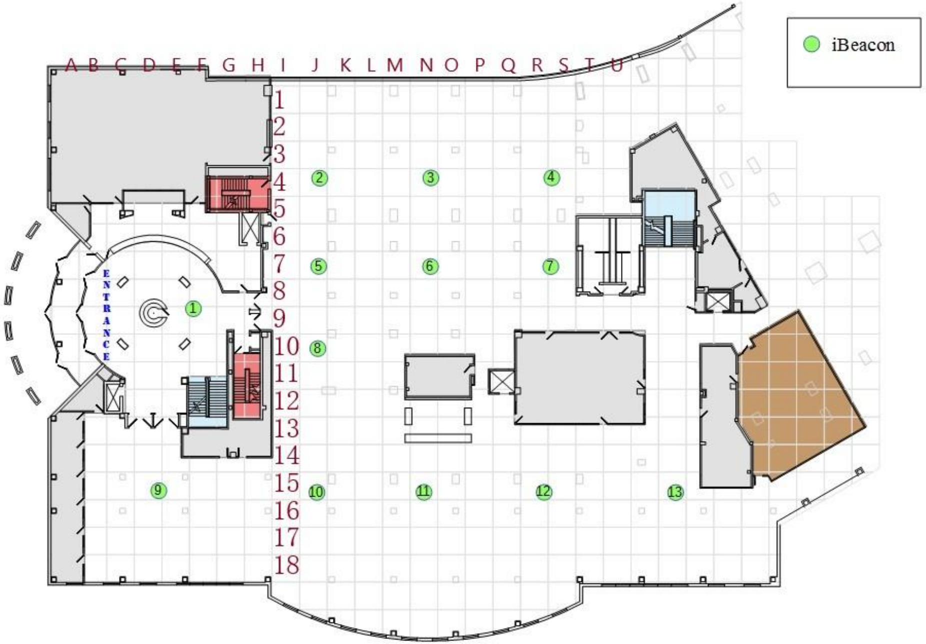
Indoor Localization with BLE and RSSI

Aluno: Victor Fonseca
Professor: Clécio R. Bom

Library Scenario



BLE and RSSI?

- BLE: Bluetooth Low Energy
  - Every smartphone has it
- iBeacon Bluetooth
  - Advertisement Mode
  - No connection
- RSSI: Received Signal Strength Indication
  - negative decibel-milliwatts (dBm)
  - Bigger RSSI values indicate closer proximity to a given iBeacon
  - Standard feature in every smartphone

Approaches to infer position with RSSI

- Distance estimation
  - Calculates the distance between the smartphone and 3 iBeacon using R
  - Applies Trilateration
- Fingerprinting
  - Maps the area
  - Reading the RSSI from all iBeacons at each position

Distance Estimation - RSSI and Distance

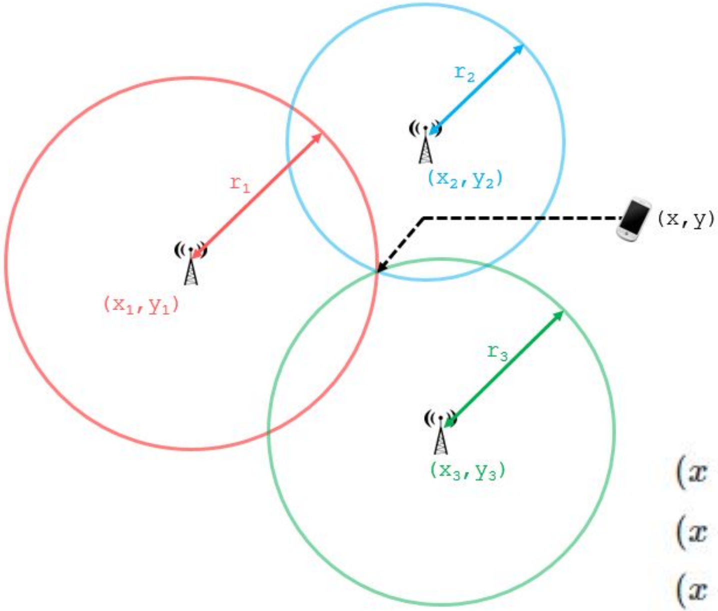
Loss Coefficient

RSSI at 1 meter

$$\mathrm{RSSI} = -10n\log_{10}\left(\frac{d}{d_0}\right) + A_0$$

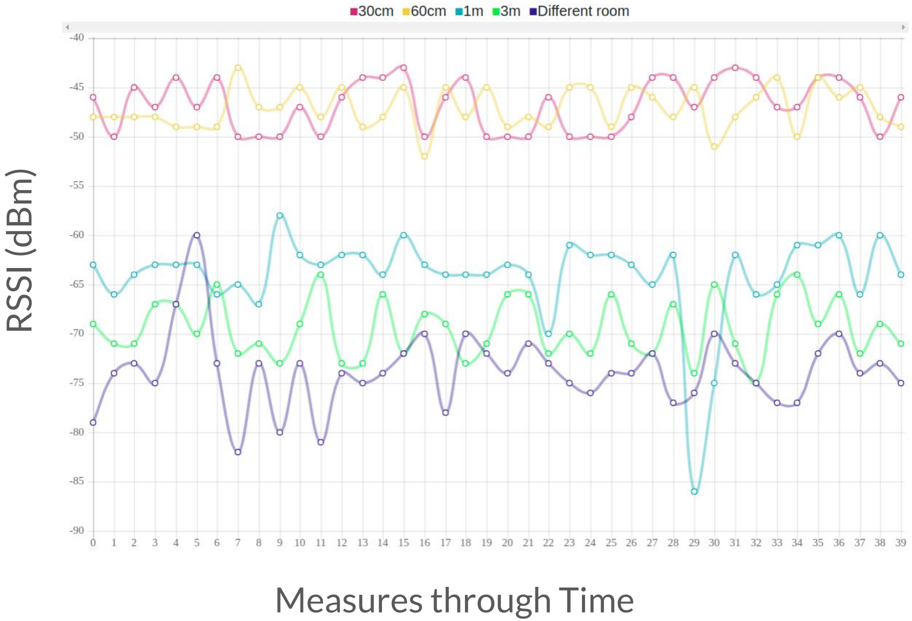
1 meter

Distance Estimation - Trilateration



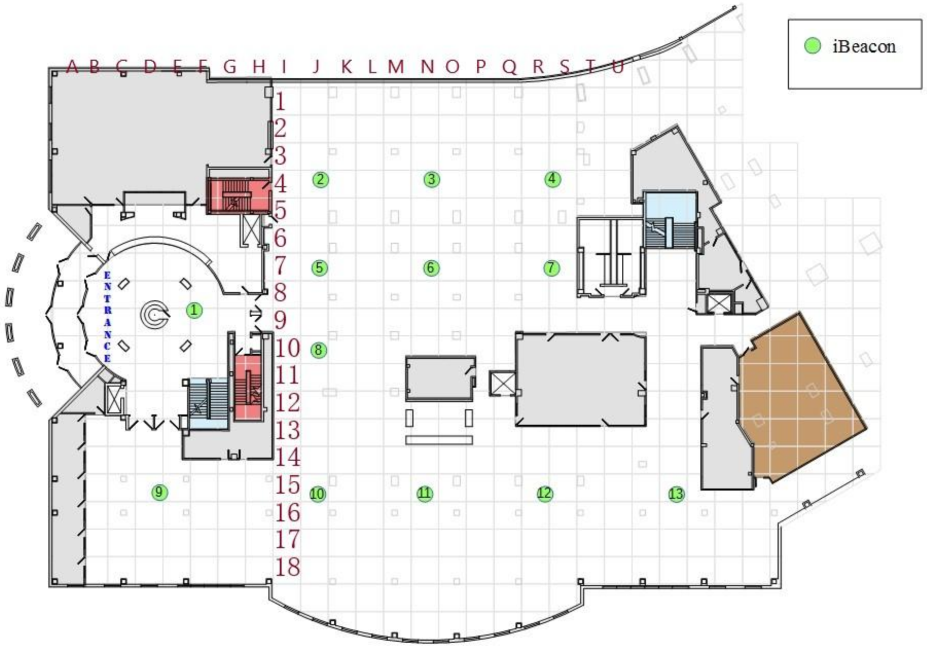
$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$
$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$

Distance Estimation - The Problem with RSSI and Distance



Source: https://www.wouterbulten.nl/

Fingerprinting - Library Scenario



Dataset

- Source: https://www.kaggle.com/mehdimka/ble-rssi-dataset?select=iBeacon_RSSI_Labeled.csv

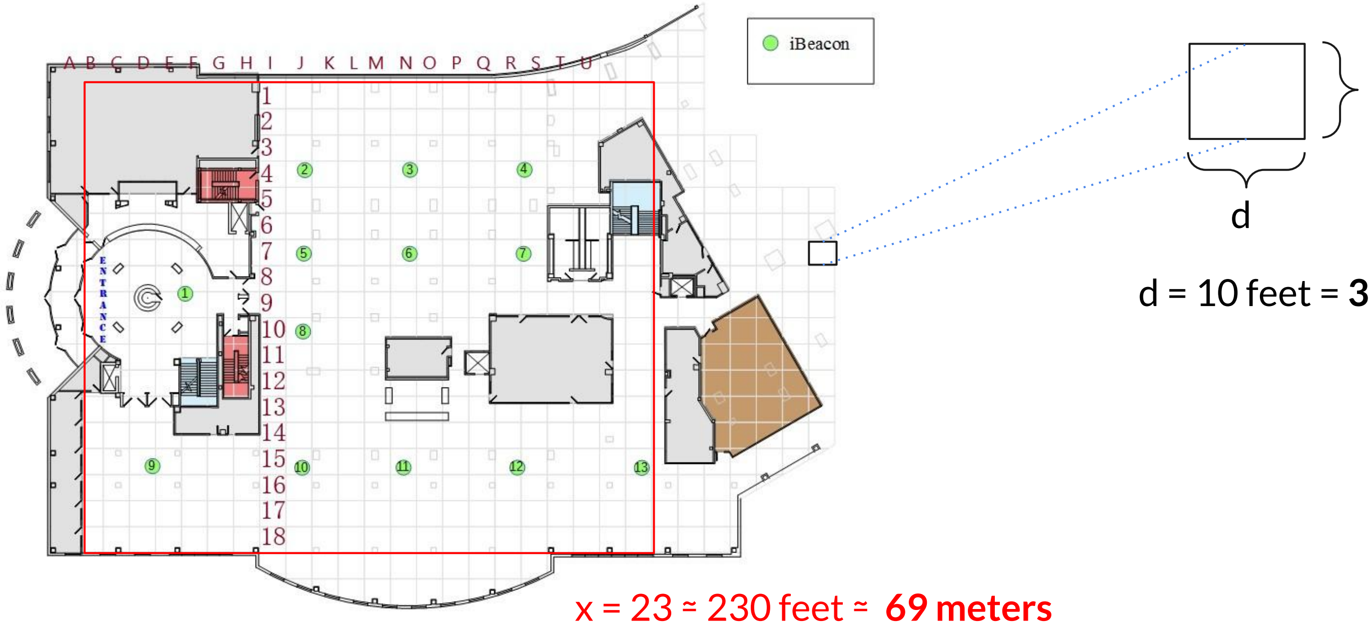
	location	date	b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
0	O02	10-18-2016 11:15:21	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
1	P01	10-18-2016 11:15:19	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
2	P01	10-18-2016 11:15:17	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
3	P01	10-18-2016 11:15:15	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
4	P01	10-18-2016 11:15:13	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200

Convert location column into x and y columns

	date	b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013	x	y
0	10-18-2016 11:15:21	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200	8	2
1	10-18-2016 11:15:19	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200	7	1
2	10-18-2016 11:15:17	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200	7	1
3	10-18-2016 11:15:15	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200	7	1
4	10-18-2016 11:15:13	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200	7	1

Real Distance

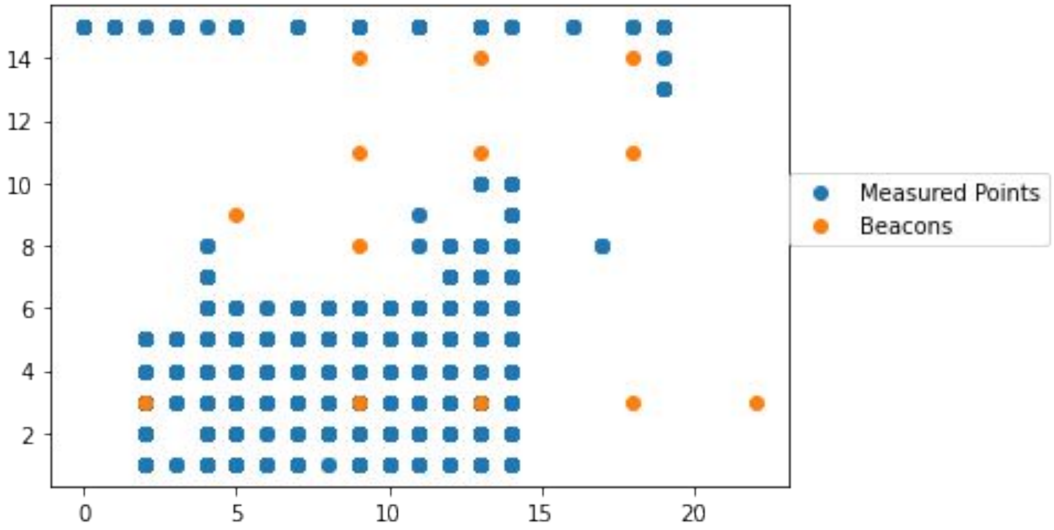
y ≃ **48 meters**



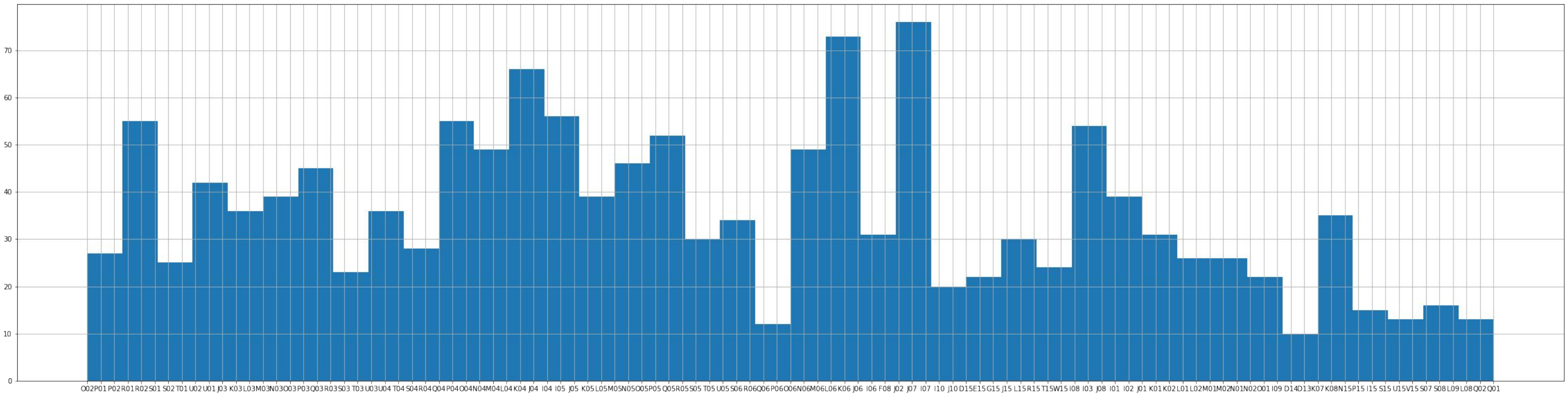
d = 10 feet = **3 meters**

x = 23 ≃ 230 feet ≃ **69 meters**

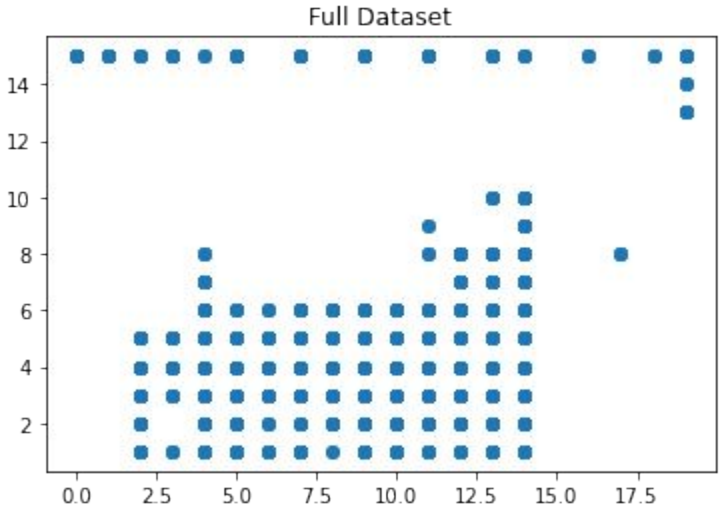
Measured Points x Beacons



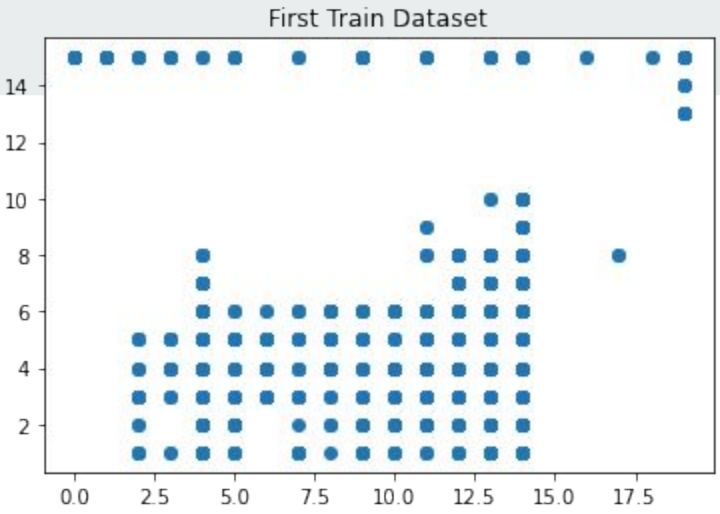
Number of Measures per position



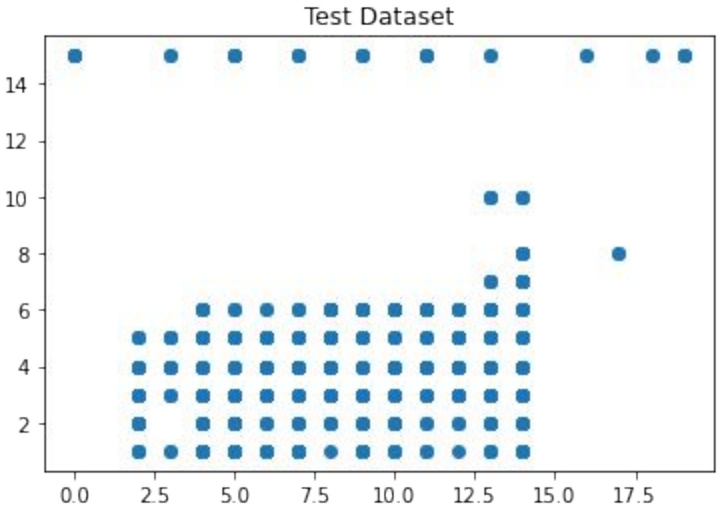
Training and Test



1420 measures

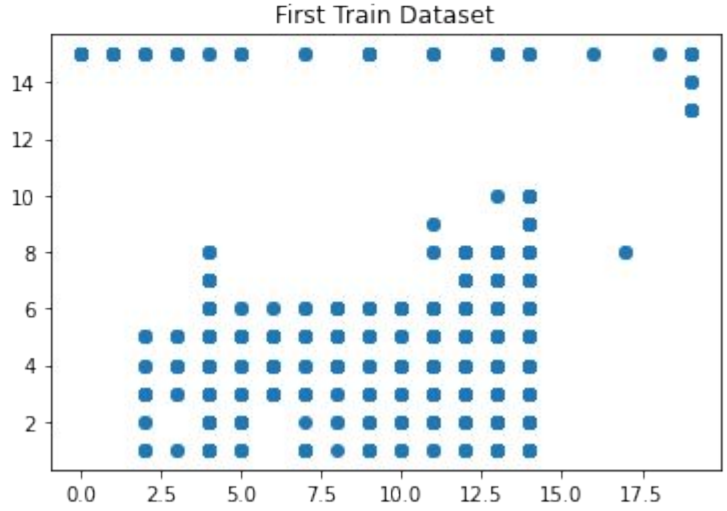


820

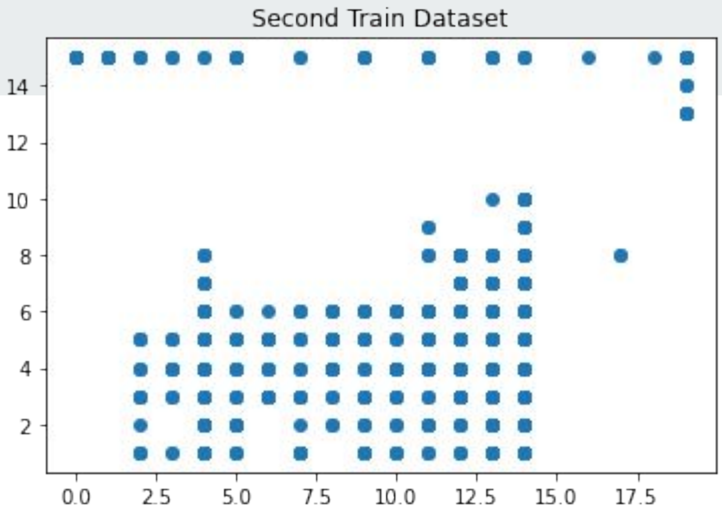


600

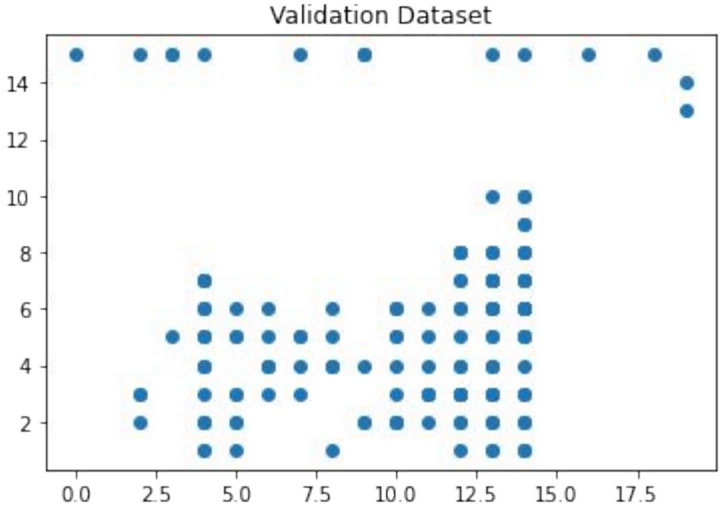
Training and Validation



820 measures



80%



20%

Approaches

- Regression Problem
- Multilayer Perceptron (MLP)
- Convolutional Neural Network (CNN)
- Auto Encoder
- KNN and Random Forest

Multilayer Perceptron (MLP)

MLP - Model

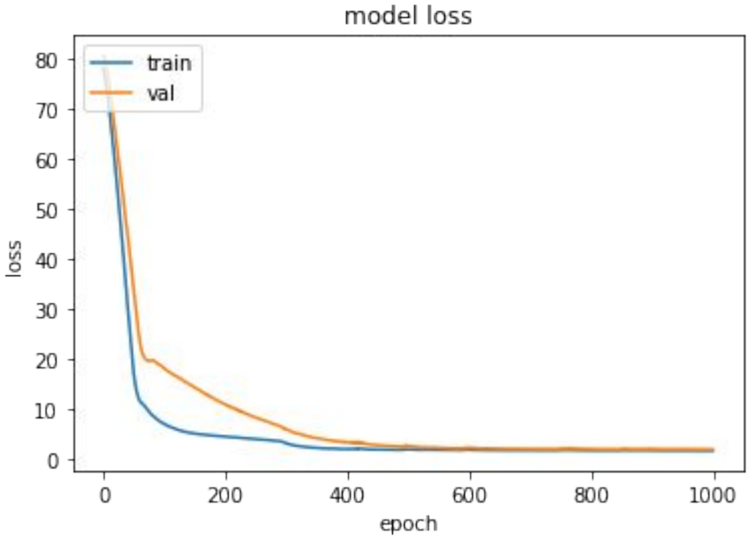
```
model = Sequential()
model.add(Dense(50, input_dim=input_dim, activation='sigmoid'))
model.add(BatchNormalization())
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer=Adam(.001))
```

```
early_stopping = EarlyStopping(monitor='val_loss', patience=100, verbose=0,
                               mode='auto', restore_best_weights=True)

model = create_deep(X_train.shape[1])
out = model.fit(x = X_train, y = y_train,
                validation_data = (X_val, y_val),
                epochs=1000,
                batch_size=1000,
                verbose=1,
                callbacks = [early_stopping])
```

Mean Squared Error - MSE

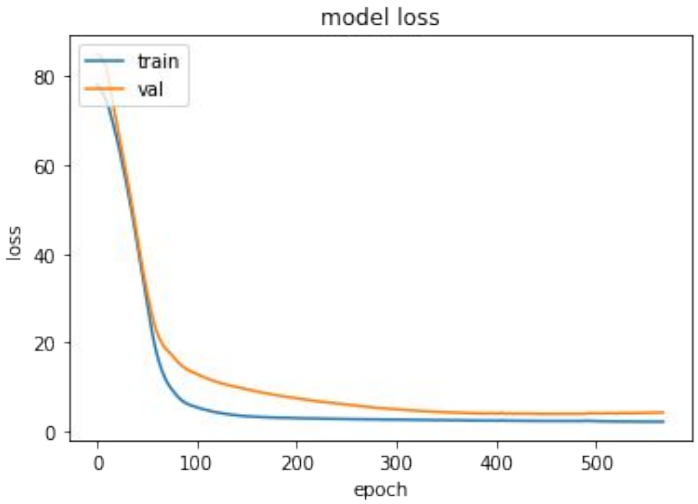
$$\mathrm{MSE} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{test set}} \left(\underbrace{y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i}_{\text{actual value}}\right)^2$$

MLP - First Training

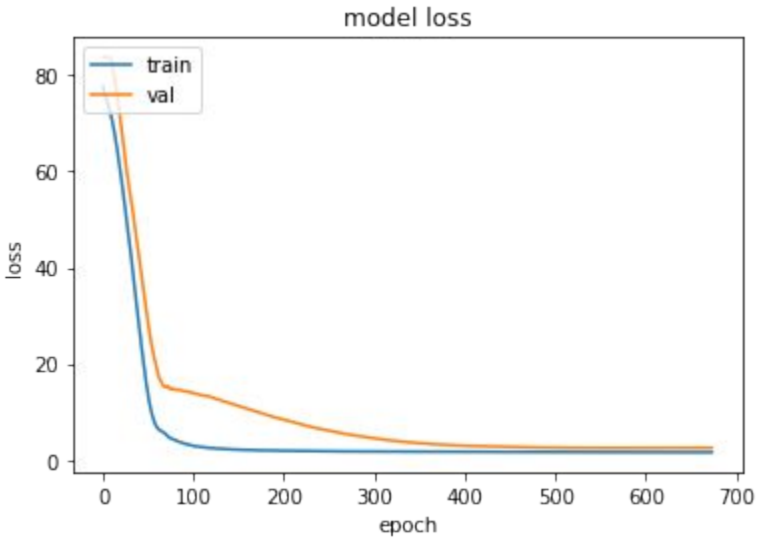


MSE = 1.927

MLP K-Fold CV : Iteration 1 & 2

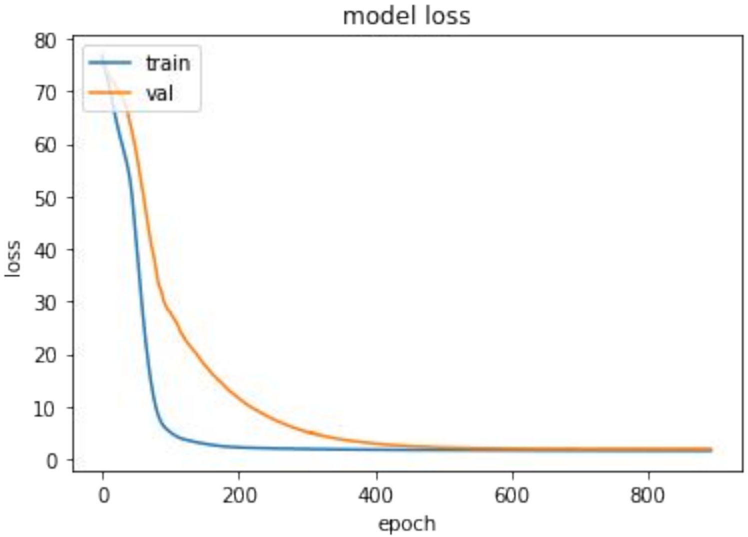


MSE = 3.836

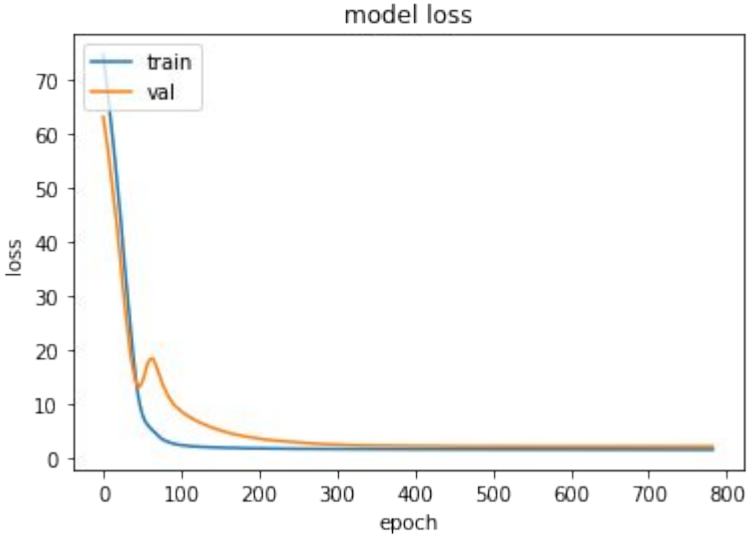


MSE = 2.490

MLP K-Fold CV : Iteration 3& 4

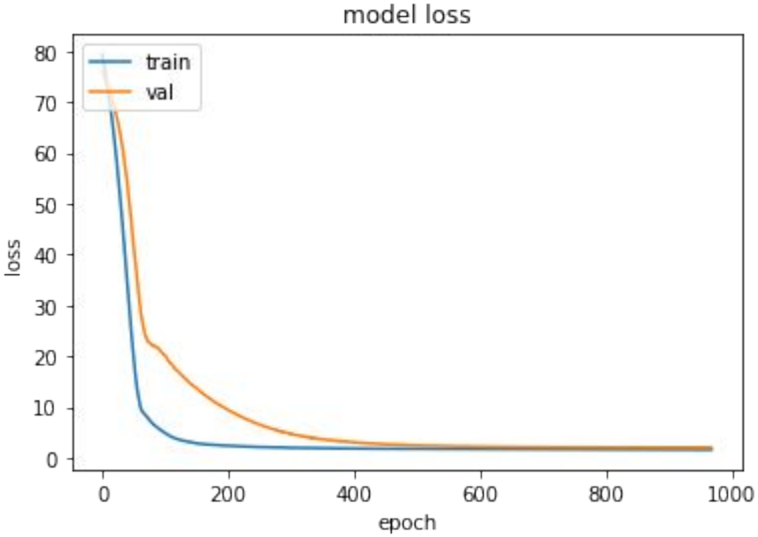


MSE = 1.934



MSE = 2.140

MLP K-Fold CV : Iteration 5



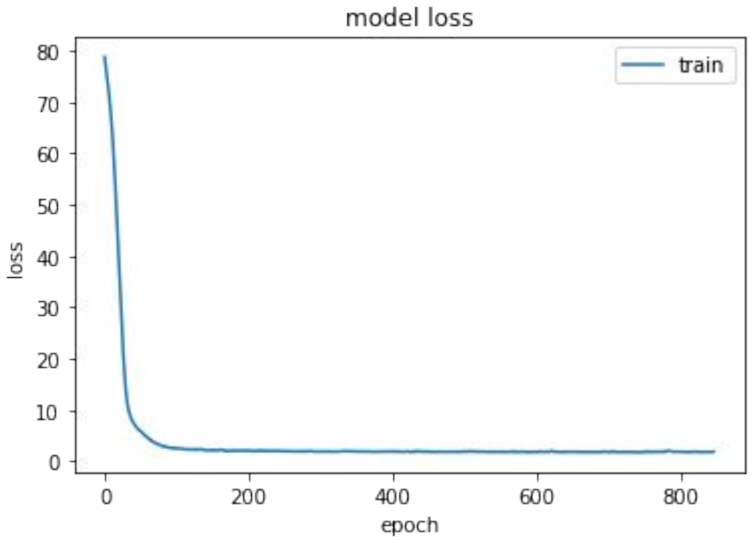
MSE = 2.036

MSE:

- 3.836
- 2.490
- 1.934
- 2.140
- 2.036

Mean = 2.487 (+/- 0.7)

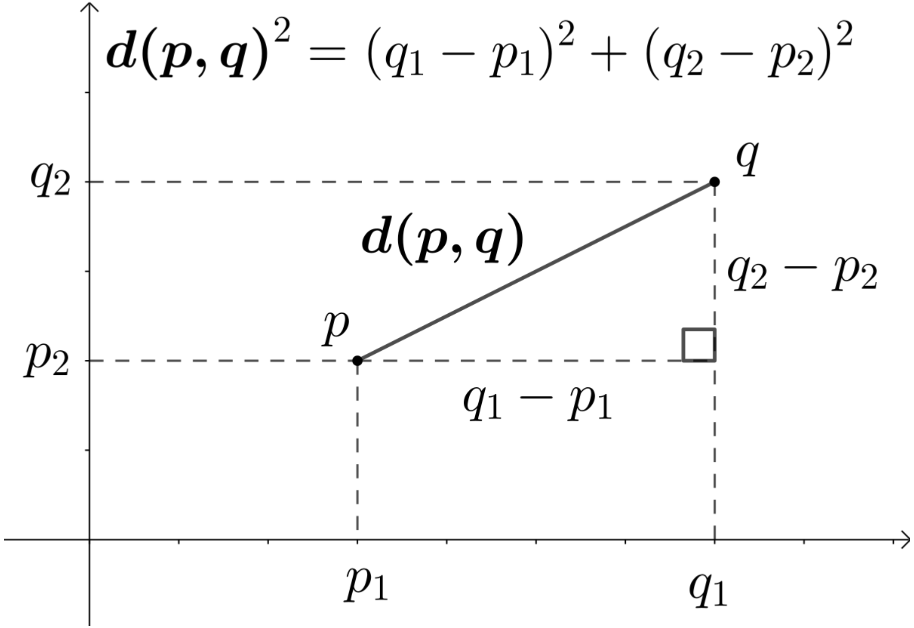
MLP Full Data Training



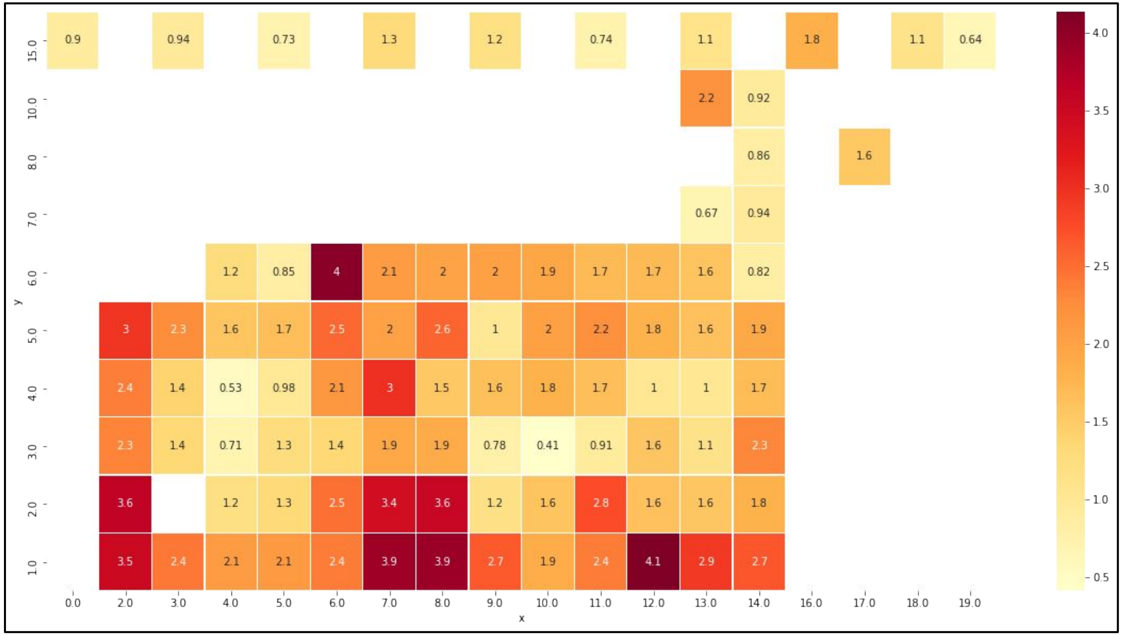
Training MSE = 2.1417

Test MSE = 2.142

Euclidean Distance



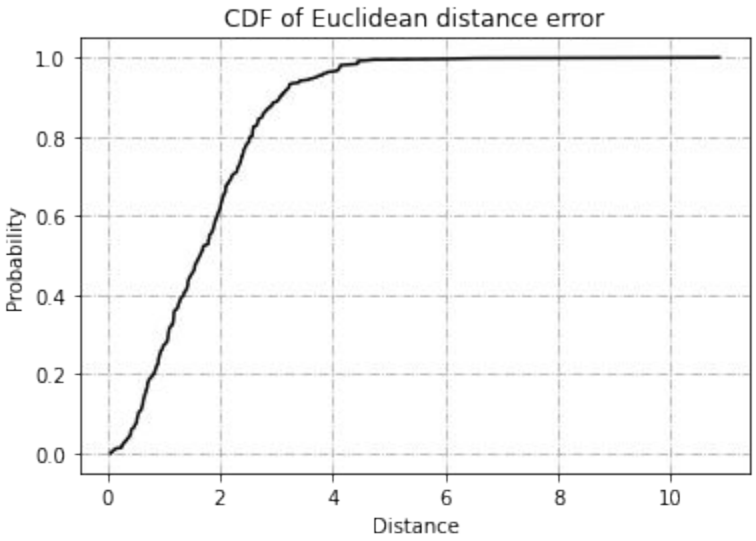
MLP - Heat map Error

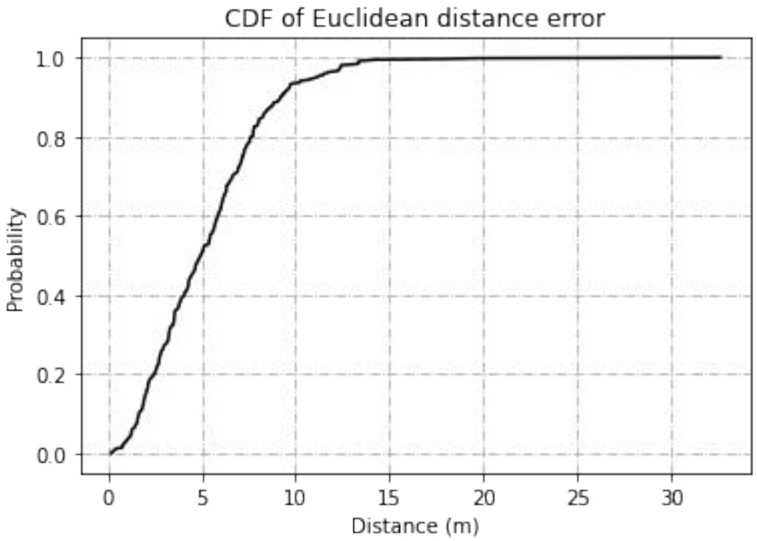


Cumulative Distribution Function - CDF

$$F_X(x) = \mathrm{P}(X \leq x)$$

MLP - CDF of Euclidean Distance Error

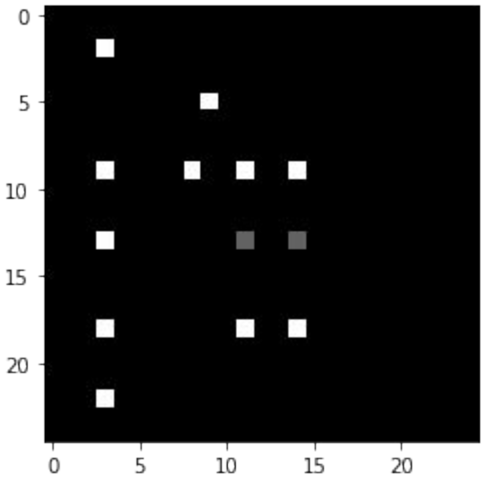




Convolutional Neural Network (CNN)

CNN - Creating Images

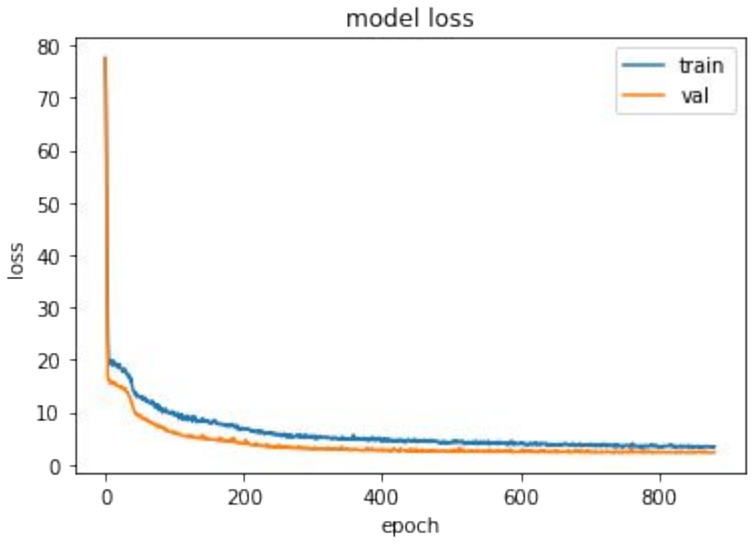
- Each measure in the dataset is turned into a 25x25 matrix, its elements represents a position in the library area



CNN - Model

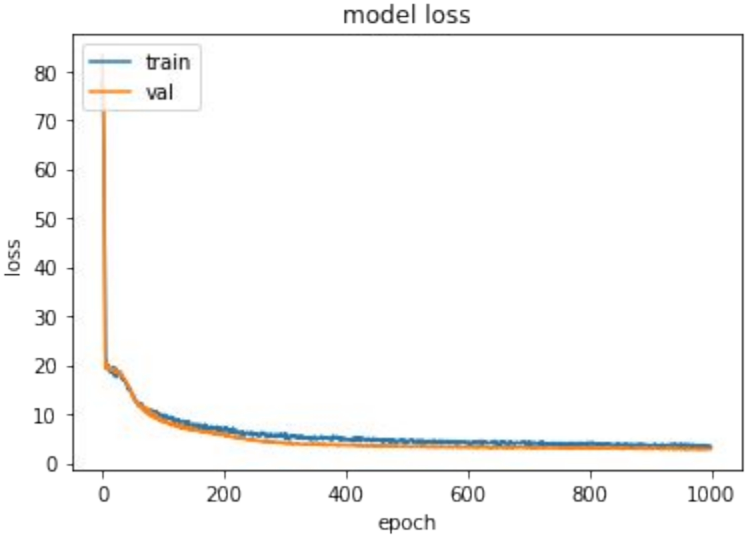
```
def create_deep():
  seed = 7
  np.random.seed(seed)
  inputs = Input(shape=(X_train.shape[1], X_train.shape[2], 1))


  x = Conv2D(3, kernel_size=(3,3), activation='relu', padding = "valid",
             data_format="channels_last", )(inputs)
  x = MaxPooling2D(2)(x)
  x = Conv2D(6, kernel_size=(3,3), activation='relu', padding = "valid",
             data_format="channels_last")(x)
  x = MaxPooling2D(2)(x)
  x = Conv2D(12, kernel_size=(3,3), activation='relu', padding = "valid",
             data_format="channels_last")(x)
  x = Dense(50, activation='relu')(Flatten()(x))
  x = Dropout(0.3)(x)

  predictions = Dense(2, activation='relu')(x)


  model = Model(inputs=inputs, outputs=predictions)
  model.compile(optimizer=Adam(0.001),
                loss='mse')
  return model

model = create_deep()
model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=100, verbose=0,
                               mode='auto', restore_best_weights=True)

out = model.fit(x = X_train_2, y = y_train_2,
                validation_data = (X_val, y_val),
                epochs=1000,
                batch_size=64,
                verbose=0,
                callbacks = [early_stopping])
```

CNN - First Training

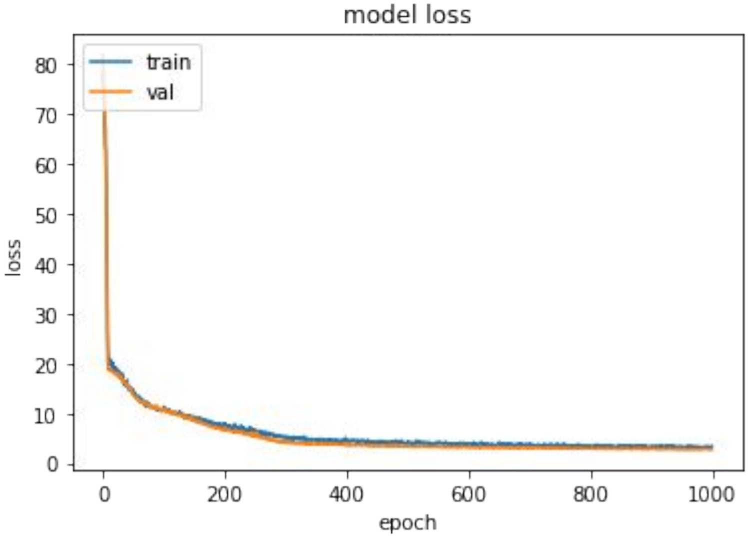


MSE = 2.214

CNN K-Fold CV : Iteration 1 & 2

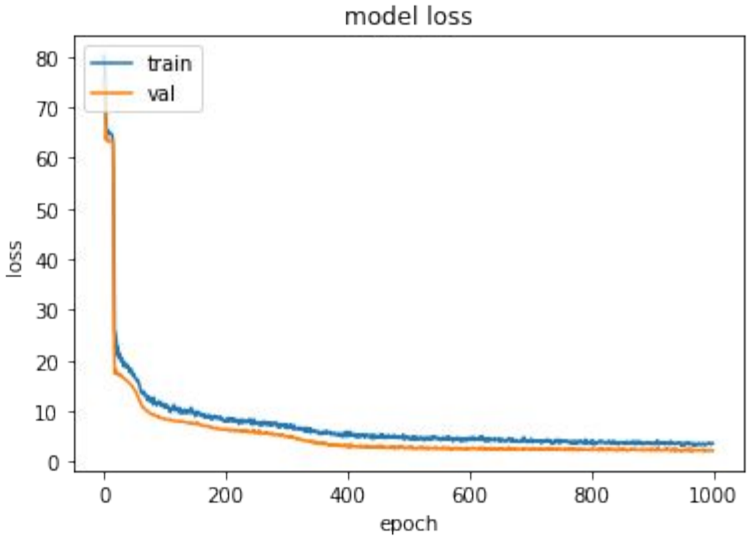


MSE = 2.876

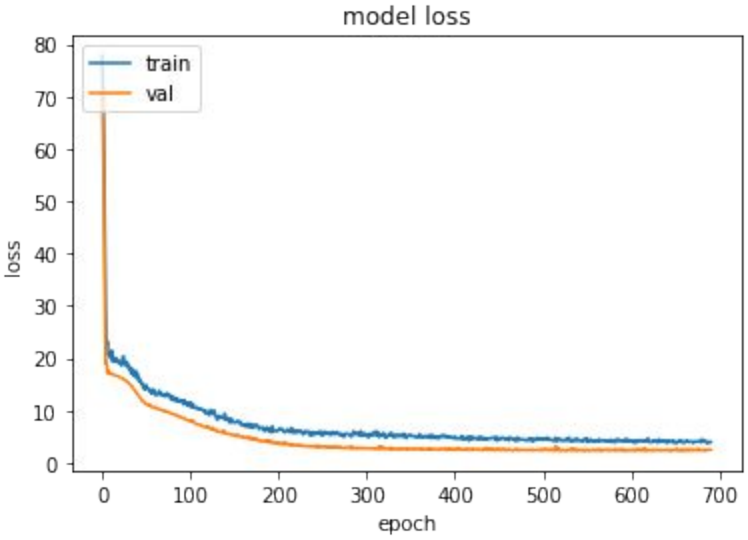


MSE = 2.735

CNN K-Fold CV : Iteration 3& 4

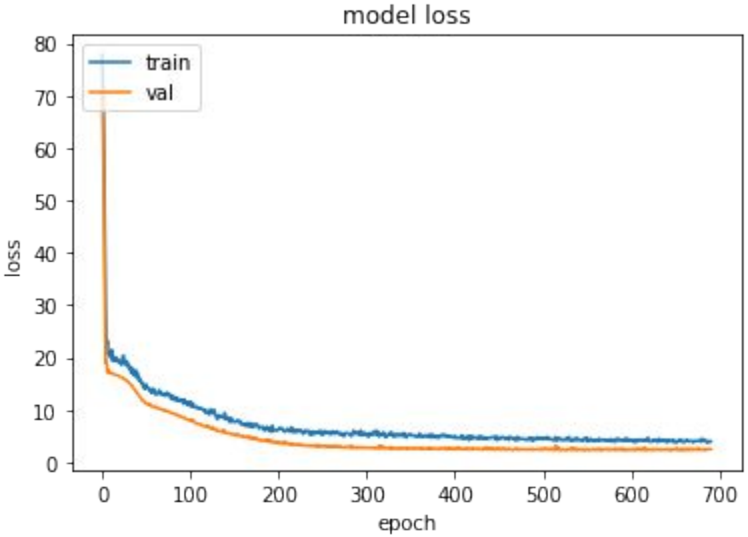


MSE = 2.147



MSE = 2.332

CNN K-Fold CV : Iteration 5



MSE = 2.052

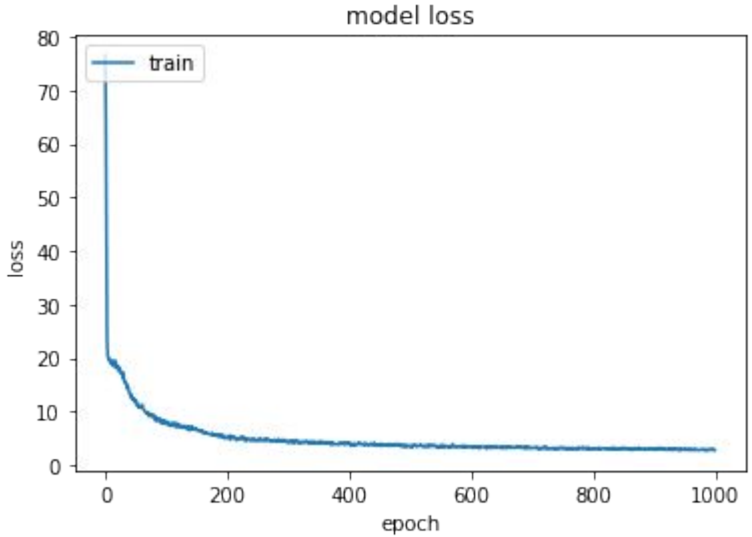
MSE:
- 2.876
- 2.735
- 2.147
- 2.140
- 2.052

Mean = 2.426 (+/- 0.325)

MSE:
- 3.836
- 2.490
- 1.934
- 2.140
- 2.036

Mean = 2.487 (+/- 0.7)

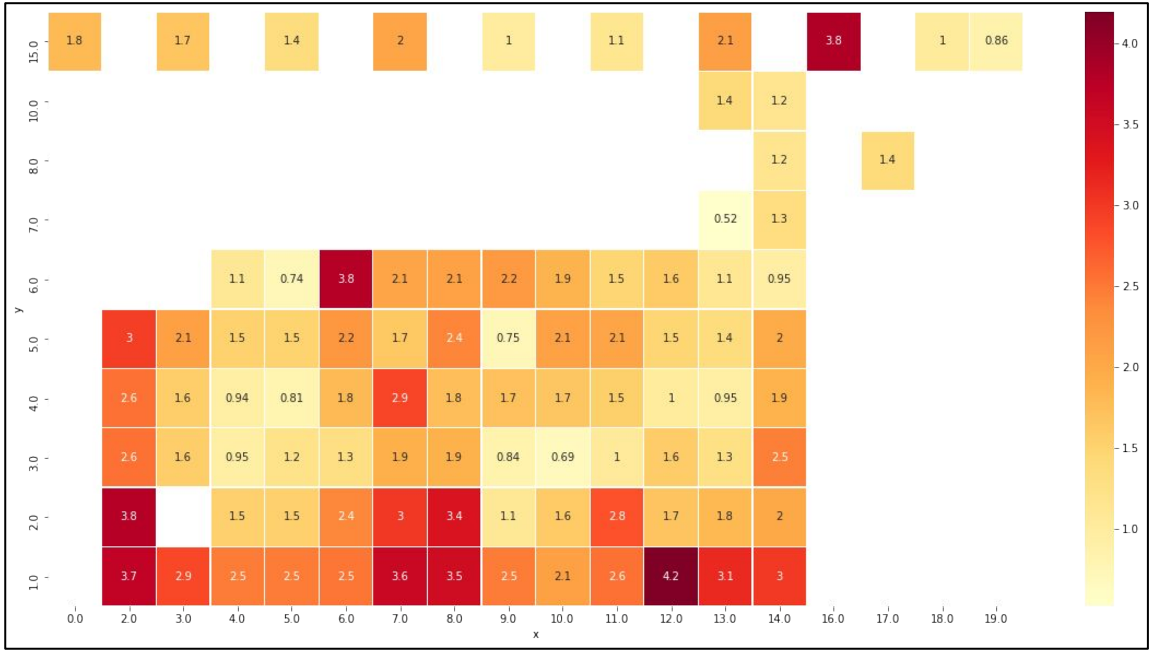
CNN Full Data Training



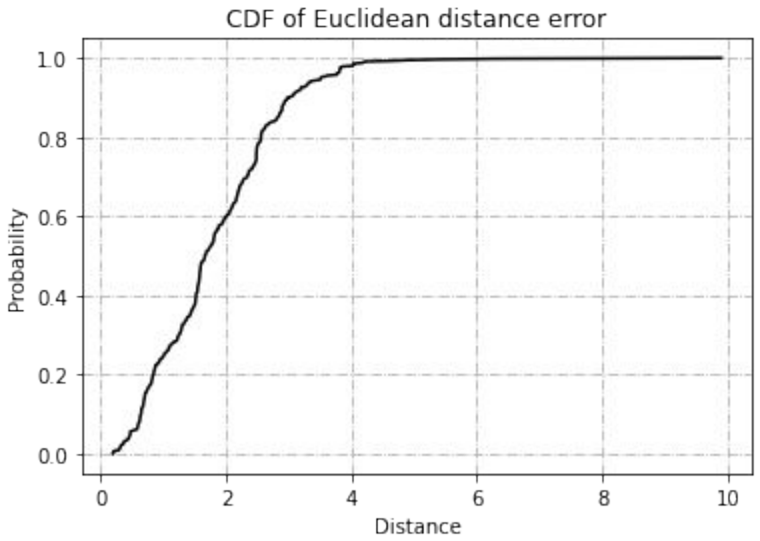
Training MSE = 2.162

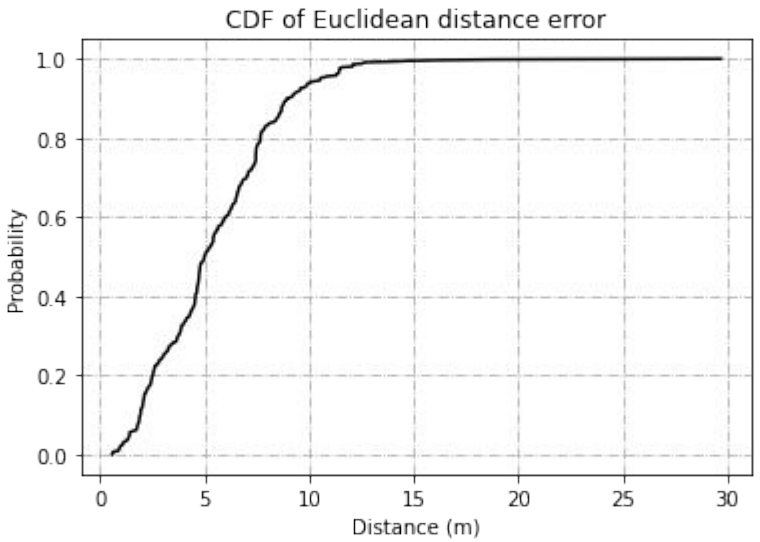
Test MSE = 2.162

CNN - Heat map Error



CNN - CDF of Euclidean Distance Error





Auto Encoder

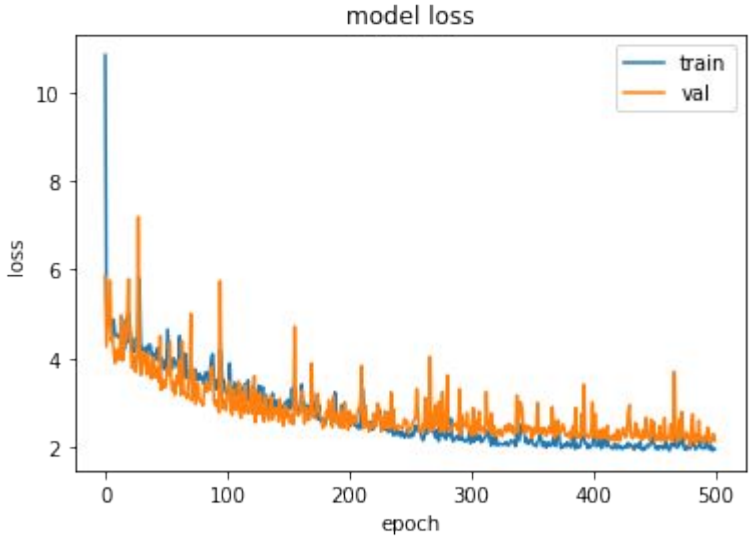
Auto Encoder

- 5000+ unlabeled instances
- Train a Neural Network (NN) with unlabeled data
- Get the encoder
- Create a new NN with encoder and train with labeled data

AE - Model

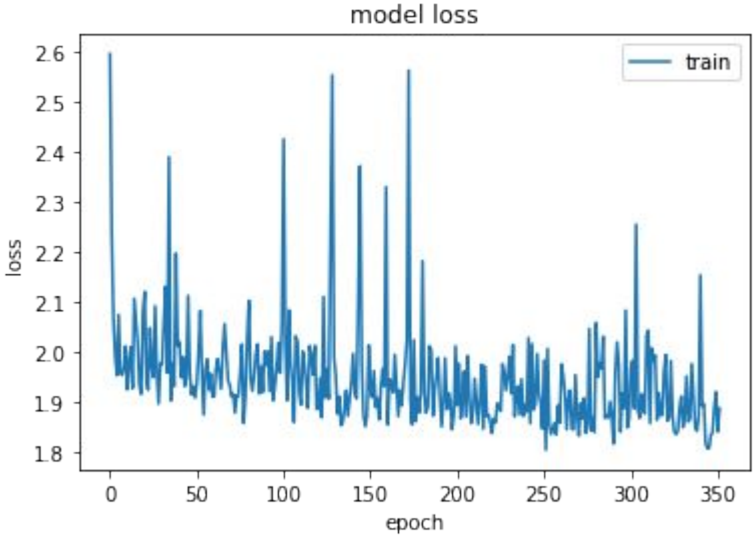
```
def create_deep():
  seed = 7
  np.random.seed(seed)
  inputs = Input(shape=(train_x_un.shape[1], train_x_un.shape[2], 1))

  # a layer instance is callable on a tensor, and returns a tensor
  x = Conv2D(24, kernel_size=(3,3), activation='relu', padding = "valid",
             data_format="channels_last")(inputs)
  x = MaxPooling2D(2)(x)
  x = Conv2D(24, kernel_size=(3,3), activation='relu', padding = "valid",
             data_format="channels_last")(x)
  x = MaxPooling2D(2)(x)
  x = Conv2D(24, kernel_size=(3,3), activation='relu', padding = "valid",
             data_format="channels_last")(x)
  x = Conv2DTranspose(24, kernel_size=(3,3),strides = (2,2), activation='relu',
                      padding = "valid", data_format="channels_last")(x)
  x = Conv2DTranspose(16, kernel_size=(3,3),strides = (2,2), activation='relu',
                      padding = "valid", data_format="channels_last")(x)
  x = Conv2DTranspose(8, kernel_size=(3,3),strides = (2,2), activation='relu',
                      padding = "valid", data_format="channels_last")(x)
  x = Conv2DTranspose(1, kernel_size=(3,3), activation='relu', padding = "valid",
                      data_format="channels_last")(x)

  # This creates a model that includes
  # the Input layer and three Dense layers
  model = Model(inputs=inputs, outputs=x)
  model.compile(optimizer=Adam(0.001),
                loss='mse')
  return model

model2 = create_deep()
model2.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=100, verbose=0,
                               mode='auto', restore_best_weights=True)

out = model2.fit(x = train_x_un, y = train_x_un,
                 validation_data = (val_x_un, val_x_un),
                 epochs=15,
                 batch_size=10,
                 #batch_size=200,
                 verbose=1,
                 callbacks = [early_stopping])
```

AE - First Training



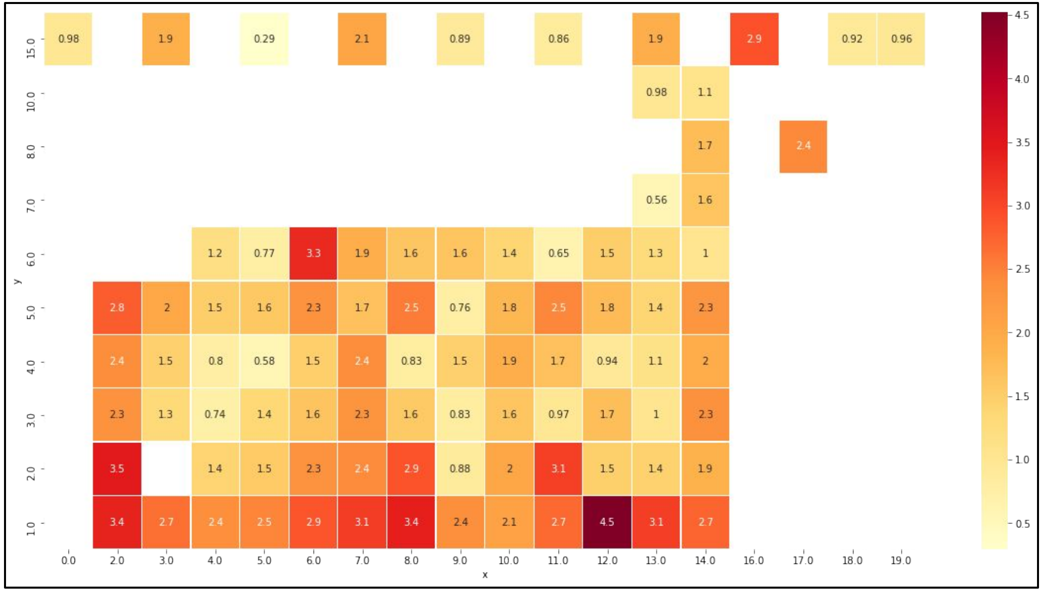
MSE = 2.144

AE - Full Data Training

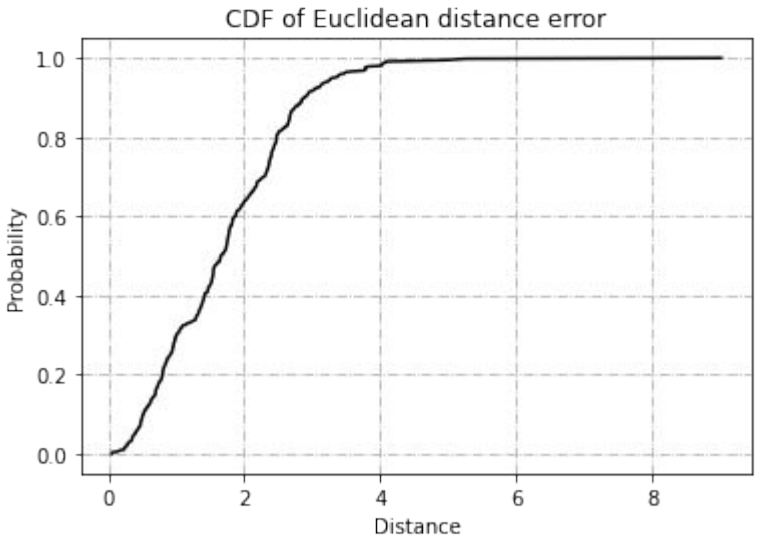


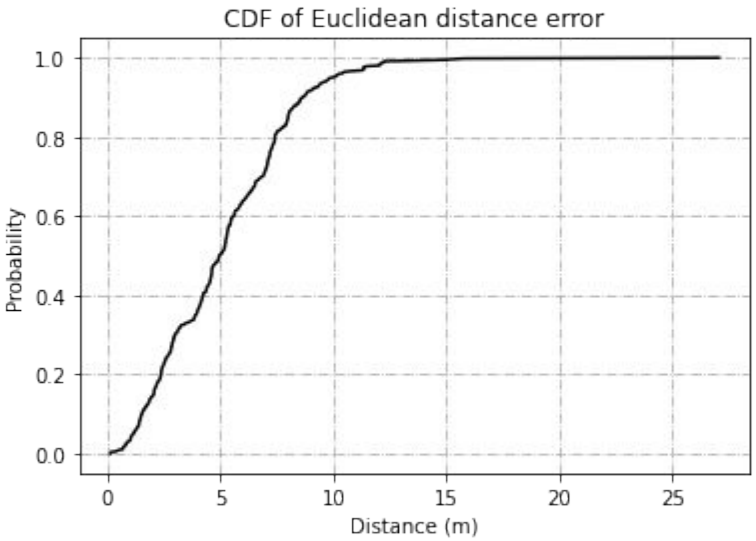
MSE = 2.034

AE - Heat map Error



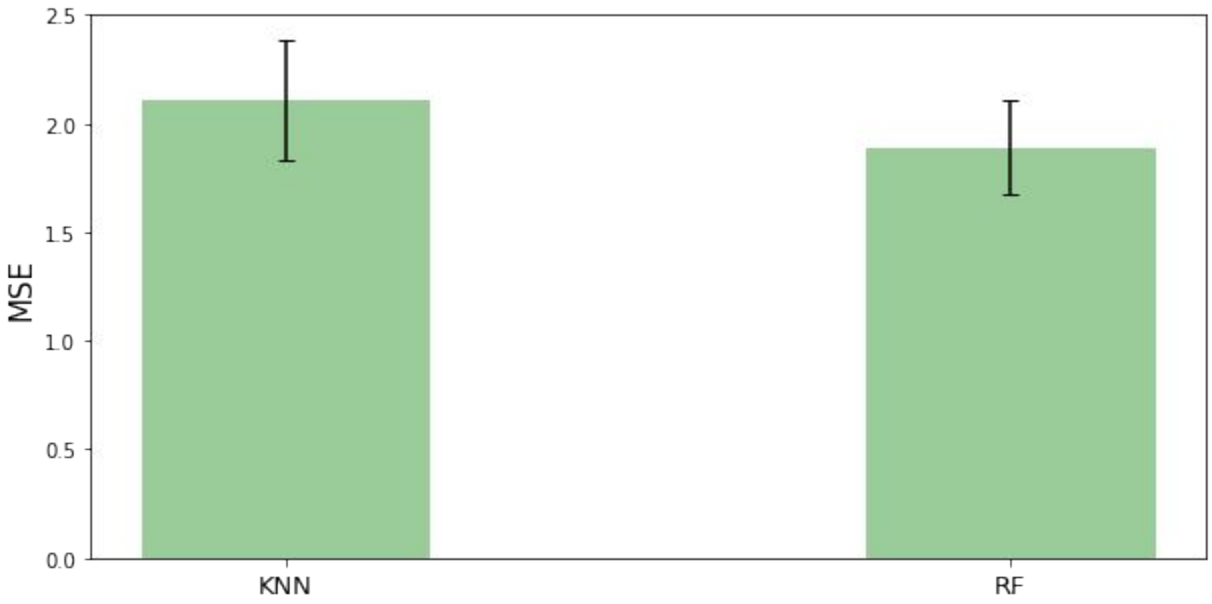
AE - CDF of Euclidean Distance Error



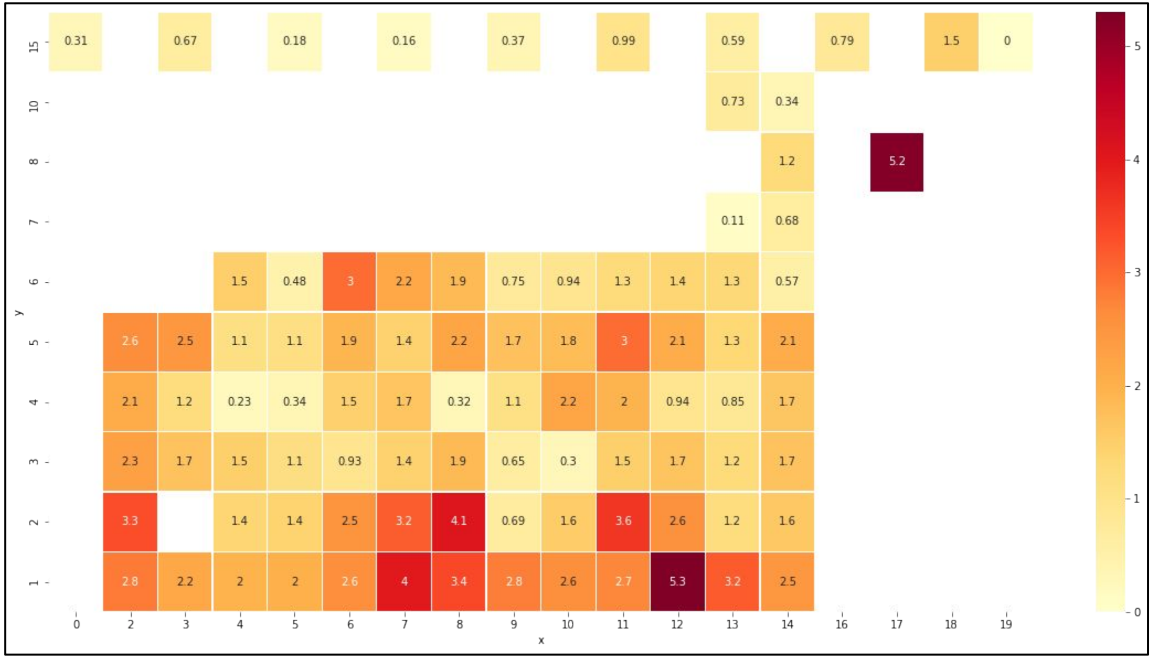


KNN and Random Forest

KNN x Random Forest - Cross Validation

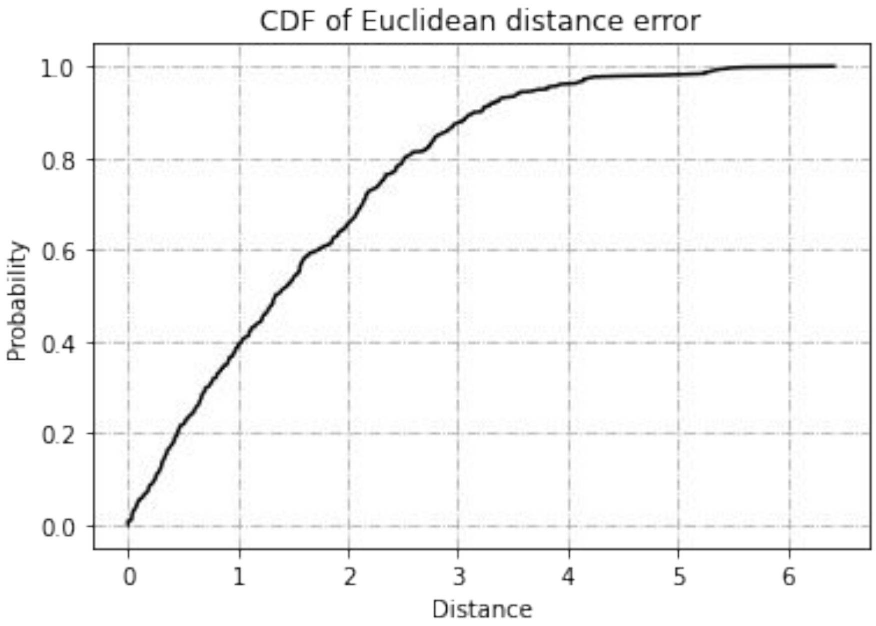


RF - Heat map



15 0.31 0.67 0.18 0.16 0.37 0.99 0.59 0.79 1.5 0
10 0.73 0.34
8 1.2 5.2
7 0.11 0.68
6 1.5 0.48 3 2.2 1.9 0.75 0.94 1.3 1.4 1.3 0.57
y
5 2.6 2.5 1.1 1.1 1.9 1.4 2.2 1.7 1.8 3 2.1 1.3 2.1
4 2.1 1.2 0.23 0.34 1.5 1.7 0.32 1.1 2.2 2 0.94 0.85 1.7
3 2.3 1.7 1.5 1.1 0.93 1.4 1.9 0.65 0.3 1.5 1.7 1.2 1.7
2 3.3 1.4 1.4 2.5 3.2 4.1 0.69 1.6 3.6 2.6 1.2 1.6
1 2.8 2.2 2 2 2.6 4 3.4 2.8 2.6 2.7 5.3 3.2 2.5
0 2 3 4 5 6 7 8 9 10 11 12 13 14 16 17 18 19
x
5
4
3
2
1
0

RF - CDF



CDF of Euclidean distance error
Probability
Distance
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6

Conclusion

- The MLP and CNN approaches reach similar results
    - Seems CNN can generalize better
- Going deeper?
- the AE approach the reaches a better loss but not so different
    - Gets a greater error at some points
- The RF reaches a similar results but with a greater range of errors
    - Doesn’t work as a classification problem
- Comparing with Kaggle
    - It’s possible to get similar results using only the training data set (820)
    - Regression and Classification approaches reaches almost similar results

Thanks for the patience!