Détection d'anomalies routières à partir d'échanges de données entre systèmes de transports intelligents coopératifs

L'intégralité de ce programme et des documents relatifs au projet sont disponible à l'adresse : https://github.com/Antoine553/projet-master2/

In [1]:

```
import pandas as pd
import numpy as np
from numpy import percentile
import matplotlib
import matplotlib.pyplot as plt

from pyod.models.abod import ABOD
from pyod.models.cblof import CBLOF
from pyod.models.feature_bagging import FeatureBagging
from pyod.models.iforest import IForest
from pyod.models.lscp import LSCP
from pyod.models.mcd import MCD

from pysad.utils import ArrayStreamer
from pysad.models.integrations import ReferenceWindowModel
from pysad.transform.ensemble import *
from pysad.evaluation import *

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle

from scipy import stats
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
```

Jeu de donnée

Importation et enrichissement des données

In [2]:

```
### Chargement des données dans un dataframe
columns = ['Time', 'CarId', 'Longitude', 'Latitude', 'Speed', 'Heading', 'Class']
df = pd.read_csv('data/cam_1000.csv', usecols=columns)
### Rajout de nouvelles colonnes avec valeurs à zero
df['ID'] = 0
df['CarId'] = df['CarId'].astype(str)
df['Time diff'] = 0.0
df['Position diff'] = 0.0
df['Speed diff'] = 0.0
df['Heading diff'] = 0.0
df = df[['ID', 'Time', 'CarId', 'Longitude', 'Latitude', 'Speed', 'Heading', 'Time diff', 'Position diff', 'Speed diff', 'Heading diff', 'Class']]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6341 entries, 0 to 6340
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   ID             6341 non-null   int64  
 1   Time           6341 non-null   float64
 2   CarId          6341 non-null   object 
 3   Longitude      6341 non-null   float64
 4   Latitude       6341 non-null   float64
 5   Speed          6341 non-null   float64
 6   Heading        6341 non-null   float64
```

```
 7   Time diff       6341 non-null   float64
 8   Position diff   6341 non-null   float64
 9   Speed diff      6341 non-null   float64
 10  Heading diff    6341 non-null   float64
 11  Class           6341 non-null   int64
dtypes: float64(9), int64(2), object(1)
memory usage: 594.6+ KB
```

In [3]:

```
### Compare chaque donnée avec la précedente et calcule les variations
NId=0
for index, row in df.iterrows():
    df.at[index, 'ID'] = NId
    NId = NId+1
    if index != 0:
        if row[2] == prec_row[2] and (row[1] - prec_row[1]) < 5:  # Si la donnée n'est pas la premiè
re ou du même identifiant
            df.at[index, 'Time diff'] = abs(row[1] - prec_row[1])
            df.at[index, 'Position diff'] = (abs(row[3] - prec_row[3])+abs(row[4] - prec_row[4]))/(r
ow[1] - prec_row[1])
            df.at[index, 'Speed diff'] = abs(row[5] - prec_row[5])/(row[1] - prec_row[1]) # Differen
ce de vitesse
            df.at[index, 'Heading diff'] = abs(min((row[6]-prec_row[6])%360, (prec_row[6]-
row[6])%360))/(row[1] - prec_row[1]) # Difference de direction
        else:
            df.at[index, 'Time diff'] = 0.0
            df.at[index, 'Position diff'] = 0.0
            df.at[index, 'Speed diff'] = 0.0
            df.at[index, 'Heading diff'] = 0.0
    prec_row = row
```

Analyse statistique

In [4]:

```
df.head(5)
```

Out[4]:

	ID	Time	CarId	Longitude	Latitude	Speed	Heading	Time diff	Position diff	Speed diff	Heading diff	Class
0	0	594.182212	118457	49.261743	4.056850	8.43	264.4	0.0	0.00000	0.0	0.0	0
1	1	594.282212	118457	49.261740	4.056839	8.37	260.5	0.1	0.00014	0.6	39.0	0
2	2	594.382212	118457	49.261737	4.056829	8.37	256.6	0.1	0.00013	0.0	39.0	0
3	3	594.482212	118457	49.261733	4.056820	8.36	250.6	0.1	0.00013	0.1	60.0	0
4	4	594.582212	118457	49.261727	4.056813	8.44	241.8	0.1	0.00013	0.8	88.0	0

In [5]:

```
df.describe()
```

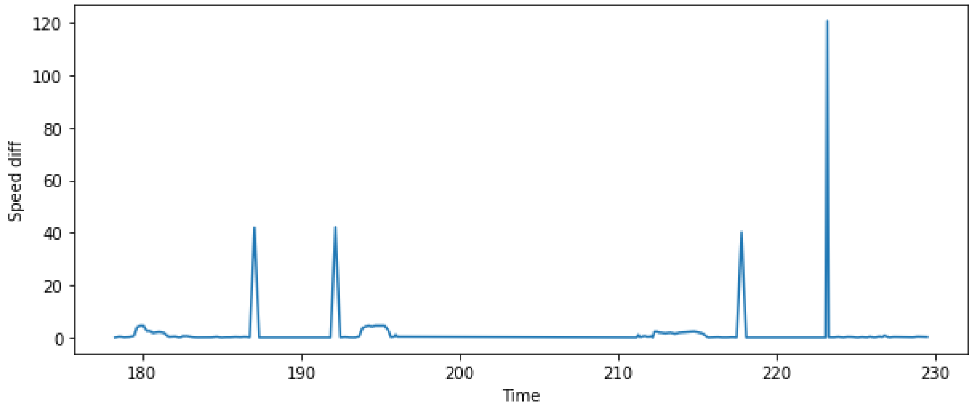
Out[5]:

	ID	Time	Longitude	Latitude	Speed	Heading	Time diff	Position diff	Speed diff	Heading
count	6341.00000	6341.000000	6341.000000	6341.000000	6341.000000	6341.000000	6341.000000	6341.000000	6341.000000	6341.000
mean	3170.00000	428.525575	49.260828	4.056626	9.786789	127.626652	0.250654	0.000130	1.648751	12.579
std	1830.63336	261.141013	0.000793	0.000153	4.959000	94.330976	0.120390	0.000047	8.614061	46.861
min	0.00000	30.960276	49.259629	4.056383	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	1585.00000	177.400747	49.260098	4.056486	5.280000	6.200000	0.200000	0.000113	0.000000	0.000
50%	3170.00000	457.684540	49.260696	4.056600	12.420000	175.200000	0.300000	0.000140	0.166667	0.000
75%	4755.00000	666.240830	49.261591	4.056769	13.930000	186.200000	0.300000	0.000153	0.800000	10.500
max	6340.00000	864.364589	49.262306	4.056999	16.220000	359.900000	1.000000	0.000500	139.700000	1752.000

Analyse des anomalies

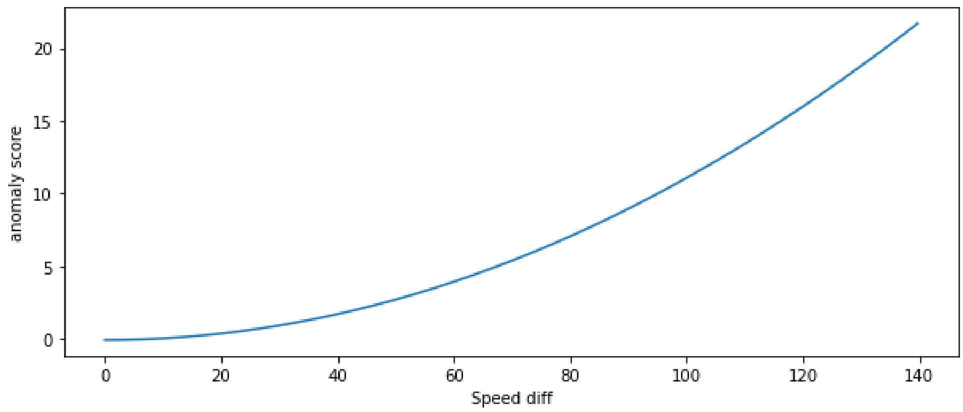
In [6]:

```
data195061 = df[(df['CarId'] == '195061')]
x = data195061['Time']
y = data195061['Speed diff']

plt.figure(figsize=(10,4))
plt.plot(x, y, label='Car 195061')
plt.xlabel('Time')
plt.ylabel('Speed diff')
plt.show();
```



In [7]:

```
lscp = LSCP(detector_list=[MCD(), MCD()])
lscp.fit(df['Speed diff'].values.reshape(-1, 1))
xx = np.linspace(df['Speed diff'].min(), df['Speed diff'].max(), len(df)).reshape(-1,1)
anomaly_score = lscp.decision_function(xx)
outlier = lscp.predict(xx)
plt.figure(figsize=(10,4))
plt.plot(xx, anomaly_score, label='anomaly score')
plt.ylabel('anomaly score')
plt.xlabel('Speed diff')
plt.show();
```



In [8]:

```
df.loc[df['Speed diff'] > 10]
```

Out[8]:

	ID	Time	CarId	Longitude	Latitude	Speed	Heading	Time diff	Position diff	Speed diff	Heading diff	Class
33	33	601.082212	118457	49.261013	4.056657	0.79	175.6	0.3	0.000160	46.366667	35.333333	1
54	54	605.682212	118457	49.260417	4.056479	14.76	185.6	0.1	0.000160	139.700000	100.000000	1

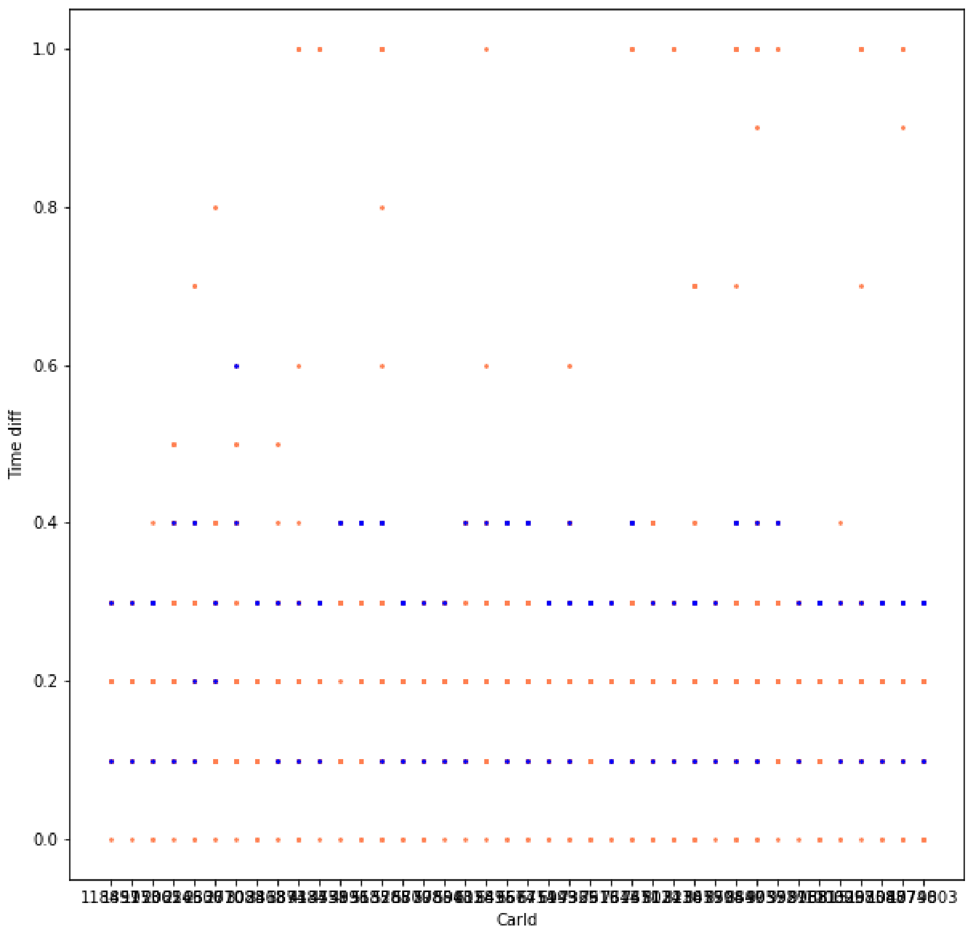
	ID	Time	CarId	Longitude	Latitude	Speed	Heading	Time diff	Position diff	Speed diff	Heading diff	Class
112	112	412.838044	189173	49.260709	4.056632	13.56	0.9	0.3	0.000153	31.466667	26.000000	1
170	170	423.138344	189173	49.260407	4.056478	13.91	185.6	0.1	0.000150	103.500000	1747.000000	1
232	232	187.062793	195061	49.260711	4.056633	1.18	0.2	0.3	0.000150	41.900000	26.333333	1
...	...	...	...	...	...	...	...	...	...	...	...	...
6016	6016	738.929830	1047740	49.260407	4.056478	15.41	185.6	0.1	0.000160	104.900000	96.000000	1
6083	6083	212.664817	1079803	49.260668	4.056622	4.20	0.5	0.3	0.000153	31.500000	25.333333	1
6101	6101	218.064817	1079803	49.261327	4.056760	13.76	6.2	0.3	0.000147	31.866667	19.000000	1
6178	6178	293.164817	1079803	49.261092	4.056672	1.50	174.2	0.3	0.000147	40.533333	40.000000	1
6203	6203	298.764817	1079803	49.260416	4.056479	13.71	185.6	0.1	0.000150	122.100000	114.000000	1

117 rows × 12 columns

Analyse graphique

In [9]:

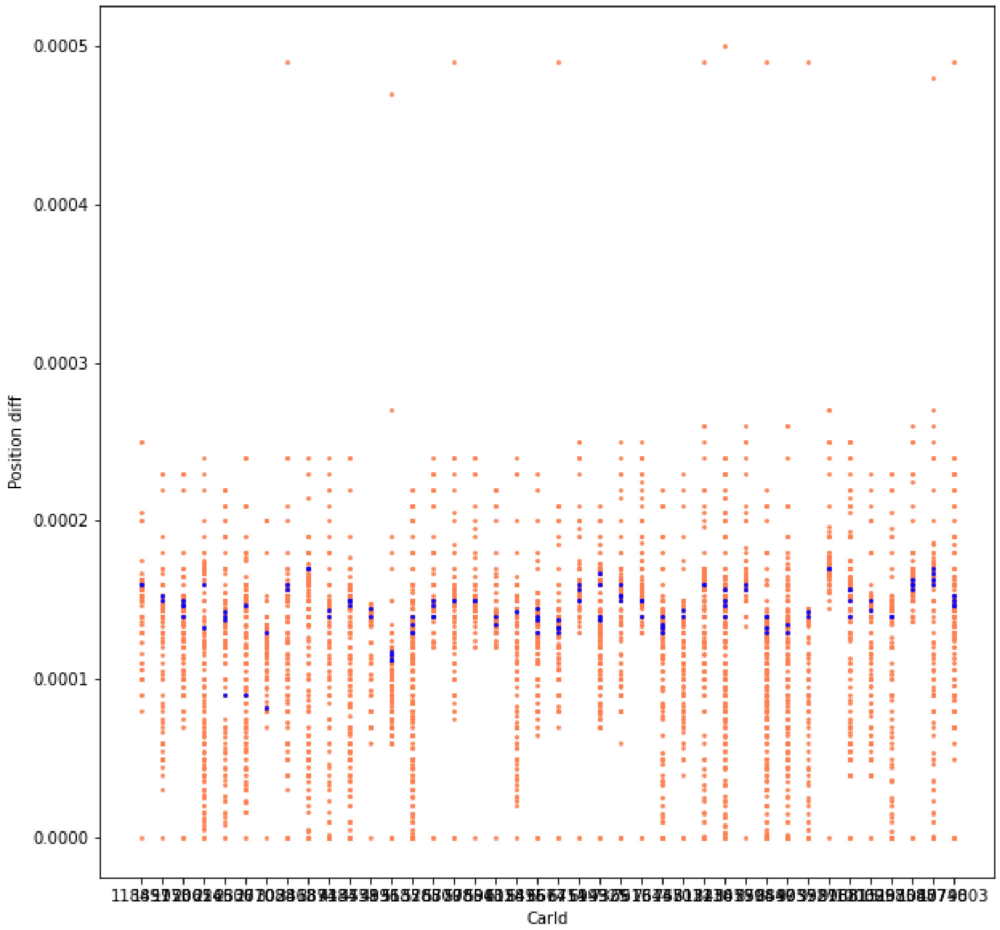
```
plt.figure(figsize=(10, 10))

plt.scatter(df[df['Class'] == 0]['CarId'],df[df['Class'] == 0]['Time diff'], s=3, c='coral')
plt.scatter(df[df['Class'] == 1]['CarId'],df[df['Class'] == 1]['Time diff'], s=3, c='blue')

plt.xlabel('CarId')
plt.ylabel('Time diff')
plt.show()
```



In [10]:

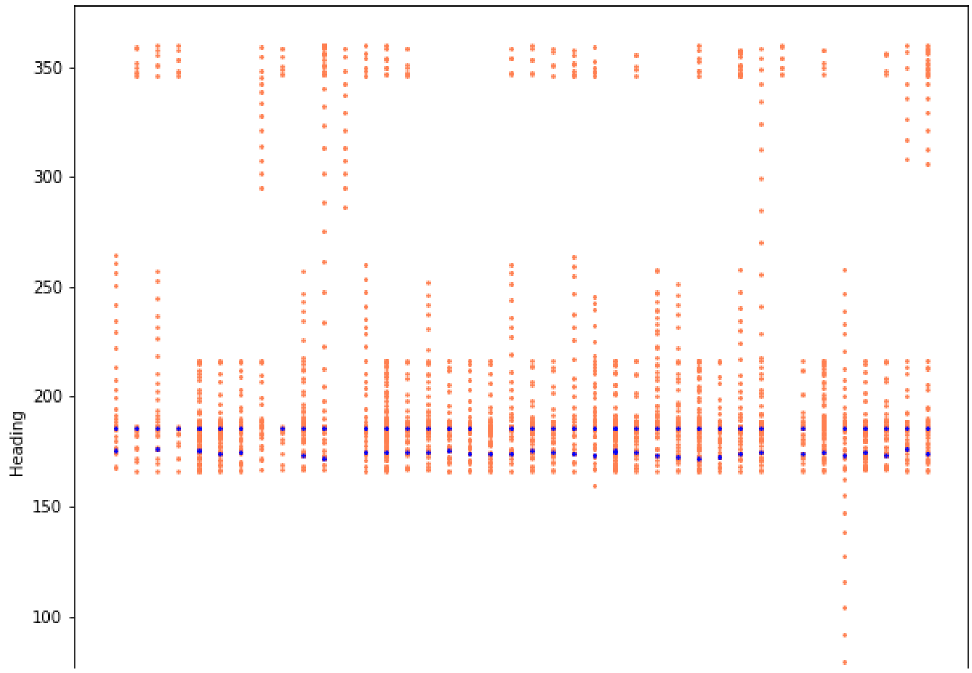
```
plt.figure(figsize=(10, 10))

plt.scatter(df[df['Class'] == 0]['CarId'],df[df['Class'] == 0]['Position diff'], s=3, c='coral')
plt.scatter(df[df['Class'] == 1]['CarId'],df[df['Class'] == 1]['Position diff'], s=3, c='blue')

plt.xlabel('CarId')
```

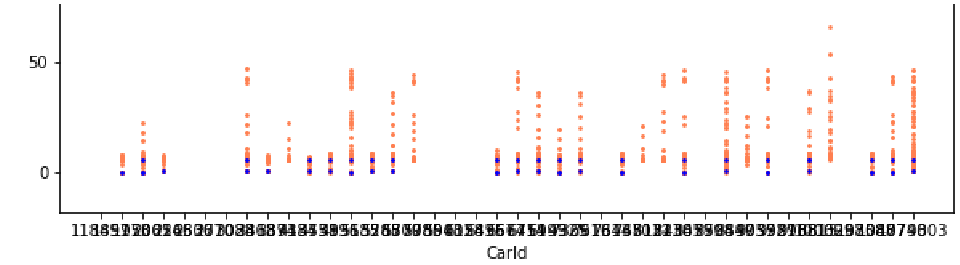
```
plt.xlabel('CarId')
plt.ylabel('Position diff')
plt.show()
```



In [11]:

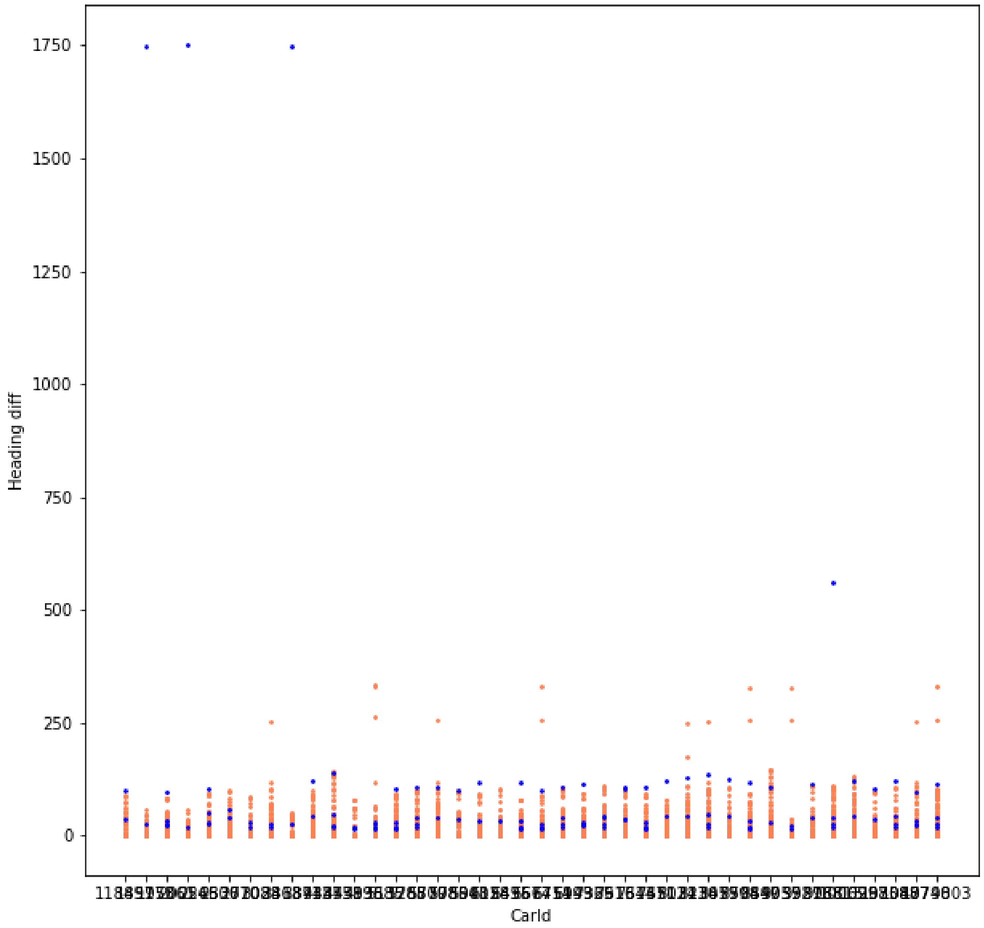
```
plt.figure(figsize=(10, 10))

plt.scatter(df[df['Class'] == 0]['CarId'],df[df['Class'] == 0]['Heading'], s=3, c='coral')
plt.scatter(df[df['Class'] == 1]['CarId'],df[df['Class'] == 1]['Heading'], s=3, c='blue')

plt.xlabel('CarId')
plt.ylabel('Heading')
plt.show()
```





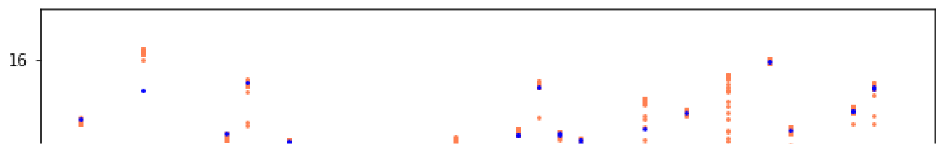
In [12]:

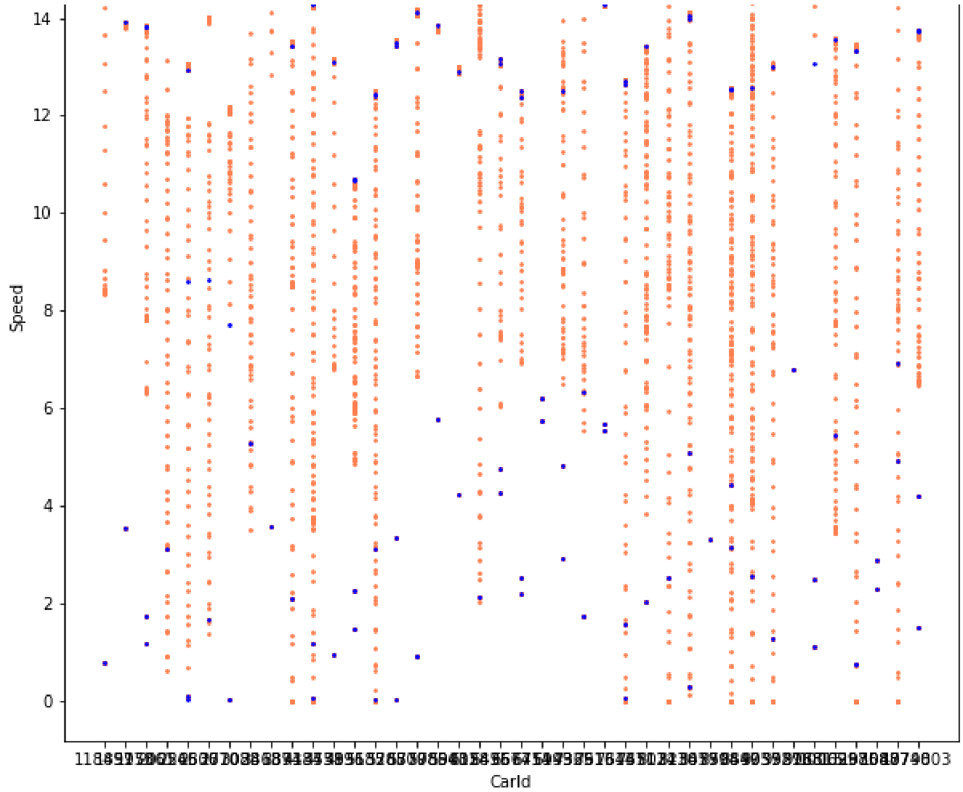
```
plt.figure(figsize=(10, 10))

plt.scatter(df[df['Class'] == 0]['CarId'],df[df['Class'] == 0]['Heading diff'], s=3, c='coral')
plt.scatter(df[df['Class'] == 1]['CarId'],df[df['Class'] == 1]['Heading diff'], s=3, c='blue')

plt.xlabel('CarId')
plt.ylabel('Heading diff')
plt.show()
```



In [13]:

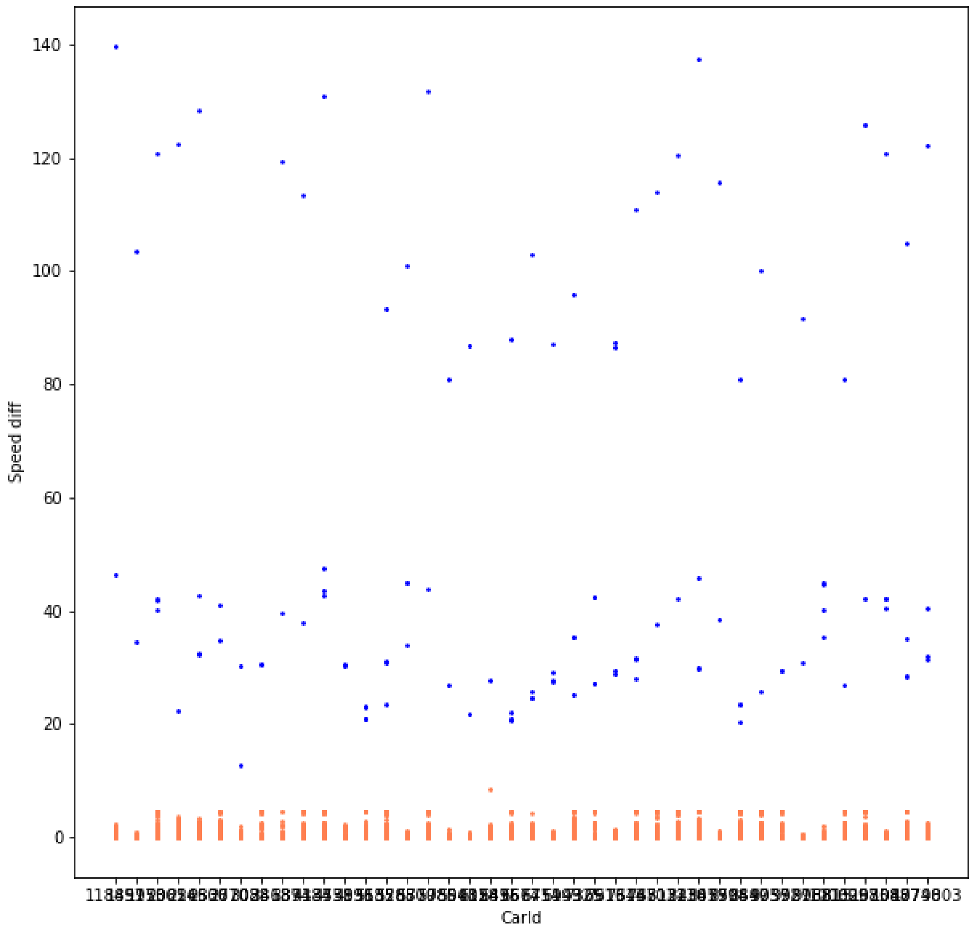
```
plt.figure(figsize=(10, 10))

plt.scatter(df[df['Class'] == 0]['CarId'],df[df['Class'] == 0]['Speed'], s=3, c='coral')
plt.scatter(df[df['Class'] == 1]['CarId'],df[df['Class'] == 1]['Speed'], s=3, c='blue')

plt.xlabel('CarId')
plt.ylabel('Speed')
plt.show()
```





In [14]:

```
plt.figure(figsize=(10, 10))

plt.scatter(df[df['Class'] == 0]['CarId'],df[df['Class'] == 0]['Speed diff'], s=3, c='coral')
plt.scatter(df[df['Class'] == 1]['CarId'],df[df['Class'] == 1]['Speed diff'], s=3, c='blue')

plt.xlabel('CarId')
plt.ylabel('Speed diff')
plt.show()
```



Standardisation des données

In [15]:

```
minmax = MinMaxScaler(feature_range=(0, 1))
df[['CarId', 'Speed diff', 'Heading diff', 'Position diff']] = minmax.fit_transform(df[['CarId', 'S
peed diff', 'Heading diff', 'Position diff']])
df[['CarId', 'Speed diff', 'Heading diff', 'Position diff']].head()
```

Out[15]:

	CarId	Speed diff	Heading diff	Position diff
0	0.0	0.000000	0.000000	0.00
1	0.0	0.004295	0.022260	0.28
2	0.0	0.000000	0.022260	0.26
3	0.0	0.000716	0.034247	0.26
4	0.0	0.005727	0.050228	0.26

In [16]:

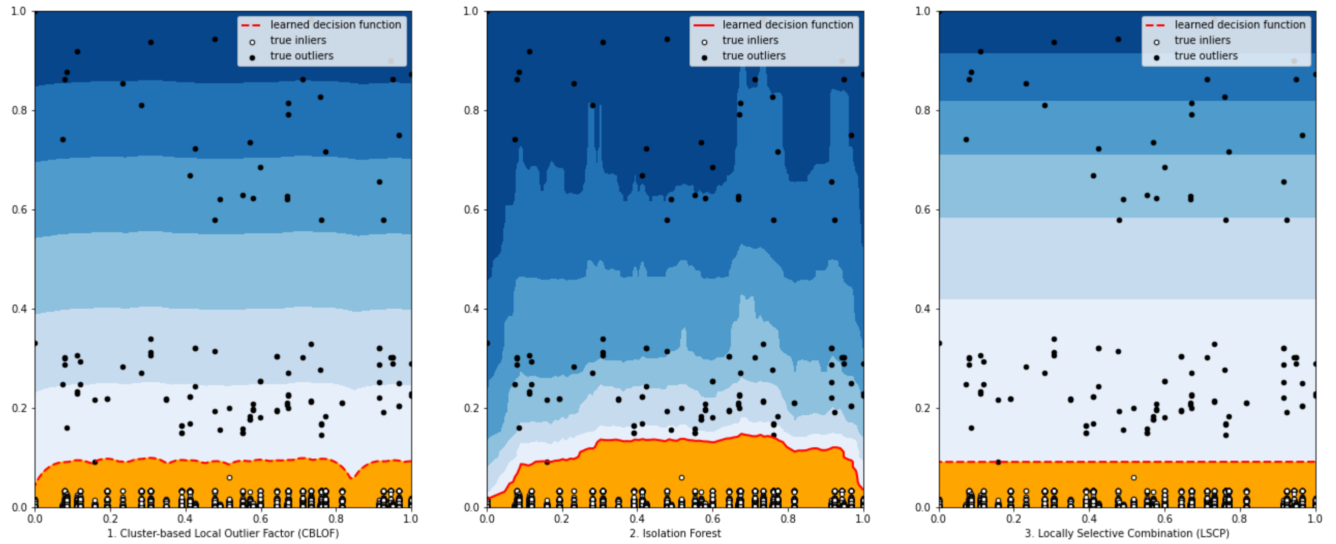
```
X1 = df['CarId'].values.reshape(-1,1)
X2 = df['Speed diff'].values.reshape(-1,1)
X3 = df['Heading diff'].values.reshape(-1,1)
X4 = df['Position diff'].values.reshape(-1,1)

X_speed = np.concatenate((X1,X2),axis=1)
X_heading = np.concatenate((X1,X3),axis=1)
X_position = np.concatenate((X1,X4),axis=1)
```

Tests LSCP, CBLOF et IForest sur notre jeu de données

In [17]:

```
outliers_fraction = 0.0183
xx, yy = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))

# Copie du dataframe
df1 = df
nb_id = df1['CarId'].nunique()+1
df1['outlier'] = df1['Class']

# Liste des algorithmes à tester
classifiers = {
    'Cluster-based Local Outlier Factor (CBLOF)':CBLOF(contamination=outliers_fraction,
check_estimator=False, random_state=0, n_clusters=nb_id),
    'Isolation Forest': IForest(contamination=outliers_fraction, random_state=0),
    'Locally Selective Combination (LSCP)': LSCP(detector_list=[MCD(),MCD()],
contamination=outliers_fraction, random_state=0),
}

# Variables contenant les données normale adaptées au graphique
inliers_CarId = np.array(df1['CarId'][df1['outlier'] == 0]).reshape(-1,1)
inliers_Speed_diff = np.array(df1['Speed diff'][df1['outlier'] == 0]).reshape(-1,1)

# Variables contenant les données anormale adaptées au graphique
outliers_CarId = df1['CarId'][df1['outlier'] == 1].values.reshape(-1,1)
outliers_Speed_diff = df1['Speed diff'][df1['outlier'] == 1].values.reshape(-1,1)


X = X_speed
plt.figure(figsize=(30, 30))
for i, (clf_name, clf) in enumerate(classifiers.items()):
    clf.fit(X)
    scores_pred = clf.decision_function(X) * -1
    y_pred = clf.predict(X)
    threshold = percentile(scores_pred, 100 * outliers_fraction)
```

```
    # Remplis la zone supérieur à la zone de décision en niveau de bleu selon le score d'anomalie
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
    Z = Z.reshape(xx.shape)
    subplot = plt.subplot(3, 4, i + 1)
    subplot.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7), cmap=plt.cm.Blues_r)

    # Dessine la ligne rouge de décision et colorie la zone inférieur en orange
    a = subplot.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors='red')
    subplot.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange')

    # Colorie les points en blanc ou noir selon la classification
    b = subplot.scatter(inliers_CarId, inliers_Speed_diff, c='white', s=20, edgecolor='k')
    c = subplot.scatter(outliers_CarId, outliers_Speed_diff, c='black', s=20, edgecolor='k')

    subplot.axis('tight')
    subplot.legend([a.collections[0], b, c],['learned decision function', 'true inliers', 'true out
liers'],prop=matplotlib.font_manager.FontProperties(size=10),loc='upper right')
    subplot.set_xlabel("%d. %s" % (i + 1, clf_name))
    subplot.set_xlim((0, 1))
    subplot.set_ylim((0, 1))
plt.show()
```



Tests LSCP, CBLOF et IForest avec données en streaming

In [69]:

```
nb_id = df['CarId'].nunique()    # nombre d'identifiants dans le jeu de donnée
# Copie du dataframe
df1 = df
df1['outlier'] = df1['Class']

X_all = pd.DataFrame(df1, columns=['CarId', 'Speed diff'])
X_all = X_all.to_numpy()
y_all = df1['Class'].to_numpy()

XS=np.size(X_all[:,1])
Y1=(y_all[:] == 1).sum()
Y2=(y_all[:] == 0).sum()

print("décompte des données :", XS)
print("donnée anormale :", Y1)
print("donnée normale :", Y2)

X_all, y_all = shuffle(X_all, y_all)  # Modification aléatoire de l'ordre des données
iterator = ArrayStreamer(shuffle=False)   # Simule l'arrivé des données en streaming

detector = [MCD(),MCD()] # Detecteur pour l'algorithme LSCP
list_models = [
    ReferenceWindowModel(model_cls=LSCP, window_size=200, sliding_size=40, initial_window_X=X_all[:
1000],detector_list=detector,),
    ReferenceWindowModel(model_cls=LSCP, window_size=1000, sliding_size=40, initial_window_X=X_all[
:1000],detector_list=detector,),
```

```
    ReferenceWindowModel(model_cls=CBLOF, window_size=200, sliding_size=40, initial_window_X=X_all[
:1000],n_clusters=nb_id,),
    ReferenceWindowModel(model_cls=CBLOF, window_size=1000, sliding_size=40, initial_window_X=X_all
[:1000],n_clusters=nb_id,),
    ReferenceWindowModel(model_cls=IForest, window_size=200, sliding_size=40, initial_window_X=X_al
l[:1000],),
    ReferenceWindowModel(model_cls=IForest, window_size=1000, sliding_size=40, initial_window_X=X_a
ll[:1000],)
]

ensembler = MedianScoreEnsembler()  # Combinaison des scores
for idx, model in enumerate(list_models):
    auroc = AUROCMetric()  # évaluation AUROC
    aupr = AUPRMetric()   # évaluation AUPR
    for X, y in tqdm(iterator.iter(X_all, y_all)):  # Iteration sur les données
        model_scores = np.empty(1, dtype=np.float)
        model.fit_partial(X)
        model_scores[i] = model.score_partial(X)
        score = ensembler.fit_transform_partial(model_scores)

        auroc.update(y, score)  # MAJ AUROC
        aupr.update(y, score)  # MAJ AUPR

    if idx == 0 :
        print("LSCP, Window_size=200")
    if idx == 1 :
        print("LSCP, Window_size=1000")
    if idx == 2 :
        print("CBLOF, Window_size=200")
    if idx == 3 :
        print("CBLOF, Window_size=1000")
    if idx == 4 :
        print("IForest, Window_size=200")
    if idx == 5 :
        print("IForest, Window_size=1000")
    print("AUROC: ", auroc.get())
    print("AUPR: ", aupr.get())
```

```
décompte des données : 6341
donnée anormale : 117
donnée normale : 6224
```

```
6341it [00:57, 109.88it/s]
20it [00:00, 99.77it/s]
```

```
LSCP, Window_size=200
AUROC:  0.9992776789049282
AUPR:  0.9584442949137293
```

```
6341it [04:04, 25.94it/s]
40it [00:00, 325.68it/s]
```

```
LSCP, Window_size=1000
AUROC:  1.0
AUPR:  0.9999999999999998
```

```
6341it [00:18, 351.64it/s]
40it [00:00, 271.93it/s]
```

```
CBLOF, Window_size=200
AUROC:  1.0
AUPR:  0.9999999999999998
```

```
6341it [00:21, 301.01it/s]
4it [00:00, 34.98it/s]
```

```
CBLOF, Window_size=1000
AUROC:  1.0
AUPR:  0.9999999999999998
```

```
6341it [03:34, 29.55it/s]
3it [00:00, 29.93it/s]
```

```
IForest, Window_size=200
AUROC:  0.9991266231626129
AUPR:  0.9705943368877611
```

```
6341it [03:38, 29.06it/s]
```

```
IForest, Window_size=1000
AUROC:  0.9999945070639158
AUPR:  0.9997132734312222
```

Système CBLOF sur jeu de données N°1

In [17]:

```
nb_id = df['CarId'].nunique()  # nombre d'identifiants dans le jeu de donnée
# Copie du dataframe
df1 = df
df1['outlier'] = df1['Class']

X_all = pd.DataFrame(df1, columns=['ID', 'CarId', 'Speed diff', 'Heading diff'])
X_all = X_all.to_numpy()
y_all = df1['Class'].to_numpy()

XS=np.size(X_all[:,0])
Y1=(y_all[:] == 1).sum()
Y2=(y_all[:] == 0).sum()

print("décompte des données :", XS)
print("donnée anormale :", Y1)
print("donnée normale :", Y2)

X_all, y_all = shuffle(X_all, y_all)  # Modification aléatoire de l'ordre des données
iterator = ArrayStreamer(shuffle=False)  # Simule l'arrivé des données en streaming

auroc = AUROCMetric()  # évaluation AUROC
aupr = AUPRMetric()  # évaluation AUPR

models = [ReferenceWindowModel(model_cls=CBLOF, window_size=1000, sliding_size=40, initial_window_X
=X_all[:1000][:,[1,2]],n_clusters=nb_id),
         ReferenceWindowModel(model_cls=CBLOF, window_size=1000, sliding_size=40, initial_window_X=
X_all[:1000][:,[1,3]],n_clusters=nb_id)]

ensembler = MedianScoreEnsembler()  # Combinaison des scores
recup = np.empty((0,3))  # Tableau pour récuperer les données avec leurs scores.

for X, y in tqdm(iterator.iter(X_all, y_all)):  # Iteration sur les données
    model_scores = np.empty(len(models), dtype=np.float)
    # Calcule le score pour chaque modèles
    for i, model in enumerate(models):
        if i == 0 :
            model.fit_partial(X[[1,2]])
            model_scores[i] = model.score_partial(X[[1,2]])
        if i == 1 :
            model.fit_partial(X[[1,3]])
            model_scores[i] = model.score_partial(X[[1,2]])

    score = ensembler.fit_transform_partial(model_scores)  # Combine les scores des modèles
    recup = np.append(recup, np.array([[X[0],y,score[0]]]), axis=0)

    auroc.update(y, score)  # MAJ AUROC
    aupr.update(y, score)  # MAJ AUPR

# Recupere les données triées par le score d'anomalie dans le tableau recup
a = np.argsort(recup[:,-1])
recup = recup[a]
recup = recup[::-1]
# Sauvegarde le tableau recup au format csv
np.savetxt("data/result/recup1.csv", recup, delimiter=",", fmt='%f')
```

```
print("Window_size=1000")
print("AUROC: ", auroc.get())
print("AUPR: ", aupr.get())
```

```
décompte des données : 6341
donnée anormale : 117
donnée normale : 6224
```

```
6341it [01:52, 56.15it/s]
```

```
Window_size=1000
AUROC:  1.0
AUPR:  0.9999999999999998
```

Système CBLOF sur jeu de données N°2

In [12]:

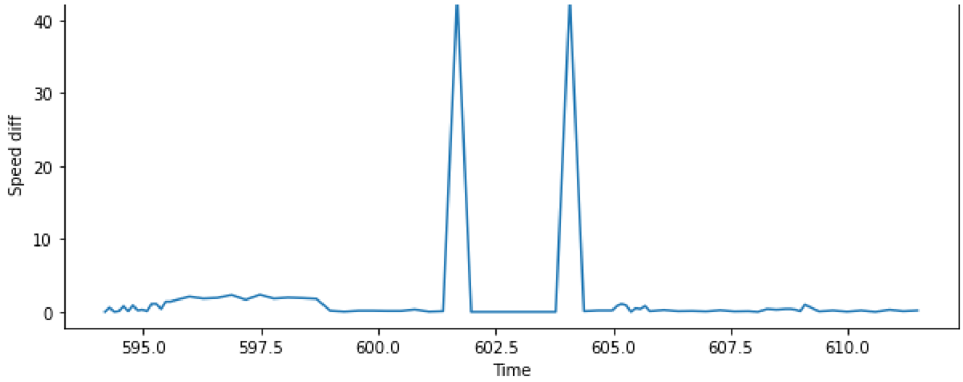
```
### Chargement des données dans un dataframe
columns = ['Time', 'CarId', 'Longitude', 'Latitude', 'Speed', 'Heading', 'Class']
df = pd.read_csv('data/n15cars_25fast.csv', usecols=columns)
### Rajout de nouvelles colonnes avec valeurs à zero
df['ID'] = 0
df['CarId'] = df['CarId'].astype(str)
df['Time diff'] = 0.0
df['Position diff'] = 0.0
df['Speed diff'] = 0.0
df['Heading diff'] = 0.0
df = df[['ID', 'Time', 'CarId', 'Longitude', 'Latitude', 'Speed', 'Heading', 'Time diff', 'Position
diff', 'Speed diff', 'Heading diff', 'Class']]
```

In [13]:

```
### Compare chaque donnée avec la précedente et calcule les variations
NId=0
for index, row in df.iterrows():
    df.at[index, 'ID'] = NId
    NId = NId+1
    if index != 0:
        if row[2] == prec_row[2] and (row[1] - prec_row[1]) < 5:  # Si la donnée n'est pas la premiè
re ou du même identifiant
            df.at[index, 'Time diff'] = abs(row[1] - prec_row[1])
            df.at[index, 'Position diff'] = (abs(row[3] - prec_row[3])+abs(row[4] - prec_row[4]))/(r
ow[1] - prec_row[1])
            df.at[index, 'Speed diff'] = abs(row[5] - prec_row[5])/(row[1] - prec_row[1]) # Differen
ce de vitesse
            df.at[index, 'Heading diff'] = abs(min((row[6]-prec_row[6])%360, (prec_row[6]-
row[6])%360))/(row[1] - prec_row[1]) # Difference de direction
        else:
            df.at[index, 'Time diff'] = 0.0
            df.at[index, 'Position diff'] = 0.0
            df.at[index, 'Speed diff'] = 0.0
            df.at[index, 'Heading diff'] = 0.0
    prec_row = row
```

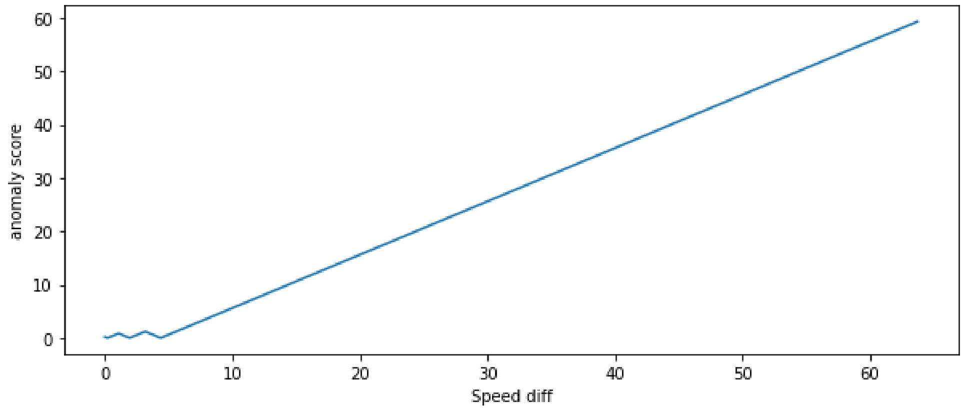
In [14]:

```
data118457 = df[(df['CarId'] == '118457')]
x = data118457['Time']
y = data118457['Speed diff']

plt.figure(figsize=(10,4))
plt.plot(x, y, label='Car 118457')
plt.xlabel('Time')
plt.ylabel('Speed diff')
plt.show();
```



In [15]:

```
cblof = CBLOF()
cblof.fit(df['Speed diff'].values.reshape(-1, 1))
xx = np.linspace(df['Speed diff'].min(), df['Speed diff'].max(), len(df)).reshape(-1,1)
anomaly_score = cblof.decision_function(xx)
outlier = cblof.predict(xx)
plt.figure(figsize=(10,4))
plt.plot(xx, anomaly_score, label='anomaly score')
plt.ylabel('anomaly score')
plt.xlabel('Speed diff')
plt.show();
```



In [16]:

```
minmax = MinMaxScaler(feature_range=(0, 1))
df[['CarId', 'Speed diff', 'Heading diff', 'Position diff']] = minmax.fit_transform(df[['CarId', 'S
peed diff', 'Heading diff', 'Position diff']])
df[['CarId', 'Speed diff', 'Heading diff', 'Position diff']].head()
```

Out[16]:

	CarId	Speed diff	Heading diff	Position diff
0	0.0	0.000000	0.000000	0.00
1	0.0	0.009419	0.117117	0.28
2	0.0	0.000000	0.117117	0.26
3	0.0	0.001570	0.180180	0.26
4	0.0	0.012559	0.264264	0.26

In [17]:

```
X1 = df['CarId'].values.reshape(-1,1)
X2 = df['Speed diff'].values.reshape(-1,1)
X3 = df['Heading diff'].values.reshape(-1,1)
X4 = df['Position diff'].values.reshape(-1,1)

X_speed = np.concatenate((X1,X2),axis=1)
X_heading = np.concatenate((X1,X3),axis=1)
```

```
X_position = np.concatenate((X1,X4),axis=1)
```

In [19]:

```
nb_id = df['CarId'].nunique()   # nombre d'identifiants dans le jeu de donnée
# Copie du dataframe
df1 = df
df1['outlier'] = df1['Class']

X_all = pd.DataFrame(df1, columns=['ID', 'CarId', 'Speed diff', 'Heading diff'])
X_all = X_all.to_numpy()
y_all = df1['Class'].to_numpy()

XS=np.size(X_all[:,0])
Y1=(y_all[:] == 1).sum()
Y2=(y_all[:] == 0).sum()

print("décompte des données :", XS)
print("donnée anormale :", Y1)
print("donnée normale :", Y2)

X_all, y_all = shuffle(X_all, y_all)  # Modification aléatoire de l'ordre des données
iterator = ArrayStreamer(shuffle=False)   # Simule l'arrivé des données en streaming

auroc = AUROCMetric()  # évaluation AUROC
aupr = AUPRMetric()   # évaluation AUPR

models = [ReferenceWindowModel(model_cls=CBLOF, window_size=1000, sliding_size=40, initial_window_X
=X_all[:1000][:,[1,2]],n_clusters=nb_id),
         ReferenceWindowModel(model_cls=CBLOF, window_size=1000, sliding_size=40, initial_window_X=
X_all[:1000][:,[1,3]],n_clusters=nb_id)]

ensembler = MedianScoreEnsembler()  # Combinaison des scores
recup = np.empty((0,3))   # Tableau pour récuperer les données avec leurs scores.

for X, y in tqdm(iterator.iter(X_all, y_all)):  # Iteration sur les données
    model_scores = np.empty(len(models), dtype=np.float)
    # Calcule le score pour chaque modèles
    for i, model in enumerate(models):
        if i == 0 :
            model.fit_partial(X[[1,2]])
            model_scores[i] = model.score_partial(X[[1,2]])
        if i == 1 :
            model.fit_partial(X[[1,3]])
            model_scores[i] = model.score_partial(X[[1,2]])

    score = ensembler.fit_transform_partial(model_scores)  # Combine les scores des modèles
    recup = np.append(recup, np.array([[X[0],y,score[0]]]), axis=0)

    auroc.update(y, score)  # MAJ AUROC
    aupr.update(y, score)  # MAJ AUPR

# Recupere les données triées par le score d'anomalie dans le tableau recup
a = np.argsort(recup[:,-1])
recup = recup[a]
recup = recup[::-1]
# Sauvegarde le tableau recup au format csv
np.savetxt("data/result/recup2.csv", recup, delimiter=",", fmt='%f')

print("Window_size=1000")
print("AUROC: ", auroc.get())
print("AUPR: ", aupr.get())
```

```
décompte des données : 6341
donnée anormale : 117
donnée normale : 6224
```

```
6341it [00:42, 148.04it/s]
```

```
Window_size=1000
AUROC:  1.0
AUPR:  0.9999999999999998
```

Système CBLOF sur jeu de données N°3

In [20]:

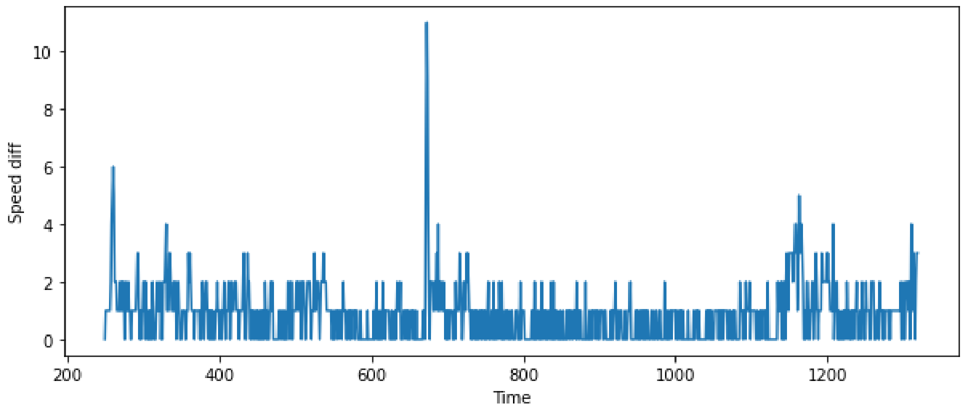
```
### Chargement des données dans un dataframe
columns = ['TimeStep', 'TripID', 'Latitude', 'Longitude', 'Speed', 'Heading']
df = pd.read_csv('data/DACTEasyDataset.csv', usecols=columns)
### Rajout de nouvelles colonnes avec valeurs à zero
df['ID'] = 0
df['Time diff'] = 0.0
df['Position diff'] = 0.0
df['Speed diff'] = 0.0
df['Heading diff'] = 0.0
df = df[['ID', 'TimeStep', 'TripID', 'Longitude', 'Latitude', 'Speed', 'Heading', 'Time diff', 'Pos
ition diff', 'Speed diff', 'Heading diff']]
```

In [21]:

```
### Compare chaque donnée avec la précedente et calcule les variations
NId=0
for index, row in df.iterrows():
    df.at[index, 'ID'] = NId
    NId = NId+1
    if index != 0:
        if row[2] == prec_row[2] and (row[1] - prec_row[1]) < 5:  # Si la donnée n'est pas la premiè
re ou du même identifiant
            df.at[index, 'Time diff'] = abs(row[1] - prec_row[1])
            df.at[index, 'Position diff'] = (abs(row[3] - prec_row[3])+abs(row[4] - prec_row[4]))/(r
ow[1] - prec_row[1])
            df.at[index, 'Speed diff'] = abs(row[5] - prec_row[5])/(row[1] - prec_row[1]) # Differen
ce de vitesse
            df.at[index, 'Heading diff'] = abs(min((row[6]-prec_row[6])%360, (prec_row[6]-
row[6])%360))/(row[1] - prec_row[1]) # Difference de direction
        else:
            df.at[index, 'Time diff'] = 0.0
            df.at[index, 'Position diff'] = 0.0
            df.at[index, 'Speed diff'] = 0.0
            df.at[index, 'Heading diff'] = 0.0
    prec_row = row
```

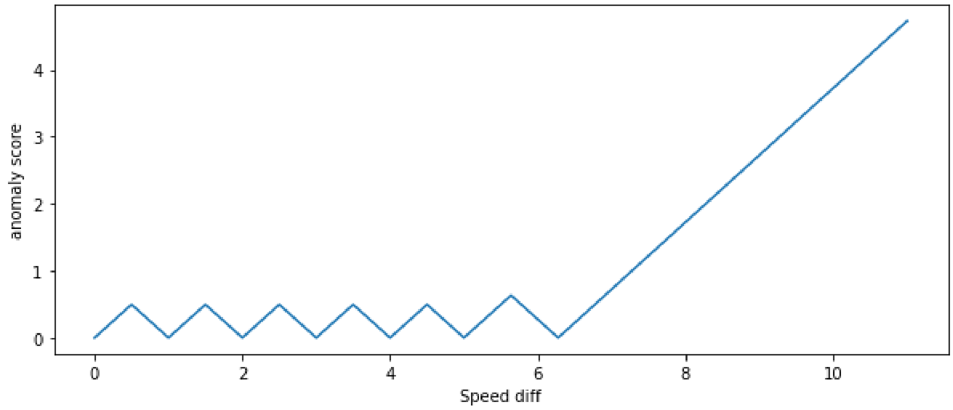
In [22]:

```
data1 = df[(df['TripID'] == 26)]
x = data1['TimeStep']
y = data1['Speed diff']

plt.figure(figsize=(10,4))
plt.plot(x, y)
plt.xlabel('Time')
plt.ylabel('Speed diff')
plt.show();
```



In [23]:

```
cblof = CBLOF()
cblof.fit(df['Speed diff'].values.reshape(-1, 1))
xx = np.linspace(df['Speed diff'].min(), df['Speed diff'].max(), len(df)).reshape(-1,1)
anomaly_score = cblof.decision_function(xx)
outlier = cblof.predict(xx)
plt.figure(figsize=(10,4))
plt.plot(xx, anomaly_score, label='anomaly score')
plt.ylabel('anomaly score')
plt.xlabel('Speed diff')
plt.show();
```



In [24]:

```
minmax = MinMaxScaler(feature_range=(0, 1))
df[['TripID', 'Speed diff', 'Heading diff']] = minmax.fit_transform(df[['TripID', 'Speed diff', 'He
ading diff']])
df[['TripID', 'Speed diff', 'Heading diff']].head()
```

Out[24]:

	TripID	Speed diff	Heading diff
0	0.0	0.000000	0.000000
1	0.0	0.000000	0.000000
2	0.0	0.454545	0.000000
3	0.0	0.272727	0.000000
4	0.0	0.090909	0.035714

In [25]:

```
X1 = df['TripID'].values.reshape(-1,1)
X2 = df['Speed diff'].values.reshape(-1,1)
X3 = df['Heading diff'].values.reshape(-1,1)
X4 = df['Position diff'].values.reshape(-1,1)

X_speed = np.concatenate((X1,X2),axis=1)
X_heading = np.concatenate((X1,X3),axis=1)
X_position = np.concatenate((X1,X4),axis=1)
```

In [26]:

```
# Copie du dataframe
df1 = df
X_all = pd.DataFrame(df1, columns=['TripID', 'Speed diff', 'Heading diff'])
X_all = X_all.to_numpy()

XS=np.size(X_all[:,0])

print("décompte des données :", XS)

iterator = ArrayStreamer(shuffle=False)  # Simule l'arrivé des données en streaming

models = [ReferenceWindowModel(model_cls=CBLOF, window_size=1000, sliding_size=40, initial_window_X
```

```
=X_all[:1000][:,[0,1]]),
         ReferenceWindowModel(model_cls=CBLOF, window_size=1000, sliding_size=40, initial_window_X=
X_all[:1000][:,[0,2]])]

ensembler = MedianScoreEnsembler()  # Combinaison des scores
recup = np.empty((0,2))  # Tableau pour récuperer les données avec leurs scores.

for X in tqdm(iterator.iter(X_all)):  # Iteration sur les données
    model_scores = np.empty(len(models), dtype=np.float)
    # Calcule le score pour chaque modèles
    for i, model in enumerate(models):
        if i == 0 :
            model.fit_partial(X[[0,1]])
            model_scores[i] = model.score_partial(X[[0,1]])
        if i == 1 :
            model.fit_partial(X[[0,2]])
            model_scores[i] = model.score_partial(X[[0,2]])

    score = ensembler.fit_transform_partial(model_scores)  # Combine les scores des modèles
    recup = np.append(recup, np.array([[X[0],score[0]]]), axis=0)


# Recupere les données triées par le score d'anomalie dans le tableau recup
a = np.argsort(recup[:,-1])
recup = recup[a]
recup = recup[::-1]
# Sauvegarde le tableau recup au format csv
np.savetxt("data/result/recup3.csv", recup, delimiter=",", fmt='%f')
```

```
59it [00:00, 588.23it/s]
```

```
décompte des données : 47846
```

```
47846it [04:30, 177.18it/s]
```

In [ ]: