Applications of Data Analysis

Exercise 4

Tiina Nokelainen 503737

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.neighbors import KNeighborsRegressor

inputs = pd.read_csv("input.csv", header=None)
outputs = pd.read_csv("output.csv", header=None)
coordinates = pd.read_csv("coordinates.csv", header=None)
```

In [2]:
```
# z-score to input datas
zscore = lambda x: (x - x.mean()) / x.std()
inputs = inputs.transform(zscore)

# combine all three datasets into one dataset
# first column has real values, second and third has coordinates, the rest are features
dataset = pd.concat([outputs, coordinates, inputs], axis=1, ignore_index=True).values
```

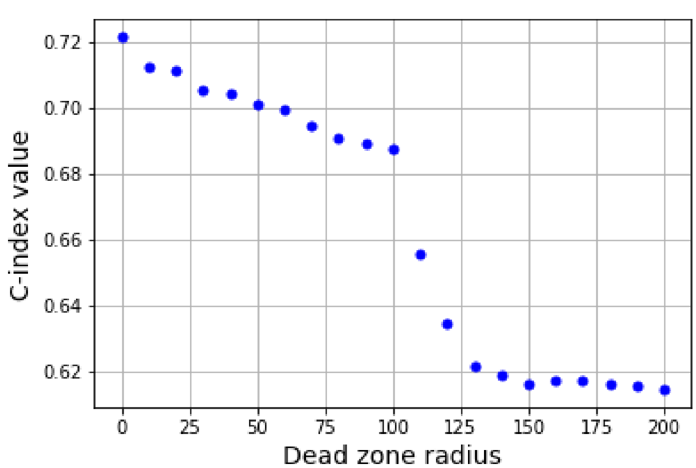
In [3]:
```
def distance(x, y):

    dis = math.pow((y[1] - x[1]),2) + math.pow((y[2] - x[2]),2)
    return math.sqrt(dis)
```

In [4]:
```
# SLOO
# inputs: dataset, dead zone radius: d, number of neighbors: k
# returns: C-index with given values of d and k
def sloo(dataset, d, k):

    # list of predicted values
    predictions = []
    # list of true values
    truevalues = []

    for i in range(len(dataset)):

        # trainset with samples outside of the dead zone (distance is bigger than d)
        trainset = np.array(list(filter(lambda x: distance(x, dataset[i]) > d, dataset)))

        # splits the data into trainset's features X_train and values Y_train
        X_train, Y_train = trainset[:,3:], trainset[:,0]
        # test sample's features: X_test
        # test sample's real value: Y_test
        X_test, Y_test = dataset[i,3:], dataset[i,0]

        knr = KNeighborsRegressor(n_neighbors=k)
        knr.fit(X_train, Y_train)

        # saves predictions and realvalues into separates lists
        predictions += list(knr.predict([X_test]))
        truevalues += [Y_test]

    return cindex(predictions, truevalues)
```

In [5]:
```python
# code from the slides
# inputs: list of predicted velus: predictions, list of real values: truevalues
# returns: C-index
def cindex(predictions, truevalues):
    n = 0
    n_sum = 0
    for i in range(len(truevalues)):
        t = truevalues[i]
        p = predictions[i]
        for j in range(i+1,len(truevalues)):
            nt = truevalues[j]
            np = predictions[j]
            if (t != nt):
                n += 1
                if (p < np and t < nt) or (p > np and t > nt):
                        n_sum += 1
                elif p == np:
                    n_sum += 0.5
    return n_sum/n
```

In [6]:
```python
k_values = [1,3,5,7,9]
cindexs = []

# SLOO for every value of d and k
for d in range(0, 210, 10):
    d_cindex = {}

    for k in k_values:
        d_cindex[str(k)] = sloo(dataset, d, k)

    # save the best value of c-index and k
    best_k = max(d_cindex, key=d_cindex.get)
    max_c = d_cindex.get(best_k)

    cindexs.append({'d' : d, 'k' : best_k, 'C-index' : round(max_c, 4)})
```

In [24]:
```python
cindex_values = []
d_values = []
for i in range(21):
    cindex_values.append(cindexs[i].get('C-index'))
    d_values.append(cindexs[i].get('d'))

plt.plot(d_values, cindex_values, 'ro', color='blue', ms=5)
plt.xlabel('Dead zone radius', fontsize=14)
plt.ylabel('C-index value', fontsize=14)
plt.grid(True)
plt.show()
```



Best C-index values for each dead zone radius

Dead zone radius	Value of k	C-index value
0	9	0.7213
10	9	0.712
20	9	0.7112
30	9	0.7052
40	9	0.7041
50	9	0.7009
60	9	0.6992
70	9	0.6945
80	9	0.6908
90	7	0.6893
100	7	0.6875
110	7	0.6558
120	7	0.6346
130	7	0.6214
140	7	0.6192
150	7	0.6164
160	7	0.6175
170	7	0.6171
180	7	0.6161
190	7	0.6158
200	7	0.6147