Occupancy Detection : Classification

Projet de Machine Learning
M2 IMPAIRS Paris Diderot - 2018
FONTAINE Victor 21301704

Occupancy Detection - Dataset

- UCI Directory : https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+
- Data utilisée pour de la **classification binaire** (occupation d'une pièce). Une data est obtenue chaque minute.
- Il y a 3 dataset fournis : 1 pour le training (8143 entrées) + 2 pour les test (2665 + 9752 entrées) soit un total de **20560** entrées.
- Une entrée correspond à un **id** puis 7 attributs dans l'ordre: **la date** (année-mois-jour heure-minute-seconde), **la temperature** (Celsius), **l'humidité relative** (%), **la lumière** (lux), **le CO2** (ppm), **le ration d'humidité** (kg-vapeur-d'eau/kg-air), et **l'occupation de la pièce** (1 si occupée, 0 sinon)

Nettoyage du Dataset

```python
def read_data(filename):

    file = open(filename,'r')
    X,Y = [],[]

    for line in file.readlines():
        split = line.split(',')
        xi = [float(split[i]) for i in range (2,7)]
        Y.append(True if (int(split[7].replace('\n','')) == 1) else False)
        X.append(xi)
    file.close()

    return X,Y
```

Nous avons supprimé les deux premiers attributs (**id** et **date**) des datasets, ainsi que la **première ligne** de chaque fichiers (le nom des attributs) et nous les avons concaténé pour obtenir un unique fichier '**dataset_clean.txt**'.

Nous répartissons alors les données dans deux fichiers (**train** et **test**) avec une certaine répartition (**proba**).

```python
def split_lines(input, seed, output1, output2):

    proba = 0.5
    if seed == -1:
        random.seed()
    else:
        random.seed(seed)
    file = open(input,'r')
    f_out1 = open(output1,'w')
    f_out2 = open(output2,'w')

    for line in file.readlines():
        if(random.random() < proba):
            f_out1.write(line)
        else:
            f_out2.write(line)

    file.close()
    f_out1.close()
    f_out2.close()
```

K Nearest Neighbours

```python
def k_nearest_neighbors(x, points, dist_function, k):

    nearest = [-1 for q in range(k)] #equivalent a [-1]*k
    nearest_dist = [-1]*k
    n = len(points)
    variances = calcul_variances(points)

    for i in range(n):
        d = dist_function(x,points[i])
        # d = dist_function(x,points[i],variances)
        for l in range(k):
            if(nearest[l] == -1 or d < nearest_dist[l]):
                nearest.insert(l,i)
                nearest_dist.insert(l,d)
                break

    return nearest[:k]
```

Calcul des K plus proches voisins de x utilisant la distance Euclidienne

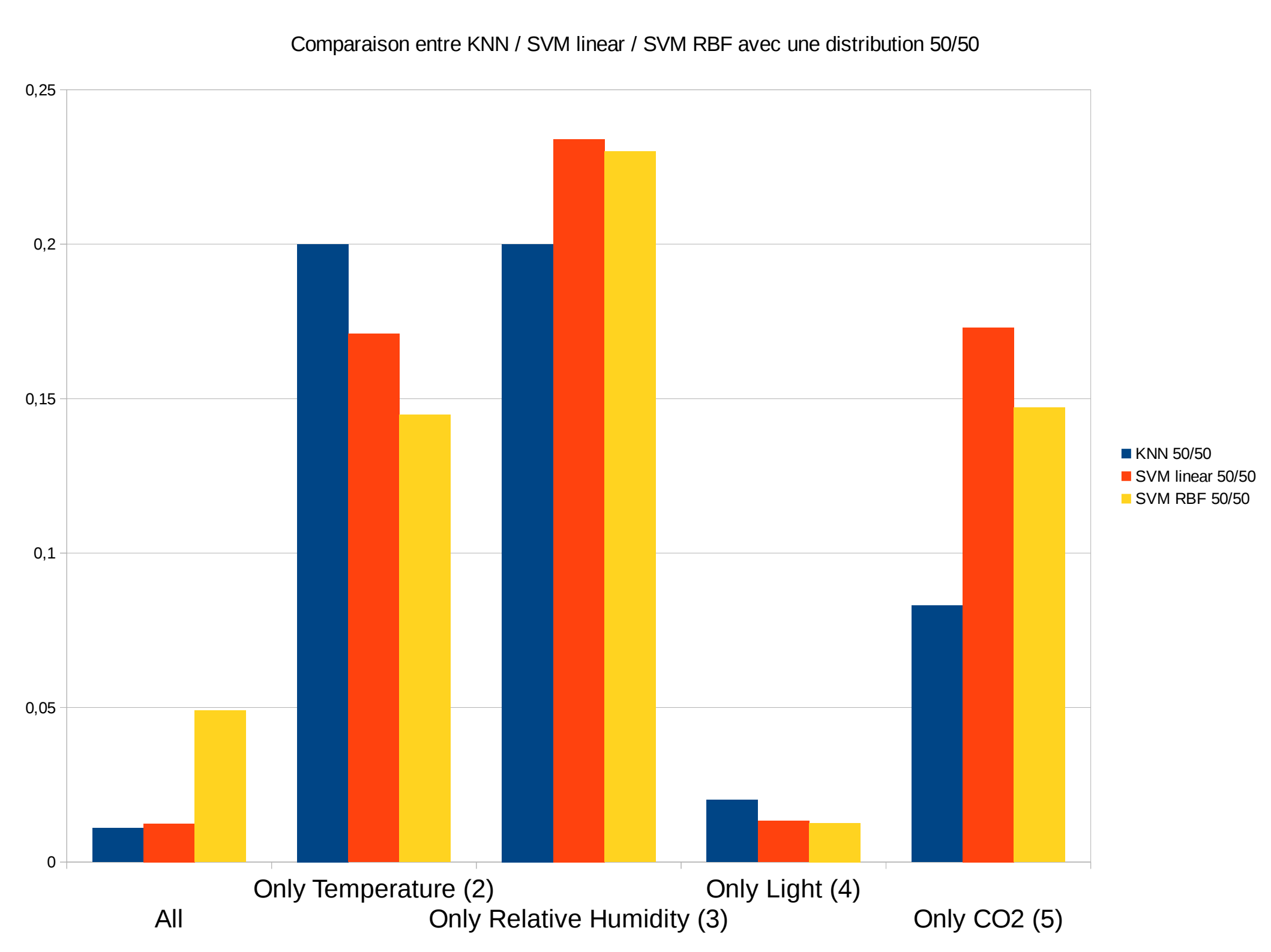
```python
def simple_distance(data1, data2):

    n = len(data1) # which is equal to len(data2)
    sum = 0
    for i in range(n):
        sum += (data1[i]-data2[i])**2

    return math.sqrt(sum)
```

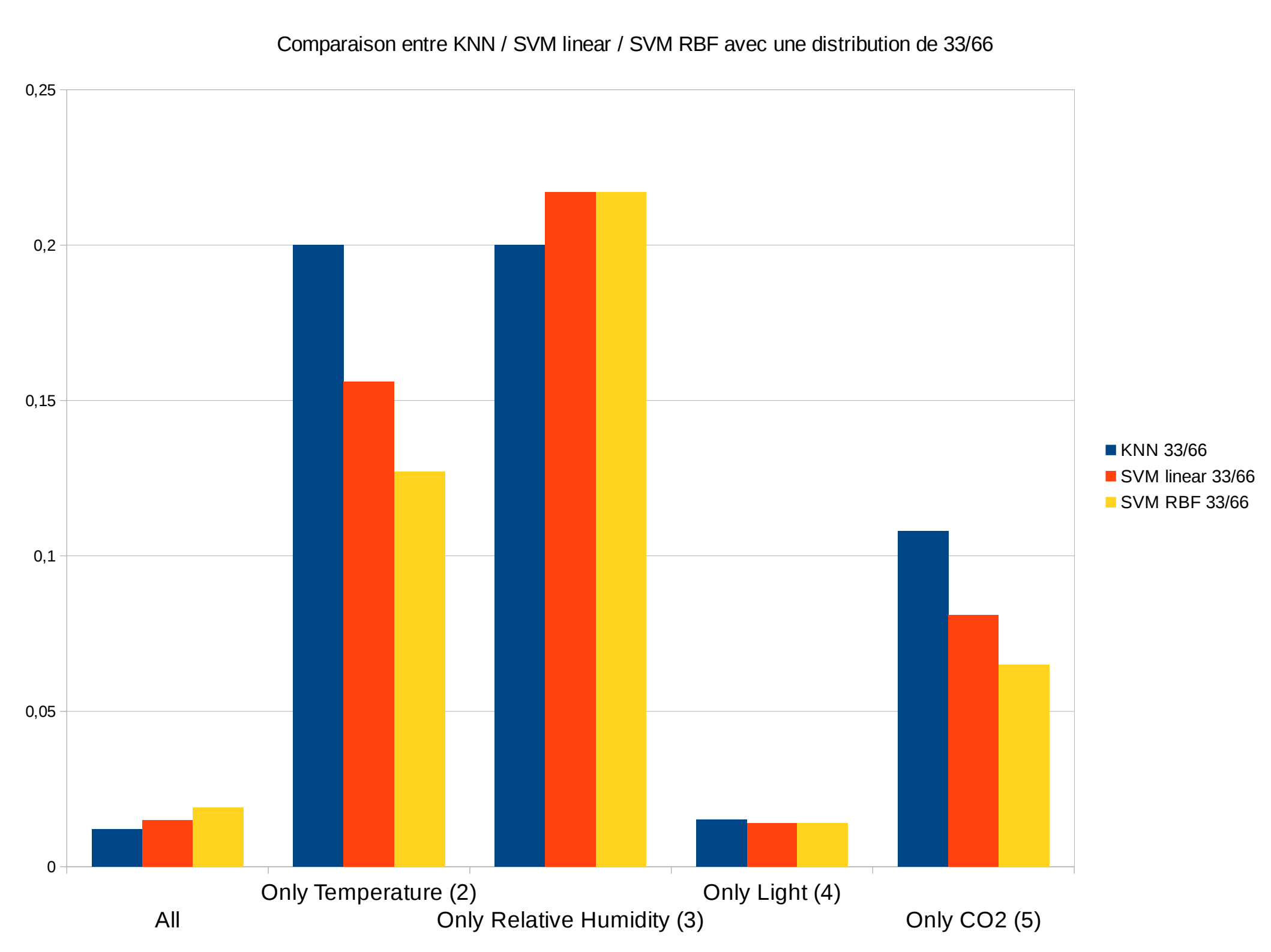
SVM

- Support Vector Classification – sklearn.svm.svc
- 2 SVC : Linéaire et RBF

```
model = SVC(C = 1)
model.fit(train_x,train_y)
predictions = model.predict(test_x)

print("SVC - Radial Basis Function : "+str(eval_SVM(predictions, test_y)))

model = SVC(C = 1, kernel='linear')
model.fit(train_x,train_y)
predictions = model.predict(test_x)

print("SVC - Linear Function : "+str(eval_SVM(predictions, test_y)))
```

```
def eval_SVM(predictions, test_y):
    sum = 0
    for i in range(len(predictions)):
        sum += 1 if predictions[i]==test_y[i] else 0

    return 1 - (sum / float(len(predictions)))
```


Comparaison entre KNN / SVM linear / SVM RBF avec une distribution 50/50
0,25
0,2
0,15
0,1
0,05
0
KNN 50/50
SVM linear 50/50
SVM RBF 50/50
All
Only Temperature (2)
Only Relative Humidity (3)
Only Light (4)
Only CO2 (5)


Comparaison entre KNN / SVM linear / SVM RBF avec une distribution de 33/66
0,25
0,2
0,15
0,1
0,05
0
KNN 33/66
SVM linear 33/66
SVM RBF 33/66
All
Only Temperature (2)
Only Relative Humidity (3)
Only Light (4)
Only CO2 (5)

Conclusion :

On voit ici que l'algorithme qui donne le plus petit taux d'erreur (**1,1%**) est **K Nearest Neighbours** avec tous les attributs pris en compte, mais le **SVM linéaire** donne de bons résultats aussi (**1,24%** avec tous les attributs).

De plus, on trouve de bons résultats si on ne prends en compte que la **lumière** (entre **1,5 %** et **2 %** d'erreur).

	KNN 33/66	SVM linear 33/66	SVM RBF 33/66	KNN 50/50	SVM linear 50/50	SVM RBF 50/50
All	0,012	0,015	0,019	0,011	0,0124	0,049
Only Temperature (2)	0,2	0,156	0,127	0,2	0,171	0,1447
Only Relative Humidity (3)	0,2	0,217	0,217	0,2	0,234	0,23
Only Light (4)	0,0152	0,014	0,014	0,02	0,0132	0,0126
Only CO2 (5)	0,108	0,081	0,065	0,083	0,173	0,147