CWC 2019 MIMO Positioning Contest: Nearest Neighbor Solution

Henrik Rydén
Email: {henrik.a.ryden}@ericsson.com

***Abstract*—This solution aims at locating the device by using fingerprinting, more specifically using a nearest neighbor regressor. The input to the nearest neighboring regressor is the aboslute of the channel frequency response for each antenna, and the closest training data sample was used to predict an output location. Using a training versus test split of 0.9, we could reach an average error distance of 0.02 meters for 16 antennas. Our results also show how the simulation degrades with the number of antennas, and the source code for the solution is provided in the appendix.**

I. Model selection

The dataset contains 17486 samples over 5032 unique locations, or 782 unique locations when rounded to decimeter accuracy. The available channel estimate is of dimension 924 x 16 values per sample. The low number of samples in combination with a high number of input features and output values motivated the use of fingerprinting.

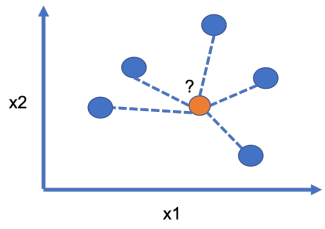


Fig. 1. Nearest neighbor with two features

The proposed model is a nearest neighborg regressor, where the network stores the samples of the training data, and predicts a new sample according to the closest sample in the training set according to figure 1. The model building only comprises filling the dataset into the memory, while the testing phase comprises finding the closest sample. The testing phase is hence a much more time-consuming step and does not scale well with larger datasets. The number of stored samples are (if no test/train split), $17486 \times 16 \times 924 = 258513024$, thus requiring 1 Gb ram if each element is stored as a 4 byte floating point. Due to the low number of samples in our problem, it is possible to execute the model prediction on a low-end PC.

The dataset was first evaluated by investigating the correlation of the channel frequency response for each sample against the other samples. The investigations showed how the correlation of the absolute of the channel frequency response are highly proportional to the geographical distance between samples.

II. ML Results

The nearest neighbor regressor was built using scikit-learn, see code in appendix. The test/training was split by (0.9/0.1), this gives 15736 training samples and 1749 test samples. The average distance error when varying the number of antennas is shown in figure 2. We see how 16 antennas can achieve an average distance error of 0.02 meters, the distance error histogram are shown in figure 3 for the 16 antenna setup.

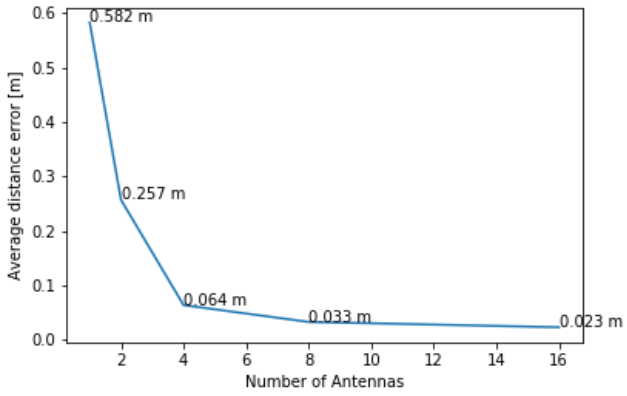


Fig. 2. Distance error with varying number of antennas

The error is further studied by dividing the prediction error results into different groups in Table I, for example samples with less than 0.001m distance error are categorized as fulfilling millimeter accuracy. Having millimeter accuacy for 87 percent of the samples indicate that there are too much correlation within consecutive samples, and it would be interesting to evaluate a dataset collected at another time instance.

Precision	Number of Samples	Percentage
Millimeter accuracy	1533 of 1749	87 %
Centimeter accuracy	1574 of 1749	89 %
Decimeter accuracy	1686 of 1749	96 %
Meter accuracy	1736 of 1749	99 %
10 meter accuracy	1749 of 1749	100 %

TABLE I
Distance error magnitude, millimeter accuracy means that the distance error is less than 0.001 meters.

III. Conclusion

We are able to locate 87 % of devices within a millimeter when dividing the available dataset into 90 % training and

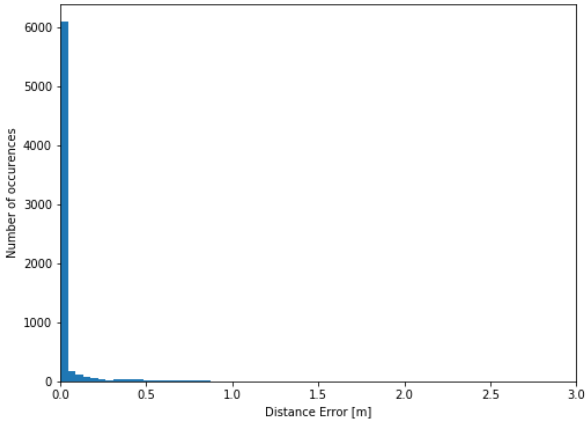


Fig. 3. Distance error with 16 antennas

10 % testing. However, the generalization abilities with the proposed method are questionable due to the dataset properties. Other methods such as Deep neural network would face the same problem in this dataset, since there is low robustness to overfitting when there exists multiple samples with almost identical input/output (duplicates).

Nearest Neighbor Regressor

March 29, 2019

In [1]:
```
import numpy as np
import hdf5storage
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer
from sklearn.model_selection import ShuffleSplit
import matplotlib.pyplot as plt
```

In [2]:
```
def true_dist(y_true, y_pred):
    return (np.sqrt(np.square(np.abs(y_pred[:,0]-y_true[:,0]))+
                    np.square(np.abs(y_pred[:,1]-y_true[:,1]))+
                    np.square(np.abs(y_pred[:,2]-y_true[:,2]))))
```

In [3]:
```
Data_Foldername = './1_Measured_Data'
Meas_Comb_h =    "%s/h_Estimated_CTW_Train.mat" % (Data_Foldername)
Meas_Comb_r =    "%s/r_Position_CTW_Train.mat" % (Data_Foldername)

h_Estimated = hdf5storage.loadmat(Meas_Comb_h)['h_Estimated']
target_pos  = hdf5storage.loadmat(Meas_Comb_r)['r_Position']
```

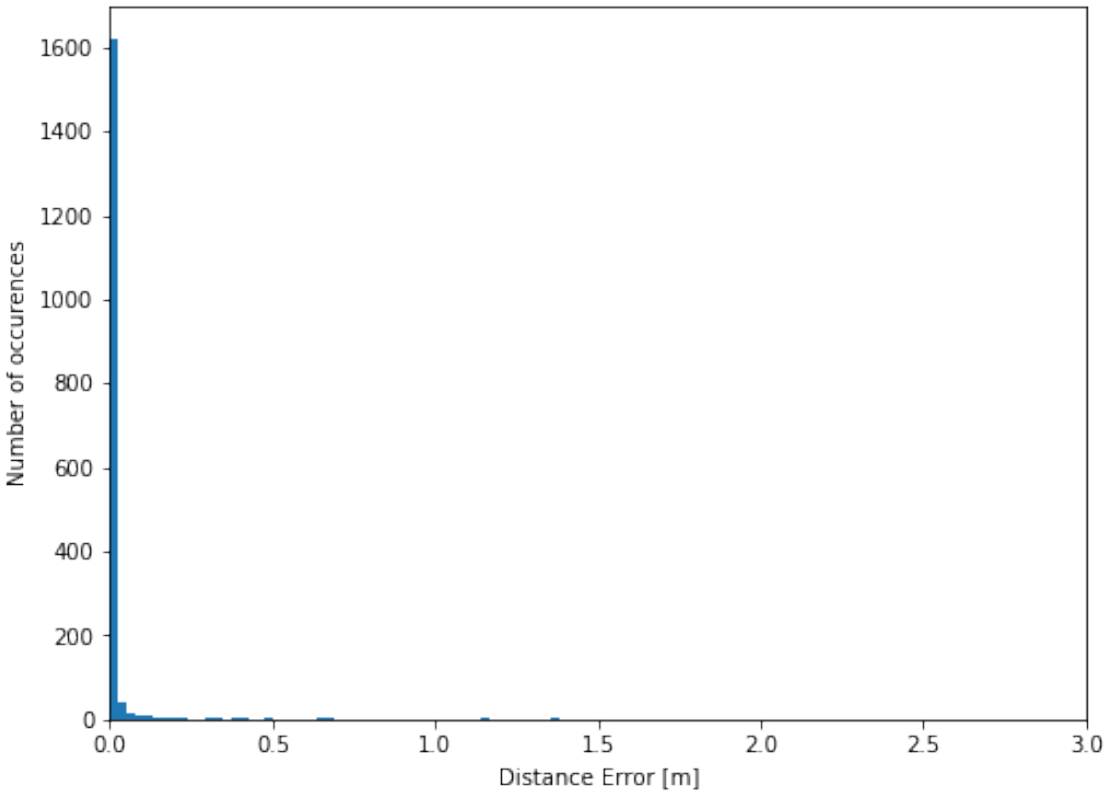
In [4]:
```
X = np.abs(h_Estimated)
targetPosCluster = target_pos
X = np.reshape(X,newshape=(X.shape[0],X.shape[1]*X.shape[2]))
X_train, X_test, y_train, y_test = train_test_split(
    X, targetPosCluster, test_size=0.1, random_state=0)
```

In [5]:
```
neigh = KNeighborsRegressor(n_neighbors=1,p=1)
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
diff = true_dist(y_pred, y_test)
```

In [6]:
```
rmse = np.sqrt(np.mean(diff**2))
print("RMSE = %.3f " % rmse)
print("Average error = %.3f m" % np.mean(diff))
```

```
RMSE = 0.161
Average error = 0.024 m
```

In [7]:
```
%matplotlib inline
plt.figure(figsize=(8,6))
plt.hist(diff,bins=100)
plt.xlim([0,3])
plt.xlabel('Distance Error [m]')
plt.ylabel('Number of occurences')
plt.savefig('distance_error')
```



In [8]:
```
mmPrecision  = np.count_nonzero(diff < 0.001)
cmPrecision  = np.count_nonzero(diff < 0.01)
dmPrecision  = np.count_nonzero(diff < 0.1)
mPrecision   = np.count_nonzero(diff < 1)
m10Precision = np.count_nonzero(diff < 10)
testSamples = diff.shape[0]

print("millimeter precision = %d of %d samples, %3d%%" %
      (mmPrecision,testSamples,100 * (mmPrecision / testSamples)))
print("centimeter precision = %d of %d samples, %3d%%" %
      (cmPrecision,testSamples,100 * (cmPrecision / testSamples)))
print("decimeter  precision = %d of %d samples, %3d%%" %
      (dmPrecision,testSamples,100 * (dmPrecision / testSamples)))
```

```
        print("meter      precision = %d of %d samples, %3d%%" %
              (mPrecision,testSamples,100  * (mPrecision / testSamples)))
        print("10 meter   precision = %d of %d samples, %3d%%" %
              (m10Precision,testSamples,100* (m10Precision / testSamples)))
```

```
millimeter precision = 1534 of 1749 samples,  87%
centimeter precision = 1577 of 1749 samples,  90%
decimeter  precision = 1683 of 1749 samples,  96%
meter      precision = 1736 of 1749 samples,  99%
10 meter   precision = 1749 of 1749 samples, 100%
```

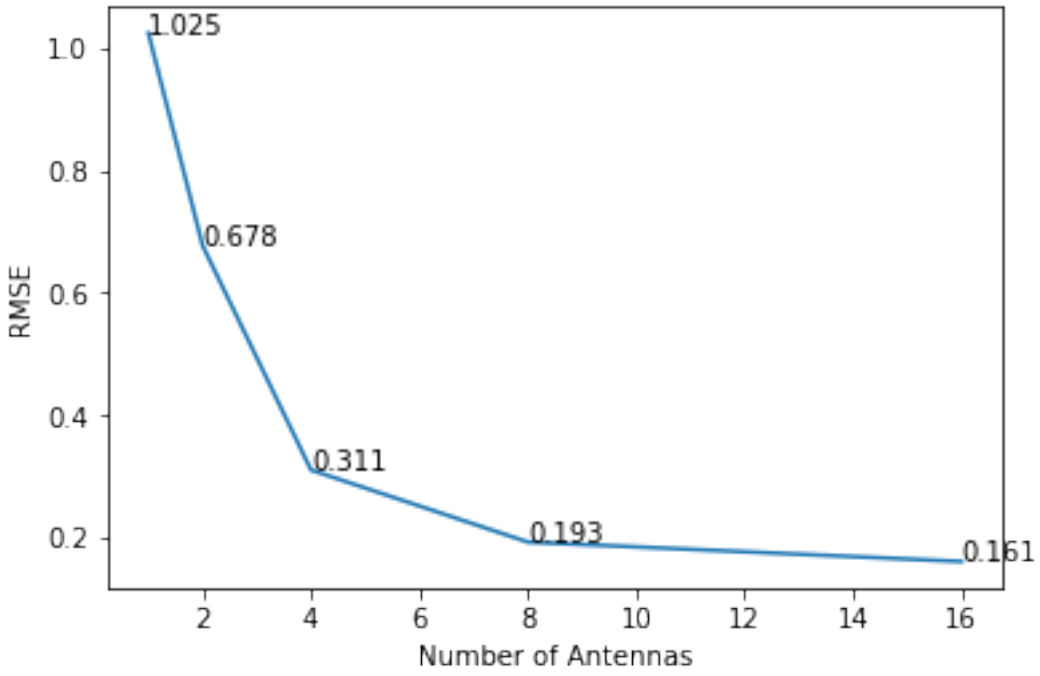
0.0.1 Investigate with varying number of antennas

In [9]:
```
numberOfAntennas = [1,2,4,8,16]
rmse = np.zeros(shape=(len(numberOfAntennas),))
mean_error = np.zeros(shape=(len(numberOfAntennas),))

for idx,antenna in enumerate(numberOfAntennas):
    X = np.abs(h_Estimated[:,0:antenna])
    targetPosCluster = target_pos[:]
    X = np.reshape(X,newshape=(X.shape[0],X.shape[1]*X.shape[2]))

    X_train, X_test, y_train, y_test = train_test_split(
        X, targetPosCluster, test_size=0.1, random_state=0)
    neigh.fit(X_train, y_train)
    y_pred = neigh.predict(X_test)
    diff   = true_dist(y_pred, y_test)
    rmse[idx]       = np.sqrt(np.mean(diff**2))
    mean_error[idx] = np.mean(diff)
    print("Number of antennas = %d " % antenna)
    print("RMSE = %.3f " % rmse[idx])
    print("Average error = %.3f m \n" % np.mean(diff))
```

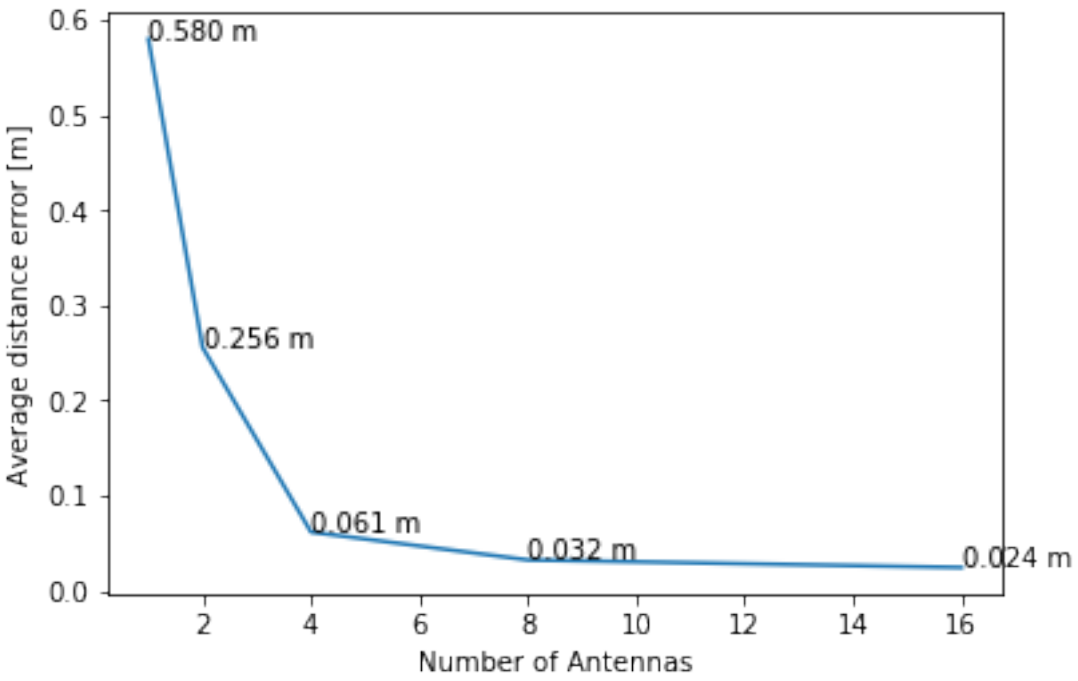
```
Number of antennas = 1
RMSE = 1.025
Average error = 0.580 m

Number of antennas = 2
RMSE = 0.678
Average error = 0.256 m

Number of antennas = 4
RMSE = 0.311
Average error = 0.061 m

Number of antennas = 8
RMSE = 0.193
```

```
Average error = 0.032 m

Number of antennas = 16
RMSE = 0.161
Average error = 0.024 m
```

```
In [10]: fig, ax = plt.subplots()
         ax.plot(numberOfAntennas,rmse)

         for i, txt in enumerate(rmse):
             ax.annotate("%.3f" % rmse[i], (numberOfAntennas[i],rmse[i]))
         ax.set_xlabel('Number of Antennas')
         ax.set_ylabel('RMSE')
         plt.savefig('rmse_antennas')
```




```
In [11]: fig, ax = plt.subplots()
         ax.plot(numberOfAntennas,mean_error)

         for i, txt in enumerate(mean_error):
             ax.annotate("%.3f m" % mean_error[i], (numberOfAntennas[i],mean_error[i]))
         ax.set_xlabel('Number of Antennas')
         ax.set_ylabel('Average distance error [m]')
         plt.savefig('average_error_antennas')
```


0.6
0.5
0.4
0.3
0.2
0.1
0.0
Average distance error [m]
0.580 m
0.256 m
0.061 m
0.032 m
0.024 m
2
4
6
8
10
12
14
16
Number of Antennas