Statistical Learning Homework 2

Mario Edoardo Pandolfo, Hejaz Nawasser, Salim Sikder

June 4, 2022

1 Part A

1.1 Importing all the libraries needed

[1]:
```
import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
```

1.2 Defining accuracy1 and accuracy2

[2]:
```
def accuracy1(y_true, y_pred):

    if len(y_pred)!=len(y_true):
        raise Exception("The two array must be of the same dimentions!")

    count = 0
    for i in range(len(y_pred)):
        if y_pred[i] <= y_true[i]*1.04 and y_pred[i] >= y_true[i]*0.96:
            count += 1

    return count/len(y_pred)


def accuracy2(y_true, y_pred):

    if len(y_pred)!=len(y_true):
        raise Exception("The two array must be of the same dimentions!")

    count = 0
    for i in range(len(y_pred)):
```

```python
        if (y_pred[i] <= y_true[i]*1.04 and y_pred[i] >= y_true[i]*0.96) or y_true[i]%y_pred[i]== 0 or y_pred[i]%y_true[i]== 0:
            count += 1

    return count/len(y_pred)
```

1.3 Loading all the train data

[3]:
```python
train_df = pd.read_csv("train4final_hw.csv")
```

1.4 Selecting at random $m = 10$ observations from the training set for Part B.

[4]:
```python
np.random.seed(1234)

chosen_idx = np.random.choice(len(train_df), size = 10)

partB_df = train_df.iloc[chosen_idx]

partB_df
```

[4]:
```
        mel1_t1     mel1_t2     mel1_t3     mel1_t4     mel1_t5     mel1_t6  \
815   275.050734  272.834307  270.534823  271.321844  265.721748  295.053231
723   282.874249  290.765179  300.027555  314.282681  311.497477  301.027809
1318  295.866554  309.168998  291.108803  272.658062  286.092493  269.386483
1077  268.325600  267.555656  267.905503  266.911166  265.841159  265.818617
1228  288.692233  286.210030  297.394528  297.734714  288.769802  290.543197
1396  273.194057  275.139660  275.090196  300.450399  290.305453  292.333299
664   208.904452  216.345940  232.873659  272.492843  295.597689  272.689490
689   297.403723  305.525745  310.613782  296.089683  295.830901  307.153993
279   281.004180  280.108439  280.115544  283.198226  274.012355  292.646752
1257  240.190807  253.744007  280.017902  267.405297  298.956521  270.333096

        mel1_t7     mel1_t8     mel1_t9    mel1_t10  …  prec.x  roughness  \
815   287.132779  278.012982  289.525987  289.342372  …   0.125   3.230718
723   303.469009  310.704548  302.823372  309.866566  …   0.125   2.082593
1318  259.063367  282.176303  262.418541  284.015301  …   0.125   1.799326
1077  266.741212  264.630055  262.436045  264.904423  …   0.125   4.807463
1228  285.969640  275.519207  296.404847  284.672493  …   0.125   1.675343
1396  285.147244  270.590157  276.359994  269.850574  …   0.125   0.727507
664   250.193753  245.604549  239.719910  237.740564  …   0.125   1.422723
689   306.130742  304.652032  304.721619  317.116978  …   0.125   2.629111
279   284.181033  275.181385  274.776309  268.744237  …   0.125   1.593020
1257  255.017715  277.225146  257.026261  293.010801  …   0.125   1.382261

          rugo     sfm.1   shannon   simpson     renyi    id  genre  tempo
815   0.175416  0.162642  0.807020  0.985819  0.682208  1159     19  128.0
```

```
723    0.174907  0.451157  0.810744  0.972550  0.576342   861      5  175.0
1318   0.175178  0.661843  0.919176  0.993205  0.800147   934     13  113.0
1077   0.179146  0.636884  0.929261  0.995097  0.852449   295     20  137.6
1228   0.171151  0.415936  0.856522  0.990450  0.745589   370      4  110.0
1396   0.173976  0.308931  0.697922  0.950689  0.482439  1697     19  134.0
664    0.175173  0.042738  0.445245  0.883417  0.344508  1312      6  180.0
689    0.176166  0.684984  0.927216  0.994066  0.821863   911      5  174.0
279    0.174331  0.414467  0.764095  0.965578  0.540060   108     19  134.0
1257   0.174572  0.471098  0.747266  0.940076  0.451193   181     19  126.0

[10 rows x 7042 columns]
```

1.5 We drop the m choosen rows in the training set.

[5]:
```
train_df = train_df.drop(chosen_idx)
```

1.6 Defining the features and the target data

We have dropped the Mel-frequency cepstrall coefficients and dominant frequency to work only with the statistics.

[6]:
```
x = train_df.iloc[:, 7011:7041].values

y = train_df.loc[:, ['tempo']].values
```

1.7 Scaling the data

[7]:
```
scaler = StandardScaler().fit(x)

x = scaler.transform(x)
```

1.8 Doing dimentionality reduction using PCA

[8]:
```
pca = PCA(.95)
pca.fit(x)
x = pca.transform(x)
```

[9]:
```
print("Number of principal components used for .95 variance:", pca.
  ↪n_components_)
```

```
Number of principal components used for .95 variance: 14
```

2 Performing SVM Regression with GridSearchCV

[10]:
```
parameters = {'kernel':('poly', 'rbf', 'sigmoid'),
              'gamma':('scale', 'auto'), 'epsilon':[0.1, 0.2]}

gs = GridSearchCV(estimator = SVR(),
                  param_grid = parameters,
                  scoring = make_scorer(accuracy2),
                  verbose = 3)

gs.fit(x, y.ravel())
print(gs.best_params_)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV 1/5] END epsilon=0.1, gamma=scale, kernel=poly;, score=0.309 total time=
0.6s
[CV 2/5] END epsilon=0.1, gamma=scale, kernel=poly;, score=0.287 total time=
0.6s
[CV 3/5] END epsilon=0.1, gamma=scale, kernel=poly;, score=0.316 total time=
0.6s
[CV 4/5] END epsilon=0.1, gamma=scale, kernel=poly;, score=0.319 total time=
0.4s
[CV 5/5] END epsilon=0.1, gamma=scale, kernel=poly;, score=0.316 total time=
0.3s
[CV 1/5] END epsilon=0.1, gamma=scale, kernel=rbf;, score=0.299 total time=
0.4s
[CV 2/5] END epsilon=0.1, gamma=scale, kernel=rbf;, score=0.252 total time=
0.4s
[CV 3/5] END epsilon=0.1, gamma=scale, kernel=rbf;, score=0.310 total time=
0.3s
[CV 4/5] END epsilon=0.1, gamma=scale, kernel=rbf;, score=0.300 total time=
0.5s
[CV 5/5] END epsilon=0.1, gamma=scale, kernel=rbf;, score=0.300 total time=
0.5s
[CV 1/5] END epsilon=0.1, gamma=scale, kernel=sigmoid;, score=0.260 total time=
0.4s
[CV 2/5] END epsilon=0.1, gamma=scale, kernel=sigmoid;, score=0.187 total time=
0.4s
[CV 3/5] END epsilon=0.1, gamma=scale, kernel=sigmoid;, score=0.265 total time=
0.4s
[CV 4/5] END epsilon=0.1, gamma=scale, kernel=sigmoid;, score=0.255 total time=
0.4s
[CV 5/5] END epsilon=0.1, gamma=scale, kernel=sigmoid;, score=0.284 total time=
0.4s
[CV 1/5] END epsilon=0.1, gamma=auto, kernel=poly;, score=0.299 total time=
0.3s
[CV 2/5] END epsilon=0.1, gamma=auto, kernel=poly;, score=0.255 total time=
```

```
0.4s
[CV 3/5] END epsilon=0.1, gamma=auto, kernel=poly;, score=0.306 total time=
0.5s
[CV 4/5] END epsilon=0.1, gamma=auto, kernel=poly;, score=0.294 total time=
0.3s
[CV 5/5] END epsilon=0.1, gamma=auto, kernel=poly;, score=0.303 total time=
0.4s
[CV 1/5] END epsilon=0.1, gamma=auto, kernel=rbf;, score=0.302 total time=
0.2s
[CV 2/5] END epsilon=0.1, gamma=auto, kernel=rbf;, score=0.242 total time=
0.3s
[CV 3/5] END epsilon=0.1, gamma=auto, kernel=rbf;, score=0.300 total time=
0.3s
[CV 4/5] END epsilon=0.1, gamma=auto, kernel=rbf;, score=0.294 total time=
0.3s
[CV 5/5] END epsilon=0.1, gamma=auto, kernel=rbf;, score=0.287 total time=
0.3s
[CV 1/5] END epsilon=0.1, gamma=auto, kernel=sigmoid;, score=0.113 total time=
0.4s
[CV 2/5] END epsilon=0.1, gamma=auto, kernel=sigmoid;, score=0.077 total time=
0.4s
[CV 3/5] END epsilon=0.1, gamma=auto, kernel=sigmoid;, score=0.116 total time=
0.4s
[CV 4/5] END epsilon=0.1, gamma=auto, kernel=sigmoid;, score=0.100 total time=
0.4s
[CV 5/5] END epsilon=0.1, gamma=auto, kernel=sigmoid;, score=0.084 total time=
0.4s
[CV 1/5] END epsilon=0.2, gamma=scale, kernel=poly;, score=0.309 total time=
0.2s
[CV 2/5] END epsilon=0.2, gamma=scale, kernel=poly;, score=0.287 total time=
0.2s
[CV 3/5] END epsilon=0.2, gamma=scale, kernel=poly;, score=0.316 total time=
0.2s
[CV 4/5] END epsilon=0.2, gamma=scale, kernel=poly;, score=0.319 total time=
0.2s
[CV 5/5] END epsilon=0.2, gamma=scale, kernel=poly;, score=0.316 total time=
0.2s
[CV 1/5] END epsilon=0.2, gamma=scale, kernel=rbf;, score=0.299 total time=
0.3s
[CV 2/5] END epsilon=0.2, gamma=scale, kernel=rbf;, score=0.252 total time=
0.2s
[CV 3/5] END epsilon=0.2, gamma=scale, kernel=rbf;, score=0.303 total time=
0.3s
[CV 4/5] END epsilon=0.2, gamma=scale, kernel=rbf;, score=0.300 total time=
0.3s
[CV 5/5] END epsilon=0.2, gamma=scale, kernel=rbf;, score=0.297 total time=
0.3s
[CV 1/5] END epsilon=0.2, gamma=scale, kernel=sigmoid;, score=0.260 total time=
```

```
0.4s
[CV 2/5] END epsilon=0.2, gamma=scale, kernel=sigmoid;, score=0.190 total time=
0.4s
[CV 3/5] END epsilon=0.2, gamma=scale, kernel=sigmoid;, score=0.265 total time=
0.4s
[CV 4/5] END epsilon=0.2, gamma=scale, kernel=sigmoid;, score=0.261 total time=
0.4s
[CV 5/5] END epsilon=0.2, gamma=scale, kernel=sigmoid;, score=0.284 total time=
0.4s
[CV 1/5] END epsilon=0.2, gamma=auto, kernel=poly;, score=0.305 total time=
0.3s
[CV 2/5] END epsilon=0.2, gamma=auto, kernel=poly;, score=0.255 total time=
0.3s
[CV 3/5] END epsilon=0.2, gamma=auto, kernel=poly;, score=0.306 total time=
0.4s
[CV 4/5] END epsilon=0.2, gamma=auto, kernel=poly;, score=0.300 total time=
0.4s
[CV 5/5] END epsilon=0.2, gamma=auto, kernel=poly;, score=0.303 total time=
0.3s
[CV 1/5] END epsilon=0.2, gamma=auto, kernel=rbf;, score=0.302 total time=
0.3s
[CV 2/5] END epsilon=0.2, gamma=auto, kernel=rbf;, score=0.242 total time=
0.3s
[CV 3/5] END epsilon=0.2, gamma=auto, kernel=rbf;, score=0.297 total time=
0.3s
[CV 4/5] END epsilon=0.2, gamma=auto, kernel=rbf;, score=0.297 total time=
0.3s
[CV 5/5] END epsilon=0.2, gamma=auto, kernel=rbf;, score=0.287 total time=
0.3s
[CV 1/5] END epsilon=0.2, gamma=auto, kernel=sigmoid;, score=0.109 total time=
0.5s
[CV 2/5] END epsilon=0.2, gamma=auto, kernel=sigmoid;, score=0.077 total time=
0.4s
[CV 3/5] END epsilon=0.2, gamma=auto, kernel=sigmoid;, score=0.113 total time=
0.4s
[CV 4/5] END epsilon=0.2, gamma=auto, kernel=sigmoid;, score=0.110 total time=
0.4s
[CV 5/5] END epsilon=0.2, gamma=auto, kernel=sigmoid;, score=0.084 total time=
0.4s
{'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'poly'}
```

2.1 Predicting on the test set

[11]:
```
test_df = pd.read_csv("test4final_hw.csv")

results = test_df.loc[:, ['id']]
```

```
x_test = test_df.iloc[:, 7011:7041].values
x_test = StandardScaler().fit_transform(x_test)
x_test = pca.transform(x_test)

results = results.assign(target = gs.best_estimator_.predict(x_test).
 ↪reshape(-1,1).ravel())
```

2.2 Saving the results as a .csv file

[12]:
```
results.to_csv('results.csv', index = False)
```

2.3 Bonus section

2.3.1 Importing needed libraries

[13]:
```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.losses import MeanSquaredLogarithmicError
```

```
2022-06-04 18:06:12.048028: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open
shared object file: No such file or directory
2022-06-04 18:06:12.048078: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
Ignore above cudart dlerror if you do not have a GPU set up on your machine.
```

2.3.2 Defining the model

[14]:
```
model = Sequential([
    Dense(pca.n_components_, kernel_initializer = 'normal', activation =␣
 ↪'relu'),
    Dropout(0.2),
    Dense(100, kernel_initializer = 'normal', activation = 'relu'),
    Dense(1, kernel_initializer='normal', activation='linear')
])
```

```
2022-06-04 18:06:15.488379: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object
file: No such file or directory
2022-06-04 18:06:15.488461: W
tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit:
UNKNOWN ERROR (303)
```

```
2022-06-04 18:06:15.488512: I
tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not
appear to be running on this host (arch-pc): /proc/driver/nvidia/version does
not exist
2022-06-04 18:06:15.489075: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

2.3.3 Training the model

[15]:
```
tf.random.set_seed(42)

msle = MeanSquaredLogarithmicError()
learning_rate = 0.01

model.compile(
    loss = msle,
    optimizer = Adam(learning_rate = learning_rate),
    metrics = [msle]
)

history = model.fit(
    x,
    y.ravel(),
    epochs=100,
    batch_size=64,
    validation_split=0.33
)
```

```
Epoch 1/100
17/17 [==============================] - 2s 53ms/step - loss: 15.3774 -
mean_squared_logarithmic_error: 14.9745 - val_loss: 5.7099 -
val_mean_squared_logarithmic_error: 5.7099
Epoch 2/100
17/17 [==============================] - 0s 16ms/step - loss: 2.7129 -
mean_squared_logarithmic_error: 2.6198 - val_loss: 0.9795 -
val_mean_squared_logarithmic_error: 0.9795
Epoch 3/100
17/17 [==============================] - 0s 17ms/step - loss: 0.5546 -
mean_squared_logarithmic_error: 0.5433 - val_loss: 0.2693 -
val_mean_squared_logarithmic_error: 0.2693
Epoch 4/100
17/17 [==============================] - 0s 15ms/step - loss: 0.2129 -
mean_squared_logarithmic_error: 0.2105 - val_loss: 0.1434 -
val_mean_squared_logarithmic_error: 0.1434
```

```
Epoch 5/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1590 -
mean_squared_logarithmic_error: 0.1554 - val_loss: 0.1173 -
val_mean_squared_logarithmic_error: 0.1173
Epoch 6/100
17/17 [==============================] - 0s 15ms/step - loss: 0.1663 -
mean_squared_logarithmic_error: 0.1651 - val_loss: 0.1093 -
val_mean_squared_logarithmic_error: 0.1093
Epoch 7/100
17/17 [==============================] - 0s 15ms/step - loss: 0.1480 -
mean_squared_logarithmic_error: 0.1485 - val_loss: 0.1047 -
val_mean_squared_logarithmic_error: 0.1047
Epoch 8/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1404 -
mean_squared_logarithmic_error: 0.1394 - val_loss: 0.1010 -
val_mean_squared_logarithmic_error: 0.1010
Epoch 9/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1469 -
mean_squared_logarithmic_error: 0.1457 - val_loss: 0.0987 -
val_mean_squared_logarithmic_error: 0.0987
Epoch 10/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1292 -
mean_squared_logarithmic_error: 0.1271 - val_loss: 0.0966 -
val_mean_squared_logarithmic_error: 0.0966
Epoch 11/100
17/17 [==============================] - 0s 28ms/step - loss: 0.1291 -
mean_squared_logarithmic_error: 0.1291 - val_loss: 0.0944 -
val_mean_squared_logarithmic_error: 0.0944
Epoch 12/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1231 -
mean_squared_logarithmic_error: 0.1245 - val_loss: 0.0921 -
val_mean_squared_logarithmic_error: 0.0921
Epoch 13/100
17/17 [==============================] - 0s 16ms/step - loss: 0.1231 -
mean_squared_logarithmic_error: 0.1238 - val_loss: 0.0909 -
val_mean_squared_logarithmic_error: 0.0909
Epoch 14/100
17/17 [==============================] - 0s 16ms/step - loss: 0.1230 -
mean_squared_logarithmic_error: 0.1211 - val_loss: 0.0895 -
val_mean_squared_logarithmic_error: 0.0895
Epoch 15/100
17/17 [==============================] - 0s 16ms/step - loss: 0.1207 -
mean_squared_logarithmic_error: 0.1204 - val_loss: 0.0873 -
val_mean_squared_logarithmic_error: 0.0873
Epoch 16/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1123 -
mean_squared_logarithmic_error: 0.1105 - val_loss: 0.0858 -
val_mean_squared_logarithmic_error: 0.0858
```

```
Epoch 17/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1120 -
mean_squared_logarithmic_error: 0.1113 - val_loss: 0.0846 -
val_mean_squared_logarithmic_error: 0.0846
Epoch 18/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1142 -
mean_squared_logarithmic_error: 0.1131 - val_loss: 0.0832 -
val_mean_squared_logarithmic_error: 0.0832
Epoch 19/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1078 -
mean_squared_logarithmic_error: 0.1049 - val_loss: 0.0822 -
val_mean_squared_logarithmic_error: 0.0822
Epoch 20/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1105 -
mean_squared_logarithmic_error: 0.1079 - val_loss: 0.0807 -
val_mean_squared_logarithmic_error: 0.0807
Epoch 21/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1200 -
mean_squared_logarithmic_error: 0.1196 - val_loss: 0.0794 -
val_mean_squared_logarithmic_error: 0.0794
Epoch 22/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1163 -
mean_squared_logarithmic_error: 0.1182 - val_loss: 0.0779 -
val_mean_squared_logarithmic_error: 0.0779
Epoch 23/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1087 -
mean_squared_logarithmic_error: 0.1078 - val_loss: 0.0778 -
val_mean_squared_logarithmic_error: 0.0778
Epoch 24/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0967 -
mean_squared_logarithmic_error: 0.0949 - val_loss: 0.0769 -
val_mean_squared_logarithmic_error: 0.0769
Epoch 25/100
17/17 [==============================] - 0s 16ms/step - loss: 0.1004 -
mean_squared_logarithmic_error: 0.1005 - val_loss: 0.0742 -
val_mean_squared_logarithmic_error: 0.0742
Epoch 26/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1003 -
mean_squared_logarithmic_error: 0.1004 - val_loss: 0.0735 -
val_mean_squared_logarithmic_error: 0.0735
Epoch 27/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0960 -
mean_squared_logarithmic_error: 0.0940 - val_loss: 0.0726 -
val_mean_squared_logarithmic_error: 0.0726
Epoch 28/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0982 -
mean_squared_logarithmic_error: 0.1000 - val_loss: 0.0721 -
val_mean_squared_logarithmic_error: 0.0721
```

```
Epoch 29/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1018 -
mean_squared_logarithmic_error: 0.1020 - val_loss: 0.0722 -
val_mean_squared_logarithmic_error: 0.0722
Epoch 30/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0921 -
mean_squared_logarithmic_error: 0.0914 - val_loss: 0.0708 -
val_mean_squared_logarithmic_error: 0.0708
Epoch 31/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0889 -
mean_squared_logarithmic_error: 0.0875 - val_loss: 0.0703 -
val_mean_squared_logarithmic_error: 0.0703
Epoch 32/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0859 -
mean_squared_logarithmic_error: 0.0864 - val_loss: 0.0697 -
val_mean_squared_logarithmic_error: 0.0697
Epoch 33/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0924 -
mean_squared_logarithmic_error: 0.0918 - val_loss: 0.0695 -
val_mean_squared_logarithmic_error: 0.0695
Epoch 34/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0913 -
mean_squared_logarithmic_error: 0.0909 - val_loss: 0.0683 -
val_mean_squared_logarithmic_error: 0.0683
Epoch 35/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0889 -
mean_squared_logarithmic_error: 0.0884 - val_loss: 0.0679 -
val_mean_squared_logarithmic_error: 0.0679
Epoch 36/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0848 -
mean_squared_logarithmic_error: 0.0867 - val_loss: 0.0671 -
val_mean_squared_logarithmic_error: 0.0671
Epoch 37/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0880 -
mean_squared_logarithmic_error: 0.0863 - val_loss: 0.0669 -
val_mean_squared_logarithmic_error: 0.0669
Epoch 38/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0857 -
mean_squared_logarithmic_error: 0.0865 - val_loss: 0.0657 -
val_mean_squared_logarithmic_error: 0.0657
Epoch 39/100
17/17 [==============================] - 0s 20ms/step - loss: 0.0833 -
mean_squared_logarithmic_error: 0.0876 - val_loss: 0.0654 -
val_mean_squared_logarithmic_error: 0.0654
Epoch 40/100
17/17 [==============================] - 0s 21ms/step - loss: 0.0841 -
mean_squared_logarithmic_error: 0.0847 - val_loss: 0.0655 -
val_mean_squared_logarithmic_error: 0.0655
```

```
Epoch 41/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0826 -
mean_squared_logarithmic_error: 0.0809 - val_loss: 0.0658 -
val_mean_squared_logarithmic_error: 0.0658
Epoch 42/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0844 -
mean_squared_logarithmic_error: 0.0826 - val_loss: 0.0643 -
val_mean_squared_logarithmic_error: 0.0643
Epoch 43/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0798 -
mean_squared_logarithmic_error: 0.0791 - val_loss: 0.0634 -
val_mean_squared_logarithmic_error: 0.0634
Epoch 44/100
17/17 [==============================] - 0s 23ms/step - loss: 0.0816 -
mean_squared_logarithmic_error: 0.0793 - val_loss: 0.0624 -
val_mean_squared_logarithmic_error: 0.0624
Epoch 45/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0806 -
mean_squared_logarithmic_error: 0.0807 - val_loss: 0.0623 -
val_mean_squared_logarithmic_error: 0.0623
Epoch 46/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0788 -
mean_squared_logarithmic_error: 0.0767 - val_loss: 0.0619 -
val_mean_squared_logarithmic_error: 0.0619
Epoch 47/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0783 -
mean_squared_logarithmic_error: 0.0764 - val_loss: 0.0617 -
val_mean_squared_logarithmic_error: 0.0617
Epoch 48/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0732 -
mean_squared_logarithmic_error: 0.0725 - val_loss: 0.0617 -
val_mean_squared_logarithmic_error: 0.0617
Epoch 49/100
17/17 [==============================] - 0s 15ms/step - loss: 0.0746 -
mean_squared_logarithmic_error: 0.0726 - val_loss: 0.0608 -
val_mean_squared_logarithmic_error: 0.0608
Epoch 50/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0726 -
mean_squared_logarithmic_error: 0.0710 - val_loss: 0.0601 -
val_mean_squared_logarithmic_error: 0.0601
Epoch 51/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0725 -
mean_squared_logarithmic_error: 0.0711 - val_loss: 0.0599 -
val_mean_squared_logarithmic_error: 0.0599
Epoch 52/100
17/17 [==============================] - 0s 15ms/step - loss: 0.0735 -
mean_squared_logarithmic_error: 0.0730 - val_loss: 0.0592 -
val_mean_squared_logarithmic_error: 0.0592
```

```
Epoch 53/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0727 -
mean_squared_logarithmic_error: 0.0721 - val_loss: 0.0588 -
val_mean_squared_logarithmic_error: 0.0588
Epoch 54/100
17/17 [==============================] - 0s 15ms/step - loss: 0.0697 -
mean_squared_logarithmic_error: 0.0693 - val_loss: 0.0580 -
val_mean_squared_logarithmic_error: 0.0580
Epoch 55/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0698 -
mean_squared_logarithmic_error: 0.0703 - val_loss: 0.0577 -
val_mean_squared_logarithmic_error: 0.0577
Epoch 56/100
17/17 [==============================] - 0s 15ms/step - loss: 0.0747 -
mean_squared_logarithmic_error: 0.0765 - val_loss: 0.0570 -
val_mean_squared_logarithmic_error: 0.0570
Epoch 57/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0678 -
mean_squared_logarithmic_error: 0.0669 - val_loss: 0.0569 -
val_mean_squared_logarithmic_error: 0.0569
Epoch 58/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0698 -
mean_squared_logarithmic_error: 0.0703 - val_loss: 0.0570 -
val_mean_squared_logarithmic_error: 0.0570
Epoch 59/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0619 -
mean_squared_logarithmic_error: 0.0613 - val_loss: 0.0563 -
val_mean_squared_logarithmic_error: 0.0563
Epoch 60/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0652 -
mean_squared_logarithmic_error: 0.0668 - val_loss: 0.0558 -
val_mean_squared_logarithmic_error: 0.0558
Epoch 61/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0655 -
mean_squared_logarithmic_error: 0.0656 - val_loss: 0.0563 -
val_mean_squared_logarithmic_error: 0.0563
Epoch 62/100
17/17 [==============================] - 0s 22ms/step - loss: 0.0643 -
mean_squared_logarithmic_error: 0.0628 - val_loss: 0.0558 -
val_mean_squared_logarithmic_error: 0.0558
Epoch 63/100
17/17 [==============================] - 0s 24ms/step - loss: 0.0635 -
mean_squared_logarithmic_error: 0.0637 - val_loss: 0.0554 -
val_mean_squared_logarithmic_error: 0.0554
Epoch 64/100
17/17 [==============================] - 0s 21ms/step - loss: 0.0665 -
mean_squared_logarithmic_error: 0.0660 - val_loss: 0.0556 -
val_mean_squared_logarithmic_error: 0.0556
```

```
Epoch 65/100
17/17 [==============================] - 0s 25ms/step - loss: 0.0632 -
mean_squared_logarithmic_error: 0.0628 - val_loss: 0.0556 -
val_mean_squared_logarithmic_error: 0.0556
Epoch 66/100
17/17 [==============================] - 0s 21ms/step - loss: 0.0650 -
mean_squared_logarithmic_error: 0.0654 - val_loss: 0.0546 -
val_mean_squared_logarithmic_error: 0.0546
Epoch 67/100
17/17 [==============================] - 0s 21ms/step - loss: 0.0611 -
mean_squared_logarithmic_error: 0.0627 - val_loss: 0.0542 -
val_mean_squared_logarithmic_error: 0.0542
Epoch 68/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0621 -
mean_squared_logarithmic_error: 0.0610 - val_loss: 0.0550 -
val_mean_squared_logarithmic_error: 0.0550
Epoch 69/100
17/17 [==============================] - 0s 20ms/step - loss: 0.0630 -
mean_squared_logarithmic_error: 0.0630 - val_loss: 0.0546 -
val_mean_squared_logarithmic_error: 0.0546
Epoch 70/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0599 -
mean_squared_logarithmic_error: 0.0613 - val_loss: 0.0546 -
val_mean_squared_logarithmic_error: 0.0546
Epoch 71/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0587 -
mean_squared_logarithmic_error: 0.0615 - val_loss: 0.0537 -
val_mean_squared_logarithmic_error: 0.0537
Epoch 72/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0610 -
mean_squared_logarithmic_error: 0.0608 - val_loss: 0.0548 -
val_mean_squared_logarithmic_error: 0.0548
Epoch 73/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0586 -
mean_squared_logarithmic_error: 0.0592 - val_loss: 0.0535 -
val_mean_squared_logarithmic_error: 0.0535
Epoch 74/100
17/17 [==============================] - 0s 22ms/step - loss: 0.0600 -
mean_squared_logarithmic_error: 0.0606 - val_loss: 0.0535 -
val_mean_squared_logarithmic_error: 0.0535
Epoch 75/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0614 -
mean_squared_logarithmic_error: 0.0639 - val_loss: 0.0543 -
val_mean_squared_logarithmic_error: 0.0543
Epoch 76/100
17/17 [==============================] - 0s 20ms/step - loss: 0.0596 -
mean_squared_logarithmic_error: 0.0583 - val_loss: 0.0547 -
val_mean_squared_logarithmic_error: 0.0547
```

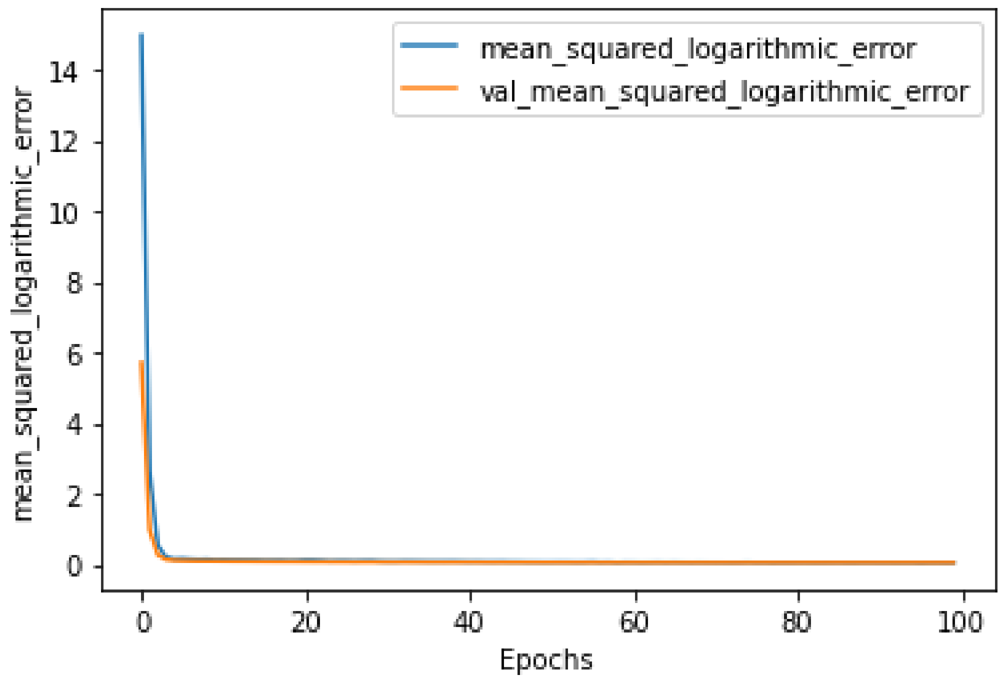
```
Epoch 77/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0610 -
mean_squared_logarithmic_error: 0.0599 - val_loss: 0.0547 -
val_mean_squared_logarithmic_error: 0.0547
Epoch 78/100
17/17 [==============================] - 0s 20ms/step - loss: 0.0556 -
mean_squared_logarithmic_error: 0.0549 - val_loss: 0.0540 -
val_mean_squared_logarithmic_error: 0.0540
Epoch 79/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0587 -
mean_squared_logarithmic_error: 0.0579 - val_loss: 0.0536 -
val_mean_squared_logarithmic_error: 0.0536
Epoch 80/100
17/17 [==============================] - 0s 20ms/step - loss: 0.0584 -
mean_squared_logarithmic_error: 0.0567 - val_loss: 0.0535 -
val_mean_squared_logarithmic_error: 0.0535
Epoch 81/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0589 -
mean_squared_logarithmic_error: 0.0590 - val_loss: 0.0536 -
val_mean_squared_logarithmic_error: 0.0536
Epoch 82/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0578 -
mean_squared_logarithmic_error: 0.0584 - val_loss: 0.0540 -
val_mean_squared_logarithmic_error: 0.0540
Epoch 83/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0560 -
mean_squared_logarithmic_error: 0.0564 - val_loss: 0.0543 -
val_mean_squared_logarithmic_error: 0.0543
Epoch 84/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0577 -
mean_squared_logarithmic_error: 0.0570 - val_loss: 0.0535 -
val_mean_squared_logarithmic_error: 0.0535
Epoch 85/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0568 -
mean_squared_logarithmic_error: 0.0585 - val_loss: 0.0534 -
val_mean_squared_logarithmic_error: 0.0534
Epoch 86/100
17/17 [==============================] - 0s 15ms/step - loss: 0.0586 -
mean_squared_logarithmic_error: 0.0590 - val_loss: 0.0541 -
val_mean_squared_logarithmic_error: 0.0541
Epoch 87/100
17/17 [==============================] - 0s 20ms/step - loss: 0.0552 -
mean_squared_logarithmic_error: 0.0547 - val_loss: 0.0533 -
val_mean_squared_logarithmic_error: 0.0533
Epoch 88/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0557 -
mean_squared_logarithmic_error: 0.0545 - val_loss: 0.0536 -
val_mean_squared_logarithmic_error: 0.0536
```

```
Epoch 89/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0590 -
mean_squared_logarithmic_error: 0.0578 - val_loss: 0.0535 -
val_mean_squared_logarithmic_error: 0.0535
Epoch 90/100
17/17 [==============================] - 0s 21ms/step - loss: 0.0564 -
mean_squared_logarithmic_error: 0.0570 - val_loss: 0.0535 -
val_mean_squared_logarithmic_error: 0.0535
Epoch 91/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0566 -
mean_squared_logarithmic_error: 0.0555 - val_loss: 0.0536 -
val_mean_squared_logarithmic_error: 0.0536
Epoch 92/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0541 -
mean_squared_logarithmic_error: 0.0554 - val_loss: 0.0534 -
val_mean_squared_logarithmic_error: 0.0534
Epoch 93/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0524 -
mean_squared_logarithmic_error: 0.0541 - val_loss: 0.0534 -
val_mean_squared_logarithmic_error: 0.0534
Epoch 94/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0544 -
mean_squared_logarithmic_error: 0.0547 - val_loss: 0.0540 -
val_mean_squared_logarithmic_error: 0.0540
Epoch 95/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0535 -
mean_squared_logarithmic_error: 0.0547 - val_loss: 0.0537 -
val_mean_squared_logarithmic_error: 0.0537
Epoch 96/100
17/17 [==============================] - 0s 18ms/step - loss: 0.0498 -
mean_squared_logarithmic_error: 0.0480 - val_loss: 0.0537 -
val_mean_squared_logarithmic_error: 0.0537
Epoch 97/100
17/17 [==============================] - 0s 20ms/step - loss: 0.0512 -
mean_squared_logarithmic_error: 0.0526 - val_loss: 0.0530 -
val_mean_squared_logarithmic_error: 0.0530
Epoch 98/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0528 -
mean_squared_logarithmic_error: 0.0537 - val_loss: 0.0526 -
val_mean_squared_logarithmic_error: 0.0526
Epoch 99/100
17/17 [==============================] - 0s 15ms/step - loss: 0.0506 -
mean_squared_logarithmic_error: 0.0502 - val_loss: 0.0528 -
val_mean_squared_logarithmic_error: 0.0528
Epoch 100/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0517 -
mean_squared_logarithmic_error: 0.0519 - val_loss: 0.0533 -
val_mean_squared_logarithmic_error: 0.0533
```

2.3.4 Plotting the history

[16]:
```
def plot_history(history, key):
  plt.plot(history.history[key])
  plt.plot(history.history['val_'+key])
  plt.xlabel("Epochs")
  plt.ylabel(key)
  plt.legend([key, 'val_'+key])
  plt.show()

plot_history(history, 'mean_squared_logarithmic_error')
```



2.3.5 Predicting on the test set

[17]:
```
results_bonus = test_df.loc[:, ['id']]

results_bonus = results_bonus.assign(target = model.predict(x_test))
```

```
21/21 [==============================] - 0s 3ms/step
```

2.3.6 Saving the results of the bonus section

[18]:
```
results.to_csv('results_bonus.csv', index = False)
```

3 Part B

3.1 Importing some libraries

[19]:
```
from math import ceil
```

3.2 Preparing the data for part B

[20]:
```
x_B = partB_df.iloc[:, 7011:7041].values

y_B = partB_df.loc[:, ['tempo']].values
```

3.2.1 Scaling the data

[21]:
```
scalerB = StandardScaler().fit(x_B)
x_B = scalerB.transform(x_B)
```

3.2.2 Performing dimensionality reduction using PCA

[22]:
```
x_B = pca.transform(x_B)
```

3.3 Implement the *Split Conformal Prediction for Regression* algorithm

3.3.1 Prepraring the $D^{(1)}$ and $D^{(2)}$ dataset

Redefining x and y for doing proper data preprocessing

[23]:
```
x = train_df.iloc[:, 7011:7041].values
y = train_df.loc[:, ['tempo']].values
```

Splitting into training and calibration set

[24]:
```
x_d1, x_d2, y_d1, y_d2 = train_test_split(x, y, test_size=0.5, random_state=42)
```

Scaling the data

[25]:
```
scalerd1 = StandardScaler().fit(x_d1)
scalerd2 = StandardScaler().fit(x_d2)

x_d1 = scalerd1.transform(x_d1)
x_d2 = scalerd2.transform(x_d2)
```

Performing dimensionality reduction

[26]:
```
x_d1 = pca.transform(x_d1)
x_d2 = pca.transform(x_d2)
```

3.3.2 Training on $D^{(1)}$ the ANN

[27]:
```
modelB = Sequential([
    Dense(pca.n_components_, kernel_initializer = 'normal', activation =␣
 ↪'relu'),
    Dropout(0.2),
    Dense(100, kernel_initializer = 'normal', activation = 'relu'),
    Dense(1, kernel_initializer='normal', activation='linear')
])

tf.random.set_seed(42)

msle = MeanSquaredLogarithmicError()
learning_rate = 0.01

modelB.compile(
    loss = msle,
    optimizer = Adam(learning_rate = learning_rate),
    metrics = [msle]
)

historyB = modelB.fit(
    x_d1,
    y_d1.ravel(),
    epochs=100,
    batch_size=64,
    validation_split=0.33
)
```

```
Epoch 1/100
9/9 [==============================] - 2s 100ms/step - loss: 20.5479 -
mean_squared_logarithmic_error: 20.1520 - val_loss: 13.4785 -
val_mean_squared_logarithmic_error: 13.4785
Epoch 2/100
9/9 [==============================] - 0s 36ms/step - loss: 9.5990 -
mean_squared_logarithmic_error: 9.1679 - val_loss: 5.2542 -
val_mean_squared_logarithmic_error: 5.2542
Epoch 3/100
9/9 [==============================] - 0s 30ms/step - loss: 3.6852 -
mean_squared_logarithmic_error: 3.4931 - val_loss: 2.0703 -
val_mean_squared_logarithmic_error: 2.0703
Epoch 4/100
9/9 [==============================] - 0s 30ms/step - loss: 1.4201 -
mean_squared_logarithmic_error: 1.3515 - val_loss: 0.9033 -
val_mean_squared_logarithmic_error: 0.9033
Epoch 5/100
9/9 [==============================] - 0s 39ms/step - loss: 0.6646 -
mean_squared_logarithmic_error: 0.6721 - val_loss: 0.4532 -
```

```
val_mean_squared_logarithmic_error: 0.4532
Epoch 6/100
9/9 [==============================] - 0s 42ms/step - loss: 0.3722 -
mean_squared_logarithmic_error: 0.3480 - val_loss: 0.2661 -
val_mean_squared_logarithmic_error: 0.2661
Epoch 7/100
9/9 [==============================] - 0s 38ms/step - loss: 0.2206 -
mean_squared_logarithmic_error: 0.2227 - val_loss: 0.1855 -
val_mean_squared_logarithmic_error: 0.1855
Epoch 8/100
9/9 [==============================] - 0s 31ms/step - loss: 0.1800 -
mean_squared_logarithmic_error: 0.1761 - val_loss: 0.1479 -
val_mean_squared_logarithmic_error: 0.1479
Epoch 9/100
9/9 [==============================] - 0s 50ms/step - loss: 0.1642 -
mean_squared_logarithmic_error: 0.1576 - val_loss: 0.1287 -
val_mean_squared_logarithmic_error: 0.1287
Epoch 10/100
9/9 [==============================] - 0s 34ms/step - loss: 0.1466 -
mean_squared_logarithmic_error: 0.1373 - val_loss: 0.1196 -
val_mean_squared_logarithmic_error: 0.1196
Epoch 11/100
9/9 [==============================] - 0s 42ms/step - loss: 0.1332 -
mean_squared_logarithmic_error: 0.1242 - val_loss: 0.1144 -
val_mean_squared_logarithmic_error: 0.1144
Epoch 12/100
9/9 [==============================] - 0s 38ms/step - loss: 0.1352 -
mean_squared_logarithmic_error: 0.1315 - val_loss: 0.1107 -
val_mean_squared_logarithmic_error: 0.1107
Epoch 13/100
9/9 [==============================] - 0s 33ms/step - loss: 0.1159 -
mean_squared_logarithmic_error: 0.1154 - val_loss: 0.1072 -
val_mean_squared_logarithmic_error: 0.1072
Epoch 14/100
9/9 [==============================] - 0s 33ms/step - loss: 0.1307 -
mean_squared_logarithmic_error: 0.1230 - val_loss: 0.1052 -
val_mean_squared_logarithmic_error: 0.1052
Epoch 15/100
9/9 [==============================] - 0s 23ms/step - loss: 0.1216 -
mean_squared_logarithmic_error: 0.1275 - val_loss: 0.1035 -
val_mean_squared_logarithmic_error: 0.1035
Epoch 16/100
9/9 [==============================] - 0s 24ms/step - loss: 0.1187 -
mean_squared_logarithmic_error: 0.1169 - val_loss: 0.1016 -
val_mean_squared_logarithmic_error: 0.1016
Epoch 17/100
9/9 [==============================] - 0s 25ms/step - loss: 0.1203 -
mean_squared_logarithmic_error: 0.1267 - val_loss: 0.1000 -
```

```
val_mean_squared_logarithmic_error: 0.1000
Epoch 18/100
9/9 [==============================] - 0s 26ms/step - loss: 0.1146 -
mean_squared_logarithmic_error: 0.1124 - val_loss: 0.0985 -
val_mean_squared_logarithmic_error: 0.0985
Epoch 19/100
9/9 [==============================] - 0s 23ms/step - loss: 0.1366 -
mean_squared_logarithmic_error: 0.1274 - val_loss: 0.0966 -
val_mean_squared_logarithmic_error: 0.0966
Epoch 20/100
9/9 [==============================] - 0s 24ms/step - loss: 0.1135 -
mean_squared_logarithmic_error: 0.1050 - val_loss: 0.0940 -
val_mean_squared_logarithmic_error: 0.0940
Epoch 21/100
9/9 [==============================] - 0s 26ms/step - loss: 0.1218 -
mean_squared_logarithmic_error: 0.1154 - val_loss: 0.0924 -
val_mean_squared_logarithmic_error: 0.0924
Epoch 22/100
9/9 [==============================] - 0s 25ms/step - loss: 0.1128 -
mean_squared_logarithmic_error: 0.1186 - val_loss: 0.0916 -
val_mean_squared_logarithmic_error: 0.0916
Epoch 23/100
9/9 [==============================] - 0s 27ms/step - loss: 0.1199 -
mean_squared_logarithmic_error: 0.1228 - val_loss: 0.0910 -
val_mean_squared_logarithmic_error: 0.0910
Epoch 24/100
9/9 [==============================] - 0s 24ms/step - loss: 0.1076 -
mean_squared_logarithmic_error: 0.1143 - val_loss: 0.0902 -
val_mean_squared_logarithmic_error: 0.0902
Epoch 25/100
9/9 [==============================] - 0s 22ms/step - loss: 0.1124 -
mean_squared_logarithmic_error: 0.1043 - val_loss: 0.0883 -
val_mean_squared_logarithmic_error: 0.0883
Epoch 26/100
9/9 [==============================] - 0s 24ms/step - loss: 0.1083 -
mean_squared_logarithmic_error: 0.1073 - val_loss: 0.0873 -
val_mean_squared_logarithmic_error: 0.0873
Epoch 27/100
9/9 [==============================] - 0s 24ms/step - loss: 0.1128 -
mean_squared_logarithmic_error: 0.1060 - val_loss: 0.0866 -
val_mean_squared_logarithmic_error: 0.0866
Epoch 28/100
9/9 [==============================] - 0s 23ms/step - loss: 0.1156 -
mean_squared_logarithmic_error: 0.1112 - val_loss: 0.0860 -
val_mean_squared_logarithmic_error: 0.0860
Epoch 29/100
9/9 [==============================] - 0s 22ms/step - loss: 0.1240 -
mean_squared_logarithmic_error: 0.1145 - val_loss: 0.0856 -
```

```
val_mean_squared_logarithmic_error: 0.0856
Epoch 30/100
9/9 [==============================] - 0s 22ms/step - loss: 0.1081 -
mean_squared_logarithmic_error: 0.1085 - val_loss: 0.0846 -
val_mean_squared_logarithmic_error: 0.0846
Epoch 31/100
9/9 [==============================] - 0s 22ms/step - loss: 0.1087 -
mean_squared_logarithmic_error: 0.1018 - val_loss: 0.0847 -
val_mean_squared_logarithmic_error: 0.0847
Epoch 32/100
9/9 [==============================] - 0s 23ms/step - loss: 0.1193 -
mean_squared_logarithmic_error: 0.1229 - val_loss: 0.0848 -
val_mean_squared_logarithmic_error: 0.0848
Epoch 33/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0920 -
mean_squared_logarithmic_error: 0.0925 - val_loss: 0.0847 -
val_mean_squared_logarithmic_error: 0.0847
Epoch 34/100
9/9 [==============================] - 0s 20ms/step - loss: 0.1065 -
mean_squared_logarithmic_error: 0.1098 - val_loss: 0.0838 -
val_mean_squared_logarithmic_error: 0.0838
Epoch 35/100
9/9 [==============================] - 0s 21ms/step - loss: 0.0996 -
mean_squared_logarithmic_error: 0.0927 - val_loss: 0.0825 -
val_mean_squared_logarithmic_error: 0.0825
Epoch 36/100
9/9 [==============================] - 0s 23ms/step - loss: 0.1117 -
mean_squared_logarithmic_error: 0.1066 - val_loss: 0.0813 -
val_mean_squared_logarithmic_error: 0.0813
Epoch 37/100
9/9 [==============================] - 0s 24ms/step - loss: 0.1099 -
mean_squared_logarithmic_error: 0.1281 - val_loss: 0.0799 -
val_mean_squared_logarithmic_error: 0.0799
Epoch 38/100
9/9 [==============================] - 0s 26ms/step - loss: 0.1013 -
mean_squared_logarithmic_error: 0.1014 - val_loss: 0.0797 -
val_mean_squared_logarithmic_error: 0.0797
Epoch 39/100
9/9 [==============================] - 0s 25ms/step - loss: 0.1082 -
mean_squared_logarithmic_error: 0.1040 - val_loss: 0.0797 -
val_mean_squared_logarithmic_error: 0.0797
Epoch 40/100
9/9 [==============================] - 0s 27ms/step - loss: 0.1034 -
mean_squared_logarithmic_error: 0.1090 - val_loss: 0.0785 -
val_mean_squared_logarithmic_error: 0.0785
Epoch 41/100
9/9 [==============================] - 0s 28ms/step - loss: 0.0909 -
mean_squared_logarithmic_error: 0.0889 - val_loss: 0.0781 -
```

```
val_mean_squared_logarithmic_error: 0.0781
Epoch 42/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0952 -
mean_squared_logarithmic_error: 0.0880 - val_loss: 0.0781 -
val_mean_squared_logarithmic_error: 0.0781
Epoch 43/100
9/9 [==============================] - 0s 22ms/step - loss: 0.1034 -
mean_squared_logarithmic_error: 0.0956 - val_loss: 0.0775 -
val_mean_squared_logarithmic_error: 0.0775
Epoch 44/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0989 -
mean_squared_logarithmic_error: 0.1020 - val_loss: 0.0767 -
val_mean_squared_logarithmic_error: 0.0767
Epoch 45/100
9/9 [==============================] - 0s 24ms/step - loss: 0.1045 -
mean_squared_logarithmic_error: 0.1011 - val_loss: 0.0779 -
val_mean_squared_logarithmic_error: 0.0779
Epoch 46/100
9/9 [==============================] - 0s 24ms/step - loss: 0.0988 -
mean_squared_logarithmic_error: 0.0979 - val_loss: 0.0778 -
val_mean_squared_logarithmic_error: 0.0778
Epoch 47/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0963 -
mean_squared_logarithmic_error: 0.0982 - val_loss: 0.0770 -
val_mean_squared_logarithmic_error: 0.0770
Epoch 48/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0953 -
mean_squared_logarithmic_error: 0.0959 - val_loss: 0.0759 -
val_mean_squared_logarithmic_error: 0.0759
Epoch 49/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0961 -
mean_squared_logarithmic_error: 0.1076 - val_loss: 0.0749 -
val_mean_squared_logarithmic_error: 0.0749
Epoch 50/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0890 -
mean_squared_logarithmic_error: 0.0840 - val_loss: 0.0765 -
val_mean_squared_logarithmic_error: 0.0765
Epoch 51/100
9/9 [==============================] - 0s 20ms/step - loss: 0.1007 -
mean_squared_logarithmic_error: 0.1069 - val_loss: 0.0763 -
val_mean_squared_logarithmic_error: 0.0763
Epoch 52/100
9/9 [==============================] - 0s 21ms/step - loss: 0.1000 -
mean_squared_logarithmic_error: 0.0985 - val_loss: 0.0748 -
val_mean_squared_logarithmic_error: 0.0748
Epoch 53/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0828 -
mean_squared_logarithmic_error: 0.0907 - val_loss: 0.0740 -
```

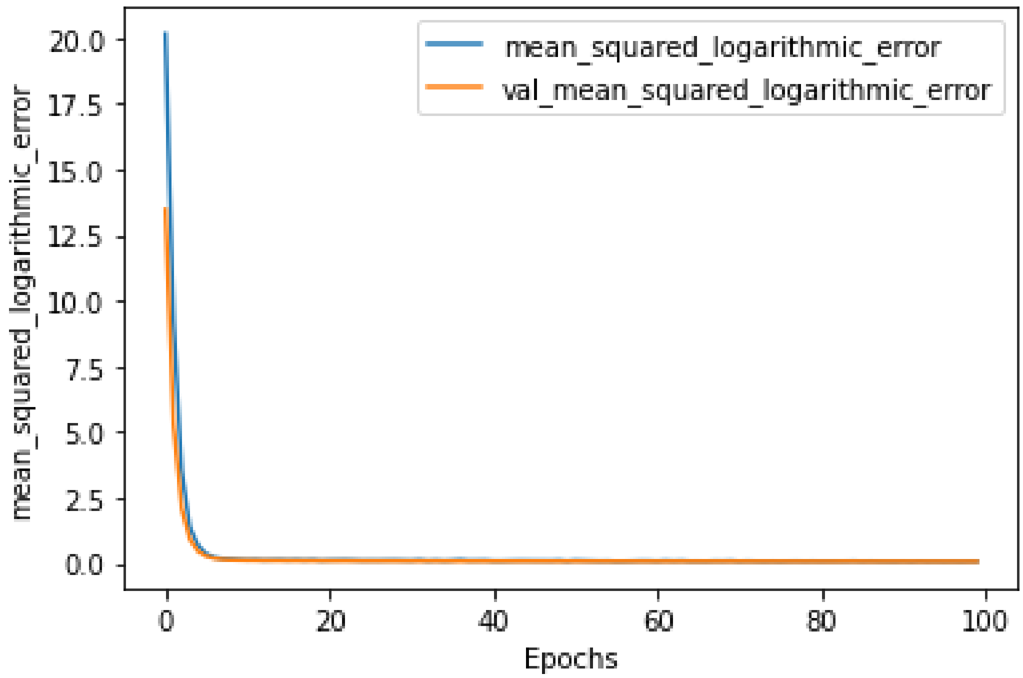
```
val_mean_squared_logarithmic_error: 0.0740
Epoch 54/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0870 -
mean_squared_logarithmic_error: 0.0838 - val_loss: 0.0743 -
val_mean_squared_logarithmic_error: 0.0743
Epoch 55/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0895 -
mean_squared_logarithmic_error: 0.0870 - val_loss: 0.0732 -
val_mean_squared_logarithmic_error: 0.0732
Epoch 56/100
9/9 [==============================] - 0s 24ms/step - loss: 0.0776 -
mean_squared_logarithmic_error: 0.0731 - val_loss: 0.0730 -
val_mean_squared_logarithmic_error: 0.0730
Epoch 57/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0804 -
mean_squared_logarithmic_error: 0.0778 - val_loss: 0.0723 -
val_mean_squared_logarithmic_error: 0.0723
Epoch 58/100
9/9 [==============================] - 0s 27ms/step - loss: 0.0840 -
mean_squared_logarithmic_error: 0.0841 - val_loss: 0.0730 -
val_mean_squared_logarithmic_error: 0.0730
Epoch 59/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0774 -
mean_squared_logarithmic_error: 0.0932 - val_loss: 0.0725 -
val_mean_squared_logarithmic_error: 0.0725
Epoch 60/100
9/9 [==============================] - 0s 24ms/step - loss: 0.0932 -
mean_squared_logarithmic_error: 0.0911 - val_loss: 0.0715 -
val_mean_squared_logarithmic_error: 0.0715
Epoch 61/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0861 -
mean_squared_logarithmic_error: 0.0807 - val_loss: 0.0708 -
val_mean_squared_logarithmic_error: 0.0708
Epoch 62/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0884 -
mean_squared_logarithmic_error: 0.0999 - val_loss: 0.0709 -
val_mean_squared_logarithmic_error: 0.0709
Epoch 63/100
9/9 [==============================] - 0s 24ms/step - loss: 0.0862 -
mean_squared_logarithmic_error: 0.0854 - val_loss: 0.0721 -
val_mean_squared_logarithmic_error: 0.0721
Epoch 64/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0817 -
mean_squared_logarithmic_error: 0.0959 - val_loss: 0.0724 -
val_mean_squared_logarithmic_error: 0.0724
Epoch 65/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0820 -
mean_squared_logarithmic_error: 0.0776 - val_loss: 0.0710 -
```

```
val_mean_squared_logarithmic_error: 0.0710
Epoch 66/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0834 -
mean_squared_logarithmic_error: 0.0787 - val_loss: 0.0706 -
val_mean_squared_logarithmic_error: 0.0706
Epoch 67/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0776 -
mean_squared_logarithmic_error: 0.0754 - val_loss: 0.0701 -
val_mean_squared_logarithmic_error: 0.0701
Epoch 68/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0927 -
mean_squared_logarithmic_error: 0.0877 - val_loss: 0.0706 -
val_mean_squared_logarithmic_error: 0.0706
Epoch 69/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0839 -
mean_squared_logarithmic_error: 0.0841 - val_loss: 0.0717 -
val_mean_squared_logarithmic_error: 0.0717
Epoch 70/100
9/9 [==============================] - 0s 21ms/step - loss: 0.0838 -
mean_squared_logarithmic_error: 0.0948 - val_loss: 0.0713 -
val_mean_squared_logarithmic_error: 0.0713
Epoch 71/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0772 -
mean_squared_logarithmic_error: 0.0739 - val_loss: 0.0703 -
val_mean_squared_logarithmic_error: 0.0703
Epoch 72/100
9/9 [==============================] - 0s 27ms/step - loss: 0.0882 -
mean_squared_logarithmic_error: 0.0843 - val_loss: 0.0698 -
val_mean_squared_logarithmic_error: 0.0698
Epoch 73/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0759 -
mean_squared_logarithmic_error: 0.0774 - val_loss: 0.0690 -
val_mean_squared_logarithmic_error: 0.0690
Epoch 74/100
9/9 [==============================] - 0s 26ms/step - loss: 0.0718 -
mean_squared_logarithmic_error: 0.0704 - val_loss: 0.0685 -
val_mean_squared_logarithmic_error: 0.0685
Epoch 75/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0789 -
mean_squared_logarithmic_error: 0.0786 - val_loss: 0.0682 -
val_mean_squared_logarithmic_error: 0.0682
Epoch 76/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0705 -
mean_squared_logarithmic_error: 0.0677 - val_loss: 0.0676 -
val_mean_squared_logarithmic_error: 0.0676
Epoch 77/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0824 -
mean_squared_logarithmic_error: 0.0775 - val_loss: 0.0674 -
```

```
val_mean_squared_logarithmic_error: 0.0674
Epoch 78/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0794 -
mean_squared_logarithmic_error: 0.0807 - val_loss: 0.0671 -
val_mean_squared_logarithmic_error: 0.0671
Epoch 79/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0767 -
mean_squared_logarithmic_error: 0.0760 - val_loss: 0.0665 -
val_mean_squared_logarithmic_error: 0.0665
Epoch 80/100
9/9 [==============================] - 0s 24ms/step - loss: 0.0739 -
mean_squared_logarithmic_error: 0.0798 - val_loss: 0.0661 -
val_mean_squared_logarithmic_error: 0.0661
Epoch 81/100
9/9 [==============================] - 0s 28ms/step - loss: 0.0768 -
mean_squared_logarithmic_error: 0.0734 - val_loss: 0.0646 -
val_mean_squared_logarithmic_error: 0.0646
Epoch 82/100
9/9 [==============================] - 0s 22ms/step - loss: 0.0722 -
mean_squared_logarithmic_error: 0.0768 - val_loss: 0.0642 -
val_mean_squared_logarithmic_error: 0.0642
Epoch 83/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0802 -
mean_squared_logarithmic_error: 0.0778 - val_loss: 0.0665 -
val_mean_squared_logarithmic_error: 0.0665
Epoch 84/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0774 -
mean_squared_logarithmic_error: 0.0776 - val_loss: 0.0696 -
val_mean_squared_logarithmic_error: 0.0696
Epoch 85/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0782 -
mean_squared_logarithmic_error: 0.0833 - val_loss: 0.0697 -
val_mean_squared_logarithmic_error: 0.0697
Epoch 86/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0731 -
mean_squared_logarithmic_error: 0.0726 - val_loss: 0.0662 -
val_mean_squared_logarithmic_error: 0.0662
Epoch 87/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0775 -
mean_squared_logarithmic_error: 0.0795 - val_loss: 0.0646 -
val_mean_squared_logarithmic_error: 0.0646
Epoch 88/100
9/9 [==============================] - 0s 24ms/step - loss: 0.0719 -
mean_squared_logarithmic_error: 0.0708 - val_loss: 0.0648 -
val_mean_squared_logarithmic_error: 0.0648
Epoch 89/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0679 -
mean_squared_logarithmic_error: 0.0678 - val_loss: 0.0649 -
```

```
val_mean_squared_logarithmic_error: 0.0649
Epoch 90/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0686 -
mean_squared_logarithmic_error: 0.0639 - val_loss: 0.0652 -
val_mean_squared_logarithmic_error: 0.0652
Epoch 91/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0761 -
mean_squared_logarithmic_error: 0.0751 - val_loss: 0.0650 -
val_mean_squared_logarithmic_error: 0.0650
Epoch 92/100
9/9 [==============================] - 0s 28ms/step - loss: 0.0662 -
mean_squared_logarithmic_error: 0.0615 - val_loss: 0.0643 -
val_mean_squared_logarithmic_error: 0.0643
Epoch 93/100
9/9 [==============================] - 0s 24ms/step - loss: 0.0687 -
mean_squared_logarithmic_error: 0.0757 - val_loss: 0.0637 -
val_mean_squared_logarithmic_error: 0.0637
Epoch 94/100
9/9 [==============================] - 0s 29ms/step - loss: 0.0630 -
mean_squared_logarithmic_error: 0.0657 - val_loss: 0.0635 -
val_mean_squared_logarithmic_error: 0.0635
Epoch 95/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0713 -
mean_squared_logarithmic_error: 0.0691 - val_loss: 0.0631 -
val_mean_squared_logarithmic_error: 0.0631
Epoch 96/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0662 -
mean_squared_logarithmic_error: 0.0629 - val_loss: 0.0633 -
val_mean_squared_logarithmic_error: 0.0633
Epoch 97/100
9/9 [==============================] - 0s 23ms/step - loss: 0.0652 -
mean_squared_logarithmic_error: 0.0679 - val_loss: 0.0626 -
val_mean_squared_logarithmic_error: 0.0626
Epoch 98/100
9/9 [==============================] - 0s 26ms/step - loss: 0.0625 -
mean_squared_logarithmic_error: 0.0603 - val_loss: 0.0627 -
val_mean_squared_logarithmic_error: 0.0627
Epoch 99/100
9/9 [==============================] - 0s 21ms/step - loss: 0.0708 -
mean_squared_logarithmic_error: 0.0681 - val_loss: 0.0617 -
val_mean_squared_logarithmic_error: 0.0617
Epoch 100/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0668 -
mean_squared_logarithmic_error: 0.0650 - val_loss: 0.0625 -
val_mean_squared_logarithmic_error: 0.0625
```

[28]:
```
plot_history(historyB, 'mean_squared_logarithmic_error')
```



3.3.3 Predict and evaluate the residuals on $D^{(2)}$

[29]:
```
pred = modelB.predict(x_d2)

residuals = np.array([abs(float(y_d2[i]- pred[i])) for i in range(len(y_d2))])
```

```
25/25 [==============================] - 0s 4ms/step
```

3.3.4 Finding d

[30]:
```
alpha = 0.4

k = ceil((len(y)/2+1)*(1-alpha))

d = np.sort(residuals)[k-1]

print("d is equal to:", d)
```

```
d is equal to: 23.3392333984375
```

3.3.5 Prediction intervals

[31]:
```
pred_b = modelB.predict(x_B)
predictions = [[float(elem-d), float(elem+d)] for elem in pred_b]
```

```
1/1 [==============================] - 0s 93ms/step
```

3.3.6 Check if the our intervals cover the actual response

[32]:
```
count = 0
for i in range(len(y_B)):
    if y_B[i] >= predictions[i][0] and y_B[i] <= predictions[i][1]:
        print(i+1, y_B[i], predictions[i], True)
        count += 1
    else:
        print(i+1, y_B[i], predictions[i], False)

print()
print("Percentage of True ( that should be >=", 1-alpha, "):", count/len(y_B))
```

```
1 [128.] [102.82158660888672, 149.50006103515625] True
2 [175.] [107.43702697753906, 154.11549377441406] False
3 [113.] [127.48135375976562, 174.15982055664062] False
4 [137.6] [132.69534301757812, 179.37380981445312] True
5 [110.] [97.92150115966797, 144.5999755859375] True
6 [134.] [103.06685638427734, 149.74533081054688] True
7 [180.] [113.42539978027344, 160.10386657714844] False
8 [174.] [134.51504516601562, 181.19351196289062] True
9 [134.] [99.95001220703125, 146.62847900390625] True
10 [126.] [115.32115173339844, 161.99961853027344] True

Percentage of True ( that should be >= 0.6 ): 0.7
```

3.3.7 Visualization of the results

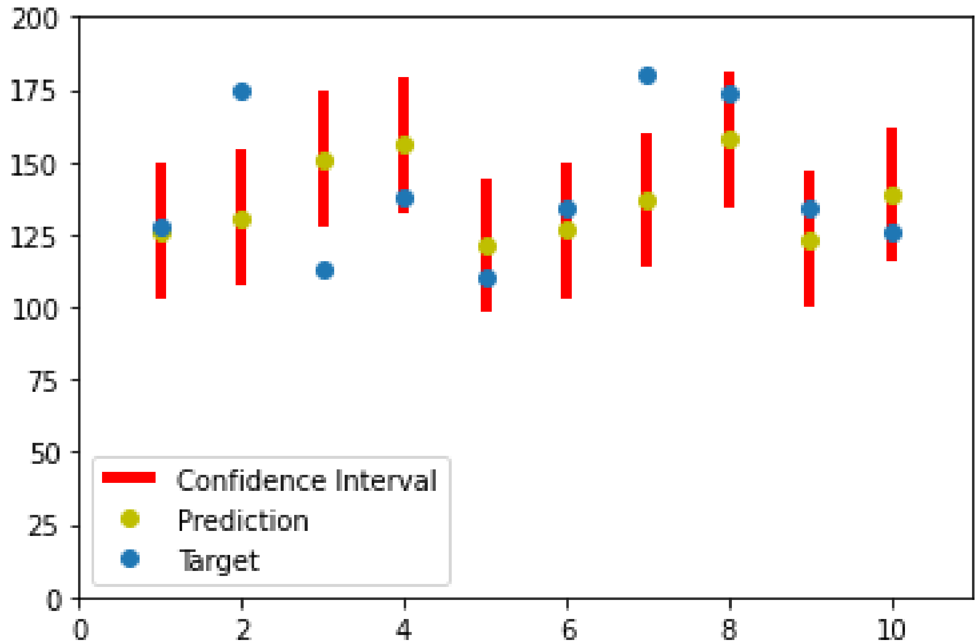
[33]:
```
x = list(range(1, 11))

start = [pred[0] for pred in predictions]
stop = [pred[1] for pred in predictions]

plt.xlim(0, 11)
plt.ylim(0, 200)

plt.vlines(x, start, stop, 'r', lw=4, label = "Confidence Interval")
plt.plot(x, pred_b, 'o', color = 'y', label = "Prediction")
plt.plot(x, y_B, 'o', label = "Target")
plt.legend()
```

```
plt.show()
```



3.3.8 Comment

Using the **Split Conformal Prediction for Regression** we're able to give as output a set on confidence interval: this means that we're confident that the target value will be inside this interval with a probability $P \geq 1 - \alpha$, in our case ($\alpha = 0.4$) with $P \geq 0.6$; this can be also be seen from the figure above, in which the *target values* falls into the confidece intervals in 7 cases of 10, so with a probability of $P' = 0.7$, which is greater than 0.6.

3.4 Building the predictive sets for the randomly picked $m = 100$ observations of the test set

[34]:
```
np.random.seed(1234)

random_idx = np.random.choice(len(test_df), size = 100)

random_df = test_df.iloc[random_idx]

x_random = random_df.iloc[:, 7011:7041].values
x_random = StandardScaler().fit_transform(x_random)
x_random = pca.transform(x_random)

pred_btest = modelB.predict(x_random)
predictions_test = [[float(elem-d), float(elem+d)] for elem in pred_btest]
```

```
4/4 [==============================] - 0s 4ms/step
```

3.4.1 Visualization of the intervals

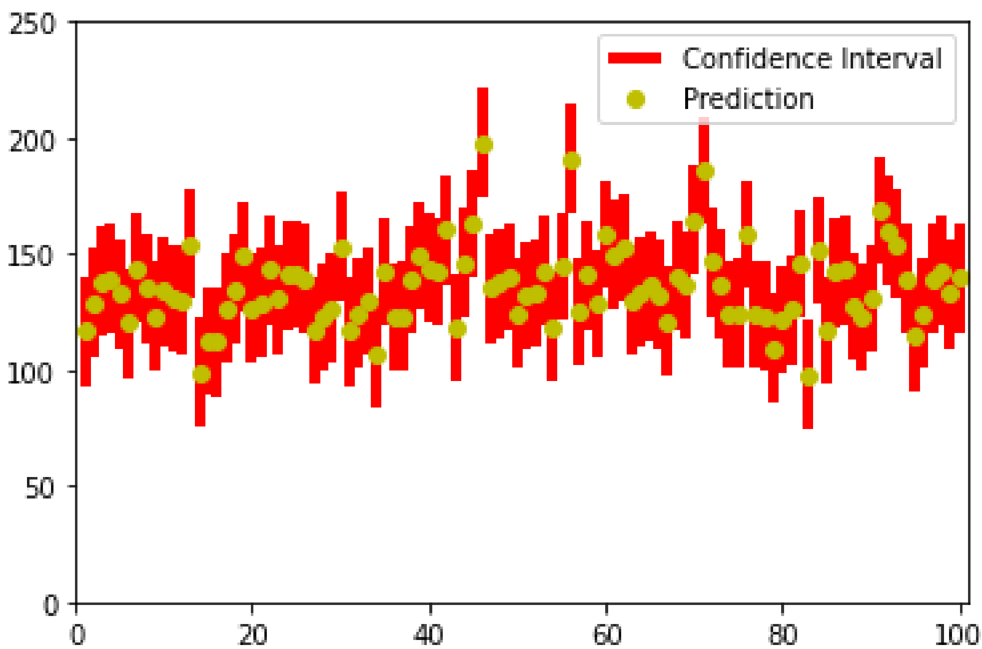
[35]:
```
x = list(range(1, 101))

start = [pred[0] for pred in predictions_test]
stop = [pred[1] for pred in predictions_test]

plt.xlim(0, 101)
plt.ylim(0, 250)

plt.vlines(x, start, stop, 'r', lw=4, label = "Confidence Interval")
plt.plot(x, pred_btest, 'o', color = 'y', label = "Prediction")
plt.legend()

plt.show()
```



3.4.2 Comment

The **Split Conformal Prediction for Regression** gives us more confidece in using our model to predict on the test set: we have the security of having at least $1 - \alpha$ % of the target values (in our case the 0.6%) falling into our confidence intervals.

We've have to remember that for gaining this confidence in the output, we're sacrificing half of our training set (loosing training values). To fully understand this last statement we can use as an example our models: using all the train set in training as for the first model makes our model gain more knowledge of the data distribution, while using part of that set to build confidence intervals, makes the output more resilient to variance; **we're sacrificing knowledge for confidence**.