=contamination, random_state=random_state)
    start_time = time.time()
    iso_forest.fit(X_train)
    factor_val = -iso_forest.decision_function(X_val)
    factor_test = -iso_forest.decision_function(X_test)
    end_time = time.time()

    return factor_val, factor_test, end_time - start_time
```

In [ ]:

```
def evaluate_model_output(factor_val, factor_test, y_val, y_test):
    auc_score = roc_auc_score(y_true=y_test, y_score=factor_test)
    fpr, tpr, thresholds = roc_curve(y_true=y_val, y_score=factor_val)
    optimal_idx = np.argmax(tpr - fpr)
    optimal_threshold = thresholds[optimal_idx]
    binary_predictions = (np.array(factor_test) > optimal_threshold).astype(int)
    f1 = f1_score(y_test, binary_predictions)

    return auc_score, f1
```

In [ ]:

```
# Funkcja do trenowania i oceniania modeli
def evaluate_models(data, param_values, dataset_name):
    X_train, X_val, X_test, y_train, y_val, y_test = data

    results = []

    for param in param_values:
        tqdm.write(f"param: {param}")

        # NNAnomalyDetector - euclidean - mean_knn_distance
        tqdm.write(f"Model: NNAnomalyDetector - euclidean - mean_knn_distance")
        factor_val, factor_test, time = use_NNAnomalyDetector(X_train, X_val, X_test, pa
ram, metric='euclidean', outlier_factor_input='mean_knn_distance')
        auc_score, f1 = evaluate_model_output(factor_val, factor_test, y_val, y_test)
        results.append({'dataset': dataset_name, 'param': param, 'model': 'NNAnomalyDete
ctor - euclidean - mean_knn_distance', 'AUC': auc_score, 'F1': f1, 'time': time})
```

```
        # NNAnomalyDetector - euclidean - lof
        tqdm.write(f"Model: NNAnomalyDetector - euclidean - lof")
        factor_val, factor_test, time = use_NNAnomalyDetector(X_train, X_val, X_test, pa
ram, metric='euclidean', outlier_factor_input='lof')
        auc_score, f1 = evaluate_model_output(factor_val, factor_test, y_val, y_test)
        results.append({'dataset': dataset_name, 'param': param, 'model': 'NNAnomalyDete
ctor - euclidean - lof', 'AUC': auc_score, 'F1': f1, 'time': time})

        # NNAnomalyDetector - manhattan - lof
        tqdm.write(f"Model: NNAnomalyDetector - manhattan - lof")
        factor_val, factor_test, time = use_NNAnomalyDetector(X_train, X_val, X_test, pa
ram, metric='manhattan', outlier_factor_input='lof')
        auc_score, f1 = evaluate_model_output(factor_val, factor_test, y_val, y_test)
        results.append({'dataset': dataset_name, 'param': param, 'model': 'NNAnomalyDete
ctor - manhattan - lof', 'AUC': auc_score, 'F1': f1, 'time': time})

        # LocalOutlierFactor - euclidean
        tqdm.write(f"Model: LocalOutlierFactor - euclidean")
        factor_val, factor_test, time = use_LocalOutlierFactor(X_train, X_val, X_test, p
aram, metric='euclidean')
        auc_score, f1 = evaluate_model_output(factor_val, factor_test, y_val, y_test)
        results.append({'dataset': dataset_name, 'param': param, 'model': 'LocalOutlierF
actor - euclidean', 'AUC': auc_score, 'F1': f1, 'time': time})

        # LocalOutlierFactor - manhattan
        tqdm.write(f"Model: LocalOutlierFactor - manhattan")
        factor_val, factor_test, time = use_LocalOutlierFactor(X_train, X_val, X_test, p
aram, metric='manhattan')
        auc_score, f1 = evaluate_model_output(factor_val, factor_test, y_val, y_test)
        results.append({'dataset': dataset_name, 'param': param, 'model': 'LocalOutlierF
actor - manhattan', 'AUC': auc_score, 'F1': f1, 'time': time})

        # OneClassSVM
        tqdm.write(f"Model: OneClassSVM")
        factor_val, factor_test, time = use_OneClassSVM(X_train, X_val, X_test, kernel='
rbf', degree=param, nu=0.5, gamma='scale')
        auc_score, f1 = evaluate_model_output(factor_val, factor_test, y_val, y_test)
        results.append({'dataset': dataset_name, 'param': param, 'model': 'OneClassSVM',
'AUC': auc_score, 'F1': f1, 'time': time})

        # IsolationForest
        tqdm.write(f"Model: IsolationForest")
        n_rep = 5
        auc_scores = []
        f1_scores = []
        durations = []
        for _ in range(n_rep):
            factor_val, factor_test, time = use_IsolationForest(X_train, X_val, X_test,
n_estimators=param, max_samples='auto', contamination='auto', random_state=None)
            auc_score, f1 = evaluate_model_output(factor_val, factor_test, y_val, y_test
)
            auc_scores.append(auc_score)
            f1_scores.append(f1)
            durations.append(time)
        results.append({'dataset': dataset_name, 'param': param, 'model': 'Isolation For
est', 'AUC': np.mean(auc_scores), 'F1': np.mean(f1_scores), 'time': np.mean(durations)})

    return results
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

def make_plot(results_df):
    sns.set(style="whitegrid")

    # Zdefiniowanie różnych stylów linii
    #line_styles = ['-', '--', '-.', ':', (0, (3, 1, 1, 1)), (0, (5, 10)), (0, (5, 5))]
    line_styles = ['-', '--', '-.', ':', (0, (3, 1, 1, 1)), (0, (5, 10)), (0, (5, 5)),
```

```
(0, (3, 5, 1, 5)), (0, (1, 1))]
    unique_models = results_df['model'].unique()
    style_mapping = {model: line_styles[i % len(line_styles)] for i, model in enumerate(
unique_models)}

    # Wykres AUC dla różnych wartości k i modeli
    plt.figure(figsize=(14, 6))

    plt.subplot(1, 2, 1)
    for model in unique_models:
        subset = results_df[results_df['model'] == model]
        sns.lineplot(data=subset, x='param', y='AUC', label=model, marker='o', linestyle
=style_mapping[model])
    plt.title('Wartość AUC dla różnych wartości k/degree/n_estimators')
    plt.xlabel('k/degree/n_estimators')
    plt.ylabel('AUC')
    plt.legend(title='Model')

    # Wykres F1 dla różnych wartości k i modeli
    plt.subplot(1, 2, 2)
    for model in unique_models:
        subset = results_df[results_df['model'] == model]
        sns.lineplot(data=subset, x='param', y='F1', label=model, marker='o', linestyle=
style_mapping[model])
    plt.title('Wartość F1 dla różnych wartości k/degree/n_estimators')
    plt.xlabel('k/degree/n_estimators')
    plt.ylabel('F1')
    plt.legend(title='Model')

    # Wyświetlenie wykresów
    plt.tight_layout()
    plt.show()
```

Wine

In [ ]:

```
# Wartości k do przetestowania
k_values = [1, 2, 3, 5, 10]

# Przygotowanie danych
!wget -O wine.mat "https://www.dropbox.com/s/uvjaudt2uto7zal/wine.mat?dl=1"
data = prepare_data('./wine.mat', 0.25, 0.25)

# Przeprowadzenie eksperymentów
results = evaluate_models(data, k_values, 'wine')

# Konwersja wyników na DataFrame i wyświetlenie
wine_results_df = pd.DataFrame(results)
wine_results_df
```

```
--2024-06-08 15:37:40--  https://www.dropbox.com/s/uvjaudt2uto7zal/wine.mat?dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.66.18, 2620:100:6022:18::a27d:4212
Connecting to www.dropbox.com (www.dropbox.com)|162.125.66.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /scl/fi/diwrjqz912rfqpd/wine.mat?rlkey=8e8vixs2sx2t9x5c1i7eaj3qs&dl=1 [followin
g]
--2024-06-08 15:37:41--  https://www.dropbox.com/scl/fi/diwrjqz912rfqpd/wine.mat?rlkey=8e
8vixs2sx2t9x5c1i7eaj3qs&dl=1
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc53dff3d0820fa4c2d2d9dfaaea.dl.dropboxusercontent.com/cd/0/inline/CUcb
odNBSrs9cXJl_nQW7yrJD6BAEV16e6ChBN8dBx9e76JJeeoGXPtHbg5XPDJjZSR-ndrwBxG02IOb5F58p1dACPhUe
ngf5wHm1rvJkeRLQI2rl0RZ2n78Z7tl6VClSpo/file?dl=1# [following]
--2024-06-08 15:37:41--  https://uc53dff3d0820fa4c2d2d9dfaaea.dl.dropboxusercontent.com/c
d/0/inline/CUcbodNBSrs9cXJl_nQW7yrJD6BAEV16e6ChBN8dBx9e76JJeeoGXPtHbg5XPDJjZSR-ndrwBxG02I
Ob5F58p1dACPhUengf5wHm1rvJkeRLQI2rl0RZ2n78Z7tl6VClSpo/file?dl=1
Resolving uc53dff3d0820fa4c2d2d9dfaaea.dl.dropboxusercontent.com (uc53dff3d0820fa4c2d2d9d
faaea.dl.dropboxusercontent.com)... 162.125.13.15, 2620:100:601c:15::a27d:60f
Connecting to uc53dff3d0820fa4c2d2d9dfaaea.dl.dropboxusercontent.com (uc53dff3d0820fa4c2d
2d9dfaaea.dl.dropboxusercontent.com)|162.125.13.15|:443... connected.
```

```
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/CUe3YY1opOwhPkze3JmSr80IerAeku7ZeDkBlPOTm6B2TF5XjG7GPVFJsCtCSTwAh
piyD4bW3gP183Cpptq7jFTplpY1lwNPCuE5bkIbNuS6oyFJsf1FBnbdl1o5atpw89QaJRwXpS587KuWb9dG7uaaLi
8mTU-_OVUBqD3Z0e6byUZZpa56tF97u7qDchMQ8Teatv8gC0A9IU2LVVAV--sVewo_UPdSFa-6TXbC1aX__3Q-N2K
nWYn0aIhRl7Ko5HOPsL7FZYaOgd3KzqQEchgl_KgR6RS_RsTkYo24i0E1UoINJvTxQCo4vB1BcPJaJ19euLmKq1_Q
uYPtkmp9oRHTnjZNrDXOe226aSJ0-F--2g/file?dl=1 [following]
--2024-06-08 15:37:42--  https://uc53dff3d0820fa4c2d2d9dfaaea.dl.dropboxusercontent.com/c
d/0/inline2/CUe3YY1opOwhPkze3JmSr80IerAeku7ZeDkBlPOTm6B2TF5XjG7GPVFJsCtCSTwAhpiyD4bW3gP18
3Cpptq7jFTplpY1lwNPCuE5bkIbNuS6oyFJsf1FBnbdl1o5atpw89QaJRwXpS587KuWb9dG7uaaLi8mTU-_OVUBqD
3Z0e6byUZZpa56tF97u7qDchMQ8Teatv8gC0A9IU2LVVAV--sVewo_UPdSFa-6TXbC1aX__3Q-N2KnWYn0aIhRl7K
o5HOPsL7FZYaOgd3KzqQEchgl_KgR6RS_RsTkYo24i0E1UoINJvTxQCo4vB1BcPJaJ19euLmKq1_QuYPtkmp9oRHT
njZNrDXOe226aSJ0-F--2g/file?dl=1
Reusing existing connection to uc53dff3d0820fa4c2d2d9dfaaea.dl.dropboxusercontent.com:443
.
HTTP request sent, awaiting response... 200 OK
Length: 4078 (4.0K) [application/binary]
Saving to: 'wine.mat'

wine.mat            100%[===================>]   3.98K  --.-KB/s    in 0s

2024-06-08 15:37:42 (2.02 GB/s) - 'wine.mat' saved [4078/4078]

param: 1
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 47.08it/s]
Predicting kNN: 100%|██████████| 35/35 [00:01<00:00, 34.80it/s]
Predicting kNN: 100%|██████████| 35/35 [00:01<00:00, 30.12it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 42.25it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 63.17it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 62.76it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████| 59/59 [00:00<00:00, 65.79it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 75.23it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 70.71it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 2
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 51.79it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 41.65it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 35.76it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 38.95it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 41.85it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 40.32it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Model: NNAnomalyDetector - manhattan - lof
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 36.60it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 37.73it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 36.11it/s]
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 3
Model: NNAnomalyDetector - euclidean - mean_knn_distance
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 45.24it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 50.21it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 39.54it/s]
Model: NNAnomalyDetector - euclidean - lof
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 45.16it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 41.23it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 42.45it/s]
Model: NNAnomalyDetector - manhattan - lof
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 58.82it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 83.86it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 68.98it/s]
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 5
Model: NNAnomalyDetector - euclidean - mean_knn_distance
Fitting kNN: 100%|██████████| 59/59 [00:00<00:00, 59.64it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 60.55it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 68.28it/s]
Model: NNAnomalyDetector - euclidean - lof
Fitting kNN: 100%|██████████| 59/59 [00:01<00:00, 33.09it/s]
Predicting kNN: 100%|██████████| 35/35 [00:01<00:00, 27.95it/s]
Predicting kNN: 100%|██████████| 35/35 [00:01<00:00, 26.44it/s]
Model: NNAnomalyDetector - manhattan - lof
Fitting kNN: 100%|██████████| 59/59 [00:00<00:00, 67.47it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 91.35it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00, 49.87it/s]
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
```

```
Model: OneClassSVM
Model: IsolationForest
param: 10
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████| 59/59 [00:00<00:00,
121.70it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00,
124.91it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00,
126.96it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 59/59 [00:00<00:00,
128.22it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00,
133.31it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00,
122.63it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████| 59/59 [00:00<00:00,
147.22it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00,
133.12it/s]
Predicting kNN: 100%|██████████| 35/35 [00:00<00:00,
122.58it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
```

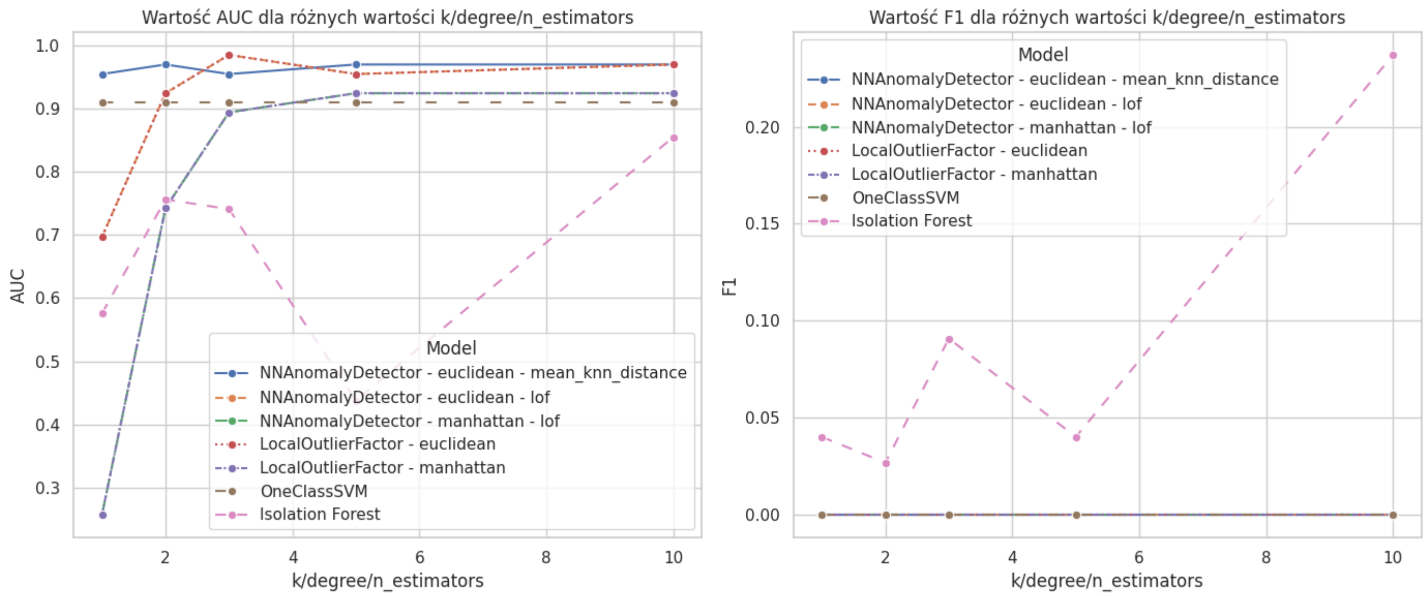
Out[ ]:

	dataset	param	model	AUC	F1	time
0	wine	1	NNAnomalyDetector - euclidean - mean_knn_distance	0.954545	0.000000	3.473277
1	wine	1	NNAnomalyDetector - euclidean - lof	0.696970	0.000000	2.562311
2	wine	1	NNAnomalyDetector - manhattan - lof	0.257576	0.000000	1.919073
3	wine	1	LocalOutlierFactor - euclidean	0.696970	0.000000	0.003701
4	wine	1	LocalOutlierFactor - manhattan	0.257576	0.000000	0.003893
5	wine	1	OneClassSVM	0.909091	0.000000	0.002481
6	wine	1	Isolation Forest	0.575758	0.040000	0.010272
7	wine	2	NNAnomalyDetector - euclidean - mean_knn_distance	0.969697	0.000000	3.042224
8	wine	2	NNAnomalyDetector - euclidean - lof	0.924242	0.000000	3.293576
9	wine	2	NNAnomalyDetector - manhattan - lof	0.742424	0.000000	3.547822
10	wine	2	LocalOutlierFactor - euclidean	0.924242	0.000000	0.003534
11	wine	2	LocalOutlierFactor - manhattan	0.742424	0.000000	0.013413
12	wine	2	OneClassSVM	0.909091	0.000000	0.002355
13	wine	2	Isolation Forest	0.756061	0.026667	0.021635
14	wine	3	NNAnomalyDetector - euclidean - mean_knn_distance	0.954545	0.000000	2.954865
15	wine	3	NNAnomalyDetector - euclidean - lof	0.984848	0.000000	3.040112
16	wine	3	NNAnomalyDetector - manhattan - lof	0.893939	0.000000	1.996876
17	wine	3	LocalOutlierFactor - euclidean	0.984848	0.000000	0.003608
18	wine	3	LocalOutlierFactor - manhattan	0.893939	0.000000	0.003673
19	wine	3	OneClassSVM	0.909091	0.000000	0.002375

	dataset	param	model	AUC	F1	time
20	wine	3	Isolation Forest	0.740909	0.090769	0.015391
21	wine	5	NNAnomalyDetector - euclidean - mean_knn_distance	0.969697	0.000000	2.172614
22	wine	5	NNAnomalyDetector - euclidean - lof	0.954545	0.000000	4.433266
23	wine	5	NNAnomalyDetector - manhattan - lof	0.924242	0.000000	2.011383
24	wine	5	LocalOutlierFactor - euclidean	0.954545	0.000000	0.026358
25	wine	5	LocalOutlierFactor - manhattan	0.924242	0.000000	0.011371
26	wine	5	OneClassSVM	0.909091	0.000000	0.008926
27	wine	5	Isolation Forest	0.437879	0.040000	0.041946
28	wine	10	NNAnomalyDetector - euclidean - mean_knn_distance	0.969697	0.000000	1.073749
29	wine	10	NNAnomalyDetector - euclidean - lof	0.969697	0.000000	1.044313
30	wine	10	NNAnomalyDetector - manhattan - lof	0.924242	0.000000	0.985311
31	wine	10	LocalOutlierFactor - euclidean	0.969697	0.000000	0.003965
32	wine	10	LocalOutlierFactor - manhattan	0.924242	0.000000	0.005157
33	wine	10	OneClassSVM	0.909091	0.000000	0.002150
34	wine	10	Isolation Forest	0.854545	0.237143	0.024516

In [ ]:

```
make_plot(wine_results_df)
```



Speech

In [ ]:

```
# Wartości k do przetestowania
k_values = [1, 2, 3, 5, 10]

# Przygotowanie danych
!wget -O speech.mat "https://www.dropbox.com/s/w6xv51ctea6uauc/speech.mat?dl=1"
data = prepare_data('./speech.mat', 0.35, 0.35)

# Przeprowadzenie eksperymentów
results = evaluate_models(data, k_values, 'speech')

# Konwersja wyników na DataFrame i wyświetlenie
speech_results_df = pd.DataFrame(results)
speech_results_df
```

```
--2024-06-08 15:38:22--  https://www.dropbox.com/s/w6xv51ctea6uauc/speech.mat?dl=1
```

```
Resolving www.dropbox.com (www.dropbox.com)... 162.125.13.18, 2620:100:6057:18::a27d:d12
Connecting to www.dropbox.com (www.dropbox.com)|162.125.13.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /scl/fi/u40qqxk0lpyawhq/speech.mat?rlkey=21i7jmexat2y6fch7zeitvkrl&dl=1 [follow
ing]
--2024-06-08 15:38:22--  https://www.dropbox.com/scl/fi/u40qqxk0lpyawhq/speech.mat?rlkey=
21i7jmexat2y6fch7zeitvkrl&dl=1
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc184d6a57d438eb9214757af938.dl.dropboxusercontent.com/cd/0/inline/CUdS
AMF4SmKcOWxWdt6u4pw7Oeem5B8tKfwHM2d-OApFSViSbtAisQ3QtMeDLO7kitKdiOahOp64DsjUiD1jweUrpbwZr
cwalie8_P6VNhPgwJaO4ikVLGVdu6BC3noJakY/file?dl=1# [following]
--2024-06-08 15:38:23--  https://uc184d6a57d438eb9214757af938.dl.dropboxusercontent.com/c
d/0/inline/CUdSAMF4SmKcOWxWdt6u4pw7Oeem5B8tKfwHM2d-OApFSViSbtAisQ3QtMeDLO7kitKdiOahOp64Ds
jUiD1jweUrpbwZrcwalie8_P6VNhPgwJaO4ikVLGVdu6BC3noJakY/file?dl=1
Resolving uc184d6a57d438eb9214757af938.dl.dropboxusercontent.com (uc184d6a57d438eb9214757
af938.dl.dropboxusercontent.com)... 162.125.13.15, 2620:100:6022:15::a27d:420f
Connecting to uc184d6a57d438eb9214757af938.dl.dropboxusercontent.com (uc184d6a57d438eb921
4757af938.dl.dropboxusercontent.com)|162.125.13.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/CUeE6gGEfNCXHa_dHPDNuPxHhFhw1_YK9DOJC6m3Irp4NXbW_TcsjRs9uxR_UFMj-
zZ8mwDn5r1N-rp3hBsG6kRwt1min1LgQEbqYvP-r7KeXmo4HhnkZdk_JhQ0IDN-CmQYSgAmUrQ6Bo4sYcW3s3z3JG
xZUTUODa0vv8s1_e0K4D1jDgJuylv3D_5pVqb1nk6HFqsM8VJdwjw4cLm7Nu-hwFiKAr8pB3tWcmWZp4-nc-LOSa_
l-IpSva4DlFcJmDLVRvbP6Do82kTx2DpNK96GBaIRdR5_uOj-E5yw7hAoWW4GkVj40Lg553oZntYGCEIHH0DiKcw_
jbi9HMDi-EY_mkGcF9bCwbMw9LU4lb-adQ/file?dl=1 [following]
--2024-06-08 15:38:23--  https://uc184d6a57d438eb9214757af938.dl.dropboxusercontent.com/c
d/0/inline2/CUeE6gGEfNCXHa_dHPDNuPxHhFhw1_YK9DOJC6m3Irp4NXbW_TcsjRs9uxR_UFMj-zZ8mwDn5r1N-
rp3hBsG6kRwt1min1LgQEbqYvP-r7KeXmo4HhnkZdk_JhQ0IDN-CmQYSgAmUrQ6Bo4sYcW3s3z3JGxZUTUODa0vv8
s1_e0K4D1jDgJuylv3D_5pVqb1nk6HFqsM8VJdwjw4cLm7Nu-hwFiKAr8pB3tWcmWZp4-nc-LOSa_l-IpSva4DlFc
JmDLVRvbP6Do82kTx2DpNK96GBaIRdR5_uOj-E5yw7hAoWW4GkVj40Lg553oZntYGCEIHH0DiKcw_jbi9HMDi-EY_
mkGcF9bCwbMw9LU4lb-adQ/file?dl=1
Reusing existing connection to uc184d6a57d438eb9214757af938.dl.dropboxusercontent.com:443
.
HTTP request sent, awaiting response... 200 OK
Length: 9509570 (9.1M) [application/binary]
Saving to: ‘speech.mat’

speech.mat          100%[===================>]   9.07M  49.2MB/s    in 0.2s

2024-06-08 15:38:24 (49.2 MB/s) - ‘speech.mat’ saved [9509570/9509570]

param: 1
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████████████████████████████████| 1087/1087 [02:57<00:00,
6.14it/s]
Predicting kNN: 100%|███████████████████████████████████| 1299/1299 [03:29<00:00,
6.19it/s]
Predicting kNN: 100%|███████████████████████████████████| 1300/1300 [03:28<00:00,
6.23it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████████████████████████████████| 1087/1087 [02:54<00:00,
6.23it/s]
Predicting kNN: 100%|███████████████████████████████████| 1299/1299 [03:29<00:00,
6.20it/s]
Predicting kNN: 100%|███████████████████████████████████| 1300/1300 [03:28<00:00,
6.23it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████████████████████████████████| 1087/1087 [02:47<00:00,
6.51it/s]
Predicting kNN: 100%|███████████████████████████████████| 1299/1299 [03:20<00:00,
6.49it/s]
Predicting kNN: 100%|███████████████████████████████████| 1300/1300 [03:20<00:00,
6.47it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 2
```

```
param: 2
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:55<00:00, 6.21it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:29<00:00, 6.20it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:27<00:00, 6.27it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:54<00:00, 6.23it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:28<00:00, 6.24it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:28<00:00, 6.24it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:47<00:00, 6.49it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:20<00:00, 6.47it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:20<00:00, 6.50it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 3
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:54<00:00, 6.23it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:28<00:00, 6.23it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:27<00:00, 6.25it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:54<00:00, 6.24it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:27<00:00, 6.25it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:28<00:00, 6.24it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:47<00:00, 6.47it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:20<00:00, 6.48it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:21<00:00, 6.45it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 5
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:54<00:00, 6.22it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:28<00:00, 6.22it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:27<00:00, 6.27it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:54<00:00, 6.22it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:28<00:00, 6.24it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:27<00:00, 6.27it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:47<00:00, 6.51it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:21<00:00, 6.46it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:20<00:00, 6.48it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 10
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:55<00:00, 6.19it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:28<00:00, 6.24it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:29<00:00, 6.21it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:54<00:00, 6.24it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:27<00:00, 6.27it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:29<00:00, 6.20it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████| 1087/1087 [02:47<00:00, 6.48it/s]
Predicting kNN: 100%|██████████| 1299/1299 [03:21<00:00, 6.44it/s]
Predicting kNN: 100%|██████████| 1300/1300 [03:21<00:00, 6.45it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
```

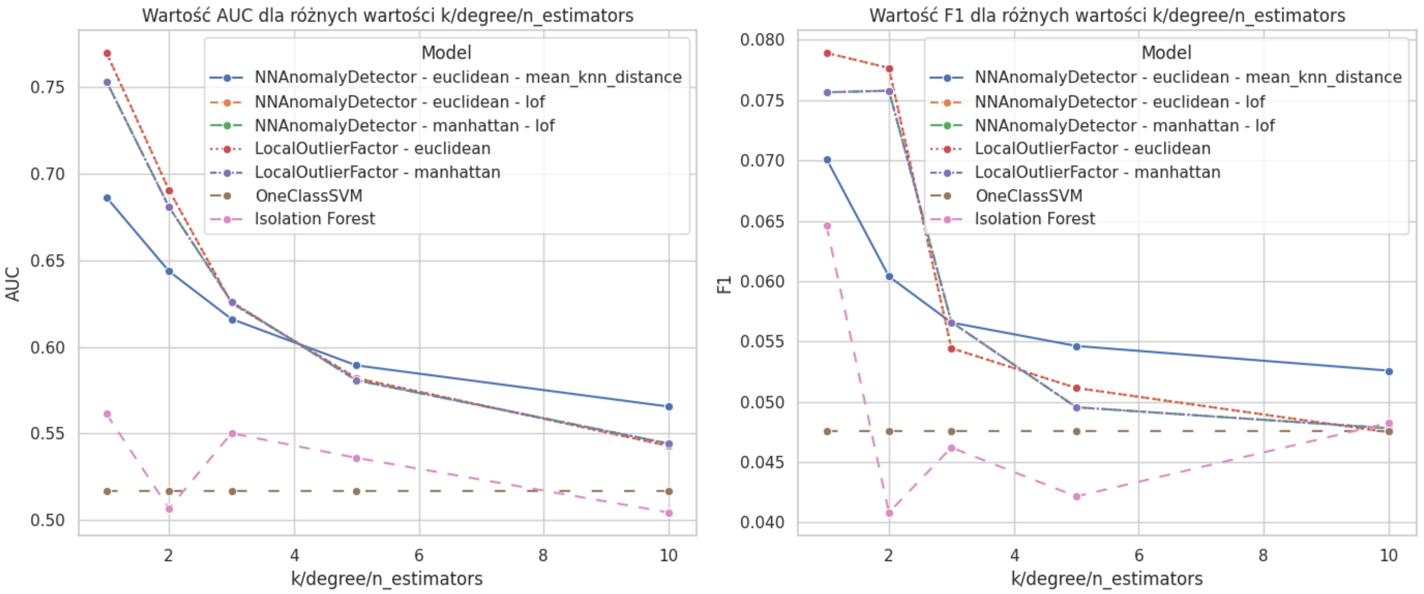
Out[ ]:

	dataset	param	model	AUC	F1	time
0	speech	1	NNAnomalyDetector - euclidean - mean_knn_distance	0.686207	0.070126	595.676975
1	speech	1	NNAnomalyDetector - euclidean - lof	0.769570	0.078915	592.777614
2	speech	1	NNAnomalyDetector - manhattan - lof	0.753067	0.075650	568.144813
3	speech	1	LocalOutlierFactor - euclidean	0.769546	0.078915	0.149556
4	speech	1	LocalOutlierFactor - manhattan	0.753067	0.075650	2.028319
5	speech	1	OneClassSVM	0.516921	0.047572	0.618802
6	speech	1	Isolation Forest	0.561283	0.064593	0.009611
7	speech	2	NNAnomalyDetector - euclidean - mean_knn_distance	0.643634	0.060386	592.306440
8	speech	2	NNAnomalyDetector - euclidean - lof	0.690488	0.077670	591.243268

	dataset	param	model	AUC	F1	time
9	speech	2	NNAnomalyDetector - manhattan - lof	0.680706	0.075783	568.620324
10	speech	2	LocalOutlierFactor - euclidean	0.690488	0.077670	0.128151
11	speech	2	LocalOutlierFactor - manhattan	0.680706	0.075783	2.053143
12	speech	2	OneClassSVM	0.516921	0.047572	0.903501
13	speech	2	Isolation Forest	0.506754	0.040821	0.020642
14	speech	3	NNAnomalyDetector - euclidean - mean_knn_distance	0.616082	0.056561	590.786069
15	speech	3	NNAnomalyDetector - euclidean - lof	0.625218	0.054435	590.596291
16	speech	3	NNAnomalyDetector - manhattan - lof	0.625984	0.056604	570.278046
17	speech	3	LocalOutlierFactor - euclidean	0.625218	0.054435	0.128197
18	speech	3	LocalOutlierFactor - manhattan	0.625984	0.056604	2.033033
19	speech	3	OneClassSVM	0.516921	0.047572	0.619271
20	speech	3	Isolation Forest	0.550152	0.046203	0.019140
21	speech	5	NNAnomalyDetector - euclidean - mean_knn_distance	0.589319	0.054622	590.830071
22	speech	5	NNAnomalyDetector - euclidean - lof	0.582000	0.051142	590.573794
23	speech	5	NNAnomalyDetector - manhattan - lof	0.580756	0.049541	569.158582
24	speech	5	LocalOutlierFactor - euclidean	0.582000	0.051142	0.132387
25	speech	5	LocalOutlierFactor - manhattan	0.580756	0.049541	2.899565
26	speech	5	OneClassSVM	0.516921	0.047572	0.973701
27	speech	5	Isolation Forest	0.535979	0.042146	0.046079
28	speech	10	NNAnomalyDetector - euclidean - mean_knn_distance	0.565569	0.052578	593.150100
29	speech	10	NNAnomalyDetector - euclidean - lof	0.542895	0.047473	591.550287
30	speech	10	NNAnomalyDetector - manhattan - lof	0.544211	0.047801	571.178561
31	speech	10	LocalOutlierFactor - euclidean	0.542895	0.047473	0.269682
32	speech	10	LocalOutlierFactor - manhattan	0.544211	0.047801	2.672717
33	speech	10	OneClassSVM	0.516921	0.047572	0.608947
34	speech	10	Isolation Forest	0.504484	0.048235	0.057015

In [ ]:

```
make_plot(speech_results_df)
```



Satellite

In [ ]:

```python
# Wartości k do przetestowania
k_values = [1, 2, 3, 5, 10]

# Przygotowanie danych
!wget -O satellite.mat "https://www.dropbox.com/s/dpzxp8jyr9h93k5/satellite.mat?dl=1"
data = prepare_data('./satellite.mat', 0.35, 0.35)

# Przeprowadzenie eksperymentów
results = evaluate_models(data, k_values, 'satellite')

# Konwersja wyników na DataFrame i wyświetlenie
satellite_results_df = pd.DataFrame(results)
satellite_results_df
```

```
--2024-06-08 22:03:24--  https://www.dropbox.com/s/dpzxp8jyr9h93k5/satellite.mat?dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.4.18, 2620:100:601c:18::a27d:612
Connecting to www.dropbox.com (www.dropbox.com)|162.125.4.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /scl/fi/vmty1xcfhk2bnaz/satellite.mat?rlkey=13tlpynr63wmcpk323pvb1o40&dl=1 [fol
lowing]
--2024-06-08 22:03:24--  https://www.dropbox.com/scl/fi/vmty1xcfhk2bnaz/satellite.mat?rlk
ey=13tlpynr63wmcpk323pvb1o40&dl=1
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucee6a0376c2fb60077516efe48b.dl.dropboxusercontent.com/cd/0/inline/CUdy
IOqC0pwIje06JN8PHy3h9lOe6L0ZVmFaIQjm9broF3s0h1nvV_W1wzsfps4C7snDvct6KbAR8bNIbkaIfVpQmMAqd
il8JF0CG8nAKfoIlAJST1K8dzEYvuDZPiujWTs/file?dl=1# [following]
--2024-06-08 22:03:24--  https://ucee6a0376c2fb60077516efe48b.dl.dropboxusercontent.com/c
d/0/inline/CUdyIOqC0pwIje06JN8PHy3h9lOe6L0ZVmFaIQjm9broF3s0h1nvV_W1wzsfps4C7snDvct6KbAR8b
NIbkaIfVpQmMAqdil8JF0CG8nAKfoIlAJST1K8dzEYvuDZPiujWTs/file?dl=1
Resolving ucee6a0376c2fb60077516efe48b.dl.dropboxusercontent.com (ucee6a0376c2fb60077516e
fe48b.dl.dropboxusercontent.com)... 162.125.4.15, 2620:100:6019:15::a27d:40f
Connecting to ucee6a0376c2fb60077516efe48b.dl.dropboxusercontent.com (ucee6a0376c2fb60077
516efe48b.dl.dropboxusercontent.com)|162.125.4.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/CUc8I3zoIT_cQxjSXgr0DThI2s5wp7pk5h73jsF34Vf3wZeVxounrPjGsqqAqTyAs
qHfMguDO0RbePQYrYqBBYhU_6p2-V9kujNj5KNjm7EK6bRd8-qfIwwmFBSRpZn9TzZSEDm4jHfUF5McB3lCHXduh0
-qdK4Aiy5dav7QvMrLvRaul2u9e_nObTDxO8Xrv38JnSCalU2yfwA24yAb3CXghQAMYEUeiST0-Jpojb4EZqj7cED
axv-vPXVv0kFcrzt8eYp07Z78pObMhEBxY9xl2uzlapYqq9mJTFK4LMcyCYmeeSGqtGkQ1bVlQ-5yUp_iW-o_BZaV
7tpKyh3pl_AlUhO-4ozmnqrCffENxjcroQ/file?dl=1 [following]
--2024-06-08 22:03:25--  https://ucee6a0376c2fb60077516efe48b.dl.dropboxusercontent.com/c
d/0/inline2/CUc8I3zoIT_cQxjSXgr0DThI2s5wp7pk5h73jsF34Vf3wZeVxounrPjGsqqAqTyAsqHfMguDO0Rbe
PQYrYqBBYhU_6p2-V9kujNj5KNjm7EK6bRd8-qfIwwmFBSRpZn9TzZSEDm4jHfUF5McB3lCHXduh0-qdK4Aiy5dav
7QvMrLvRaul2u9e_nObTDxO8Xrv38JnSCalU2yfwA24yAb3CXghQAMYEUeiST0-Jpojb4EZqj7cEDaxv-vPXVv0kF
crzt8eYp07Z78pObMhEBxY9xl2uzlapYqq9mJTFK4LMcyCYmeeSGqtGkQ1bVlQ-5yUp_iW-o_BZaV7tpKyh3pl_Al
UhO-4ozmnqrCffENxjcroQ/file?dl=1
Reusing existing connection to ucee6a0376c2fb60077516efe48b.dl.dropboxusercontent.com:443
.
HTTP request sent, awaiting response... 200 OK
Length: 144833 (141K) [application/binary]
Saving to: ‘satellite.mat’

satellite.mat       100%[===================>] 141.44K  --.-KB/s    in 0.02s

2024-06-08 22:03:25 (6.60 MB/s) - ‘satellite.mat’ saved [144833/144833]

param: 1
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████████████████████████████████████████| 1319/1319 [04:19<00:00,
5.08it/s]
Predicting kNN: 100%|███████████████████████████████████████████| 2558/2558 [08:11<00:00,
5.20it/s]
Predicting kNN: 100%|███████████████████████████████████████████| 2558/2558 [08:10<00:00,
5.21it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████████████████████████████████████████| 1319/1319 [04:10<00:00,
5.26it/s]
```

```
Predicting kNN: 100%|████████████████████| 2558/2558 [08:06<00:00, 5.26it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:21<00:00, 5.10it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|█████████████████████| 1319/1319 [04:07<00:00, 5.32it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [07:56<00:00, 5.37it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [07:57<00:00, 5.35it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 2
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|█████████████████████| 1319/1319 [04:13<00:00, 5.21it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:10<00:00, 5.22it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:09<00:00, 5.23it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|█████████████████████| 1319/1319 [04:13<00:00, 5.20it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:09<00:00, 5.22it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:06<00:00, 5.26it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|█████████████████████| 1319/1319 [03:58<00:00, 5.53it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [07:44<00:00, 5.51it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [07:59<00:00, 5.34it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 3
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|█████████████████████| 1319/1319 [04:14<00:00, 5.19it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:07<00:00, 5.25it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:07<00:00, 5.25it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|█████████████████████| 1319/1319 [04:16<00:00, 5.13it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:03<00:00, 5.29it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [08:03<00:00, 5.29it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|█████████████████████| 1319/1319 [04:03<00:00, 5.43it/s]
Predicting kNN: 100%|████████████████████| 2558/2558 [07:45<00:00,
```

```
5.49it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [07:43<00:00, 5.52it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 5
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|███████████████████████████████████████████| 1319/1319 [04:04<00:00, 5.40it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [07:58<00:00, 5.34it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:02<00:00, 5.30it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|███████████████████████████████████████████| 1319/1319 [04:12<00:00, 5.22it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:08<00:00, 5.23it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:09<00:00, 5.23it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|███████████████████████████████████████████| 1319/1319 [04:07<00:00, 5.33it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [07:56<00:00, 5.37it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:01<00:00, 5.32it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 10
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|███████████████████████████████████████████| 1319/1319 [04:16<00:00, 5.14it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:11<00:00, 5.21it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:10<00:00, 5.22it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|███████████████████████████████████████████| 1319/1319 [04:15<00:00, 5.17it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:08<00:00, 5.23it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [08:11<00:00, 5.21it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|███████████████████████████████████████████| 1319/1319 [04:02<00:00, 5.45it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [07:51<00:00, 5.43it/s]
Predicting kNN: 100%|████████████████████████████████████████| 2558/2558 [07:50<00:00, 5.43it/s]
```

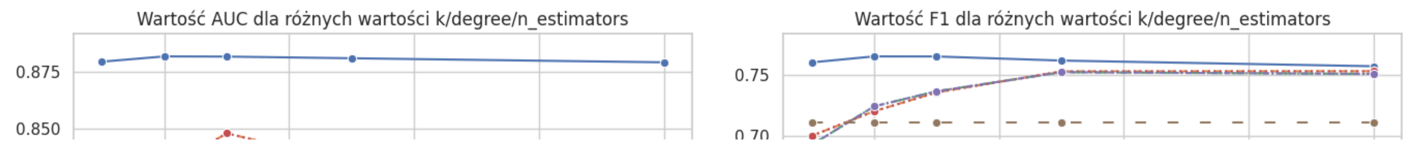
```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
```

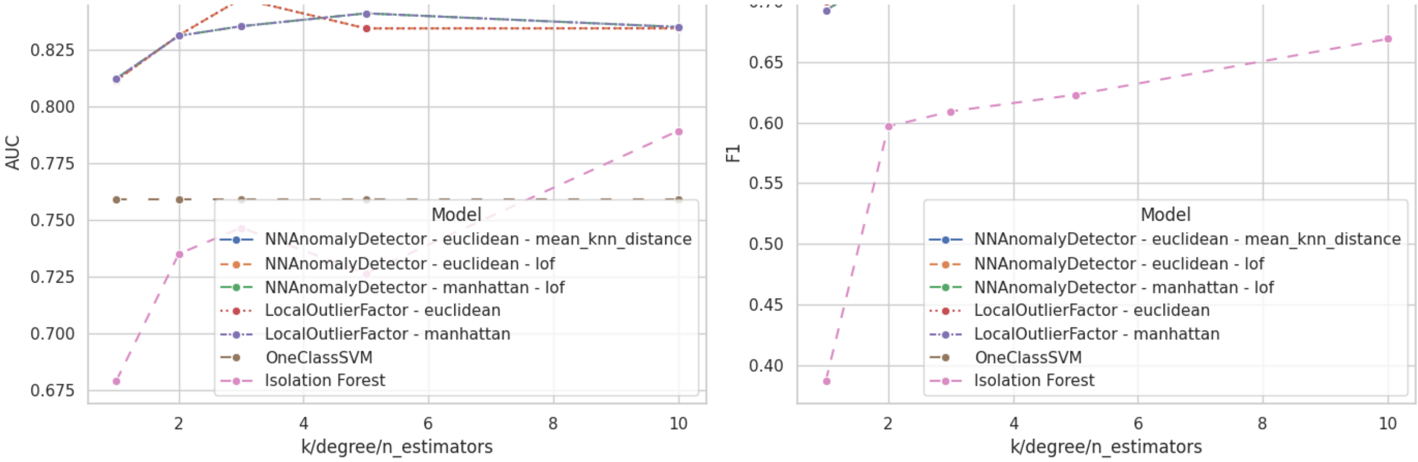
Out[ ]:

	dataset	param	model	AUC	F1	time
0	satellite	1	NNAnomalyDetector - euclidean - mean_knn_distance	0.879353	0.760624	1242.003535
1	satellite	1	NNAnomalyDetector - euclidean - lof	0.811639	0.700115	1239.515458
2	satellite	1	NNAnomalyDetector - manhattan - lof	0.812507	0.692683	1201.992401
3	satellite	1	LocalOutlierFactor - euclidean	0.811803	0.700115	0.106527
4	satellite	1	LocalOutlierFactor - manhattan	0.812507	0.692683	0.405690
5	satellite	1	OneClassSVM	0.759192	0.711217	0.308807
6	satellite	1	Isolation Forest	0.679005	0.387228	0.006749
7	satellite	2	NNAnomalyDetector - euclidean - mean_knn_distance	0.881674	0.765556	1232.873002
8	satellite	2	NNAnomalyDetector - euclidean - lof	0.831590	0.720418	1229.717489
9	satellite	2	NNAnomalyDetector - manhattan - lof	0.831371	0.724419	1181.887689
10	satellite	2	LocalOutlierFactor - euclidean	0.831595	0.720418	0.063335
11	satellite	2	LocalOutlierFactor - manhattan	0.831371	0.724419	0.401762
12	satellite	2	OneClassSVM	0.759192	0.711217	0.305112
13	satellite	2	Isolation Forest	0.735019	0.596817	0.012183
14	satellite	3	NNAnomalyDetector - euclidean - mean_knn_distance	0.881595	0.765502	1228.886257
15	satellite	3	NNAnomalyDetector - euclidean - lof	0.847754	0.735894	1224.627598
16	satellite	3	NNAnomalyDetector - manhattan - lof	0.835596	0.736842	1172.685835
17	satellite	3	LocalOutlierFactor - euclidean	0.847754	0.735894	0.384337
18	satellite	3	LocalOutlierFactor - manhattan	0.835595	0.736842	0.414232
19	satellite	3	OneClassSVM	0.759192	0.711217	0.318665
20	satellite	3	Isolation Forest	0.746528	0.609266	0.016810
21	satellite	5	NNAnomalyDetector - euclidean - mean_knn_distance	0.880860	0.762122	1206.256786
22	satellite	5	NNAnomalyDetector - euclidean - lof	0.834632	0.753201	1230.895240
23	satellite	5	NNAnomalyDetector - manhattan - lof	0.841289	0.752566	1205.369629
24	satellite	5	LocalOutlierFactor - euclidean	0.834631	0.753201	0.066253
25	satellite	5	LocalOutlierFactor - manhattan	0.841289	0.752566	0.420472
26	satellite	5	OneClassSVM	0.759192	0.711217	0.356856
27	satellite	5	Isolation Forest	0.726416	0.623040	0.038945
28	satellite	10	NNAnomalyDetector - euclidean - mean_knn_distance	0.879064	0.757456	1238.241774
29	satellite	10	NNAnomalyDetector - euclidean - lof	0.834674	0.753368	1236.456679
30	satellite	10	NNAnomalyDetector - manhattan - lof	0.835269	0.751152	1184.866279
31	satellite	10	LocalOutlierFactor - euclidean	0.834674	0.753368	0.067494
32	satellite	10	LocalOutlierFactor - manhattan	0.835269	0.751152	0.408858
33	satellite	10	OneClassSVM	0.759192	0.711217	0.293509
34	satellite	10	Isolation Forest	0.789351	0.669061	0.047572

In [ ]:

```
make_plot(satellite_results_df)
```





ForestCover

In [ ]:

```
# Wartości k do przetestowania
k_values = [1, 2, 3, 5, 10]

# Przygotowanie danych
!wget -O cover.mat "https://www.dropbox.com/s/awx8iuzbu8dkxf1/cover.mat?dl=1"
data = prepare_data('./cover.mat', 0.35, 0.35, sample_fraction=0.015)

#X_train, X_val, X_test, y_train, y_val, y_test = data

#X_train, X_val, X_test, y_train, y_val, y_test = X_train.sample(0.1), X_val.sample(0.1),
X_test.sample(0.1), y_train.sample(0.1), y_val.sample(0.1), y_test.sample(0.1)
#X_train = np.random.choice(X_train, size=0.1, replace=False)
#X_val = np.random.choice(X_train, size=0.1, replace=False)

#data = (X_train, X_val, X_test, y_train, y_val, y_test)

# Przeprowadzenie eksperymentów
results = evaluate_models(data, k_values, 'cover')

# Konwersja wyników na DataFrame i wyświetlenie
cover_results_df = pd.DataFrame(results)
cover_results_df
```

```
--2024-06-09 09:15:58--  https://www.dropbox.com/s/awx8iuzbu8dkxf1/cover.mat?dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.18, 2620:100:601d:18::a27d:512
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /scl/fi/qco62n6heb46bt1/cover.mat?rlkey=fxjp24ma19odscx26bshg84i9&dl=1 [followi
ng]
--2024-06-09 09:15:58--  https://www.dropbox.com/scl/fi/qco62n6heb46bt1/cover.mat?rlkey=f
xjp24ma19odscx26bshg84i9&dl=1
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucd5555b0ce766ed1a6c14974907.dl.dropboxusercontent.com/cd/0/inline/CUcL
8F5uDlOzoOOZgVG27tky7zC9P4YBZiOqY7LBFlLIHZXE8sVK_F9p8HOUNixke9IVn1XF2KXu1s126e37bToAQwicu
XV6yQC_iUzWWSnku93uLsIQKYCtKcDPFWubXTI/file?dl=1# [following]
--2024-06-09 09:15:58--  https://ucd5555b0ce766ed1a6c14974907.dl.dropboxusercontent.com/c
d/0/inline/CUcL8F5uDlOzoOOZgVG27tky7zC9P4YBZiOqY7LBFlLIHZXE8sVK_F9p8HOUNixke9IVn1XF2KXu1s
126e37bToAQwicuXV6yQC_iUzWWSnku93uLsIQKYCtKcDPFWubXTI/file?dl=1
Resolving ucd5555b0ce766ed1a6c14974907.dl.dropboxusercontent.com (ucd5555b0ce766ed1a6c149
74907.dl.dropboxusercontent.com)... 162.125.5.15, 2620:100:601f:15::a27d:90f
Connecting to ucd5555b0ce766ed1a6c14974907.dl.dropboxusercontent.com (ucd5555b0ce766ed1a6
c14974907.dl.dropboxusercontent.com)|162.125.5.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/CUc352G3qFyTY_PYSROPcN8yIMS3wsTH6DMe0A72A1hRgSCiFvNNWz2T81IPKlp2_
tq2_y9ZupdyGg1gaR5A7uZttXbjbPaoLqDBFthhEpkoQ6aU17XYT0QkCNHV3Bnysp-71-KqguisG3Ks--GAhzPXEj
HCNhBTyqhA-eWViKdvDHYdx2NVHglq_eVfX9_7qK8pB-_khi7xD3Ywd7Kj3wlC-7mxcfO2TB8vjn-geMyMXm2Xl5N
dKGyHTwYdaRlYvNigjgLiDR8roPGt-IRhAt7PvuIhBxypOOdll-NkTFfDEaXzuT3EbFentbSB3YwgydadqzhzB2ef
BCopvy7NYoeOupDTJjQBoZGbvFae0x35Lg/file?dl=1 [following]
--2024-06-09 09:15:59--  https://ucd5555b0ce766ed1a6c14974907.dl.dropboxusercontent.com/c
d/0/inline2/CUc352G3qFyTY_PYSROPcN8yIMS3wsTH6DMe0A72A1hRgSCiFvNNWz2T81IPKlp2_tq2_y9ZupdyG
```

```
g1gaR5A7uZttXbjbPaoLqDBFthhEpkoQ6aU17XYT0QkCNHV3Bnysp-71-KqguisG3Ks--GAhzPXEjHCNhBTyqhA-e
WViKdvDHYdx2NVHglq_eVfX9_7qK8pB-_khi7xD3Ywd7Kj3wlC-7mxcfO2TB8vjn-geMyMXm2Xl5NdKGyHTwYdaRl
YvNigjgLiDR8roPGt-IRhAt7PvuIhBxypOOdll-NkTFfDEaXzuT3EbFentbSB3YwgydadqzhzB2efBCopvy7NYoeO
upDTJjQBoZGbvFae0x35Lg/file?dl=1
Reusing existing connection to ucd5555b0ce766ed1a6c14974907.dl.dropboxusercontent.com:443
.
HTTP request sent, awaiting response... 200 OK
Length: 3089443 (2.9M) [application/binary]
Saving to: ‘cover.mat’

cover.mat           100%[===================>]   2.95M  --.-KB/s    in 0.1s

2024-06-09 09:15:59 (20.7 MB/s) - ‘cover.mat’ saved [3089443/3089443]

param: 1
Model: NNAnomalyDetector - euclidean - mean_knn_distance
Fitting kNN: 100%|█████████████████████████████████████████████| 1274/1274 [04:02<00:00,
5.26it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:44<00:00,
5.31it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:42<00:00,
5.33it/s]
Model: NNAnomalyDetector - euclidean - lof
Fitting kNN: 100%|█████████████████████████████████████████████| 1274/1274 [03:58<00:00,
5.34it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:42<00:00,
5.34it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:35<00:00,
5.47it/s]
Model: NNAnomalyDetector - manhattan - lof
Fitting kNN: 100%|█████████████████████████████████████████████| 1274/1274 [03:55<00:00,
5.41it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:43<00:00,
5.32it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:39<00:00,
5.40it/s]
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 2
Model: NNAnomalyDetector - euclidean - mean_knn_distance
Fitting kNN: 100%|█████████████████████████████████████████████| 1274/1274 [03:59<00:00,
5.32it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:44<00:00,
5.30it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:42<00:00,
5.33it/s]
Model: NNAnomalyDetector - euclidean - lof
Fitting kNN: 100%|█████████████████████████████████████████████| 1274/1274 [03:55<00:00,
5.40it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:42<00:00,
5.34it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:41<00:00,
5.35it/s]
Model: NNAnomalyDetector - manhattan - lof
Fitting kNN: 100%|█████████████████████████████████████████████| 1274/1274 [03:53<00:00,
5.46it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:40<00:00,
5.38it/s]
Predicting kNN: 100%|██████████████████████████████████████████| 1508/1508 [04:37<00:00,
5.44it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 3
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████████████████████████| 1274/1274 [04:04<00:00,
5.22it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:42<00:00,
5.34it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:42<00:00,
5.35it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████████████████████████| 1274/1274 [03:55<00:00,
5.41it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:43<00:00,
5.32it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:44<00:00,
5.30it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████████████████████████| 1274/1274 [03:54<00:00,
5.43it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:34<00:00,
5.50it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:36<00:00,
5.45it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 5
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████████████████████████| 1274/1274 [04:07<00:00,
5.15it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:42<00:00,
5.33it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:37<00:00,
5.44it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████████████████████████| 1274/1274 [03:56<00:00,
5.39it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:39<00:00,
5.39it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:41<00:00,
5.36it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████████████████████████| 1274/1274 [03:50<00:00,
5.54it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:32<00:00,
5.53it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:35<00:00,
5.46it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
param: 10
Model: NNAnomalyDetector - euclidean - mean_knn_distance
```

```
Fitting kNN: 100%|██████████████████████████████| 1274/1274 [03:56<00:00,
5.39it/s]
Predicting kNN: 100%|████████████████████████████| 1508/1508 [04:45<00:00,
5.29it/s]
```

```
Predicting kNN: 100%|██████████| 1508/1508 [04:46<00:00,
5.27it/s]
```

```
Model: NNAnomalyDetector - euclidean - lof
```

```
Fitting kNN: 100%|██████████| 1274/1274 [03:57<00:00,
5.37it/s]
Predicting kNN: 100%|██████████| 1508/1508 [04:45<00:00,
5.29it/s]
Predicting kNN: 100%|██████████| 1508/1508 [04:40<00:00,
5.37it/s]
```

```
Model: NNAnomalyDetector - manhattan - lof
```

```
Fitting kNN: 100%|██████████| 1274/1274 [03:51<00:00,
5.50it/s]
Predicting kNN: 100%|██████████| 1508/1508 [04:34<00:00,
5.49it/s]
Predicting kNN: 100%|██████████| 1508/1508 [04:36<00:00,
5.46it/s]
```

```
Model: LocalOutlierFactor - euclidean
Model: LocalOutlierFactor - manhattan
Model: OneClassSVM
Model: IsolationForest
```

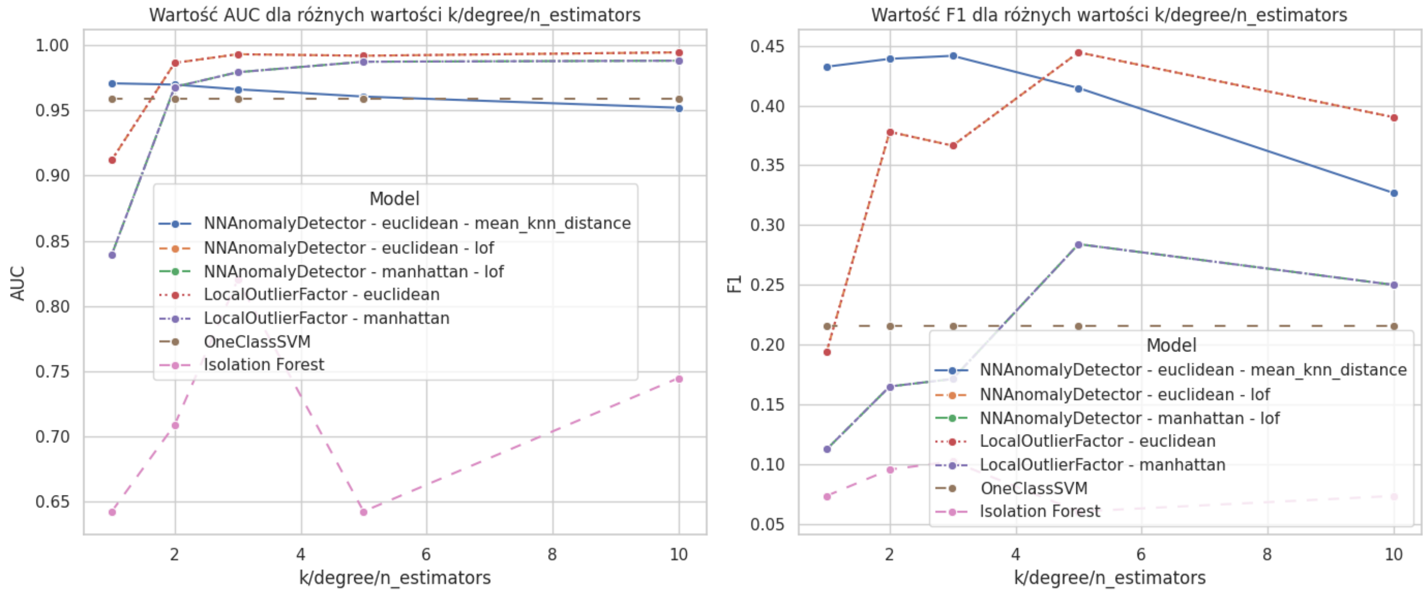
Out[ ]:

	dataset	param	model	AUC	F1	time
0	cover	1	NNAnomalyDetector - euclidean - mean_knn_distance	0.970735	0.432432	809.397963
1	cover	1	NNAnomalyDetector - euclidean - lof	0.911989	0.194286	796.350460
2	cover	1	NNAnomalyDetector - manhattan - lof	0.839393	0.112426	798.094907
3	cover	1	LocalOutlierFactor - euclidean	0.911989	0.194286	0.099850
4	cover	1	LocalOutlierFactor - manhattan	0.839393	0.112426	0.172937
5	cover	1	OneClassSVM	0.958543	0.215247	0.368464
6	cover	1	Isolation Forest	0.642541	0.073505	0.009137
7	cover	2	NNAnomalyDetector - euclidean - mean_knn_distance	0.969764	0.439024	806.692149
8	cover	2	NNAnomalyDetector - euclidean - lof	0.986379	0.377953	800.210507
9	cover	2	NNAnomalyDetector - manhattan - lof	0.968011	0.164948	790.830117
10	cover	2	LocalOutlierFactor - euclidean	0.986379	0.377953	0.114563
11	cover	2	LocalOutlierFactor - manhattan	0.968011	0.164948	0.213699
12	cover	2	OneClassSVM	0.958543	0.215247	0.316003
13	cover	2	Isolation Forest	0.709025	0.095477	0.014732
14	cover	3	NNAnomalyDetector - euclidean - mean_knn_distance	0.966150	0.441558	808.627168
15	cover	3	NNAnomalyDetector - euclidean - lof	0.993041	0.366412	803.787875
16	cover	3	NNAnomalyDetector - manhattan - lof	0.979231	0.171233	785.965734
17	cover	3	LocalOutlierFactor - euclidean	0.993041	0.366412	0.128719
18	cover	3	LocalOutlierFactor - manhattan	0.979231	0.171233	0.232738
19	cover	3	OneClassSVM	0.958543	0.215247	0.321926
20	cover	3	Isolation Forest	0.820183	0.102309	0.024041
21	cover	5	NNAnomalyDetector - euclidean - mean_knn_distance	0.960593	0.414634	807.139263
22	cover	5	NNAnomalyDetector - euclidean - lof	0.991881	0.444444	797.715861
23	cover	5	NNAnomalyDetector - manhattan - lof	0.987296	0.284024	778.966452
24	cover	5	LocalOutlierFactor - euclidean	0.991881	0.444444	0.104657

	dataset	param	model	AUC	F1	time
24	cover	5	LocalOutlierFactor - euclidean	0.991881	0.444444	0.104657
25	cover	5	LocalOutlierFactor - manhattan	0.987296	0.284024	0.205212
26	cover	5	OneClassSVM	0.958543	0.215247	0.307101
27	cover	5	Isolation Forest	0.642619	0.060386	0.031282
28	cover	10	NNAnomalyDetector - euclidean - mean_knn_distance	0.952043	0.326923	807.987225
29	cover	10	NNAnomalyDetector - euclidean - lof	0.994471	0.390244	803.490796
30	cover	10	NNAnomalyDetector - manhattan - lof	0.988078	0.250000	782.654489
31	cover	10	LocalOutlierFactor - euclidean	0.994471	0.390244	0.108792
32	cover	10	LocalOutlierFactor - manhattan	0.988078	0.250000	0.183002
33	cover	10	OneClassSVM	0.958543	0.215247	0.155985
34	cover	10	Isolation Forest	0.744755	0.073388	0.039378

In [ ]:

```
make_plot(cover_results_df)
```



Wyniki i wnioski

1. **Czas wykonania zaprezentowanej implementacji algorytmu kNN do zadania wykrywania anomalii okazał się być znacznie dłuższy względem gotowych algorytmów. Wpływ na to ma fakt, że w naszej implementacji algorytmu *kNN* uwzględniamy sytuację, w której algorytm może zwrócić więcej niż k sąsiadów, jeżeli k+n-ty sąsiad jest tak samo odległy jak k-ty sąsiad. W związku z tym dla każdego przykładu algorytm może zwrócić inną długość listy, dlatego w algorytmie *NNAnomalyDetector* operujemy na listach np.ndarray zamiast po prostu na np.ndarray. Skutkuje to zwiększeniem czasu obliczeń. Najprawdopodobniej kolejnym czynnikiem wpływającym na czas jest to, że nie stosujemy technik zrównoleglania, mimo wykonywania wielu iteracji zbioru danych, co prawdopodobnie jest zoptymalizowane w implementacjach znajdujących się w bibliotekach.**
2. **Wartości wskaźników nieprawidłowości LOF dla naszego algorytmu *NNAnomalyDetector* są identyczne jak dla algorytmu *LocalOutlierFactor* z sklearn, co oznacza, że nasz algorytm został prawidłowo zaimplementowany.**
3. **Predykcja naszego algorytmu *NNAnomalyDetector* okazała się być w prawie wszystkich eksperymentach znacząco lepsza od pozostałych algorytmów klasyfikacji jednoklasowej, tj. *OneClassSVM* i *Isolation Forest*, których zasada działania różni się od *kNN*.**
4. **Najwyższe wartości wskazaźników uzyskano na zbiorze *Satellite*, co oznacza, że rozważany algorytm najlepiej sprawdził się w przypadku zbioru danych z największą liczbą anomalii. Z kolei niższe wartości badanych miar zmierzono na pozostałych zbiorach, które miały dużą wymiarowość (*Speech*), a ponadto małą liczbę przykładów odstających (*Speech* i *Forest Cover*). Obserwacje te sugerują, iż zaimplementowany algorytm najlepiej sprawdza się w przypadku zbalansowanego zbioru danych.**

5. **Przy zastosowaniu metryki *Manhattan* uzyskauje się gorsze lub tak samo dobre wyniki jak przy wykorzystaniu metryki *Euclidean*. Różnica w wartościach zależy od konkretnego zbioru i liczby sąsiadów. W związku z tym, że metryka *Manhattan* jest prostsza obliczeniowo, dla każdego zbioru danych zastosowanie jej skraca czas obliczeń. Oznacza to, że wykorzystanie jej może być korzystne dla niektórych zbiorów danych.**
6. **Jakość predykcji przy użyciu wskaźnika *mean_knn_distance*, który nie uwzględnia gęstości przykładów trenujących, jest znacząco inna niż przy użyciu wskaźnika *lof*, lecz nadal wysoka.**
7. **Optymalna liczba sąsiadów *k* powinna rosnąć wraz z rozmiarem zbioru danych uczących.**

In [ ]:

```

```