lab02

April 25, 2024

0.0.1 People's Friendship University in Russia

Faculty of Science

Department of Mathematical Modeling and Artificial Intelligence

0.1 Labratory work №2 report

0.1.1 Meathods of machine learning

Student: Abu Suveilim Mukhammed M.

Group: NKNbd-01-21

0.2 Moscow 2024

0.2.1 Version №13

1. Dataset: cherry_blossoms
2. Independent variable: temp_lower
3. Dependent variable: year
4. Add. characteristic: having minimal covariance with the independent variable
5. Visualization additional. characteristic – empirical distribution function
6. Regression quality indicator – R^2 (coefficient of determination)
7. Polynomial degree: 4
8. Parameters of the deep neural network: number of hidden layers – 5, number of neurons in the hidden layer – 32, activation function – hyperbolic tangent.

0.3 1. Load the data set specified in the individual task from Tensorflow Datasets, including the independent feature and dependent feature (response) specified in the task. Leave in the set features that take numeric values.

Download the needed libraries and packages

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import tensorflow_datasets as tfds
```

[2]:
```
# loading cherry_blossoms dataset
ds = tfds.load("cherry_blossoms", split='train')
ds
# Convert tf.data.cherry_blossoms to a panda dataframe
df = tfds.as_dataframe(ds)
df.head()
```

```
WARNING:absl:You use TensorFlow DType <dtype: 'int32'> in tfds.features This
will soon be deprecated in favor of NumPy DTypes. In the meantime it was
converted to int32.
WARNING:absl:You use TensorFlow DType <dtype: 'float32'> in tfds.features This
will soon be deprecated in favor of NumPy DTypes. In the meantime it was
converted to float32.
```

[2]:
```
     doy  temp  temp_lower  temp_upper  year
0    NaN  6.46        4.76        8.16  1300
1  105.0  5.63        4.90        6.37  1638
2  109.0  5.81        4.68        6.95  1347
3  104.0  5.70        4.87        6.53  1187
4  107.0  6.20        5.31        7.09  1617
```

[3]:
```
print(df.info())
```

```
<class
'tensorflow_datasets.core.as_dataframe.as_dataframe.<locals>.StyledDataFrame'>
RangeIndex: 1215 entries, 0 to 1214
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   doy         827 non-null    float32
 1   temp        1124 non-null   float32
 2   temp_lower  1124 non-null   float32
 3   temp_upper  1124 non-null   float32
 4   year        1215 non-null   int32
dtypes: float32(4), int32(1)
memory usage: 23.9 KB
None
```

[4]:
```
#Let's see which columns have NaN values
# count NaN values in each column
print(df.isnull().sum())
```

```
doy           388
temp           91
temp_lower     91
```

```
temp_upper     91
year            0
dtype: int64
```

[5]:
```
#Let's drop the doy column since it has a lot of NaN values
df = df.drop(columns=['doy'])
```

[6]:
```
#Check first 5 rows and info
print(df.head())
print(df.info())
```

```
   temp  temp_lower  temp_upper  year
0  6.46        4.76        8.16  1300
1  5.63        4.90        6.37  1638
2  5.81        4.68        6.95  1347
3  5.70        4.87        6.53  1187
4  6.20        5.31        7.09  1617
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1215 entries, 0 to 1214
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   temp        1124 non-null   float32
 1   temp_lower  1124 non-null   float32
 2   temp_upper  1124 non-null   float32
 3   year        1215 non-null   int32
dtypes: float32(3), int32(1)
memory usage: 19.1 KB
None
```

[7]:
```
df[df.isnull().any(axis=1)]
```

[7]:
```
       temp  temp_lower  temp_upper  year
38      NaN         NaN         NaN   856
42      NaN         NaN         NaN   829
78      NaN         NaN         NaN   807
93      NaN         NaN         NaN   834
107     NaN         NaN         NaN   817
…     …         …         …    …
1132    NaN         NaN         NaN   803
1136    NaN         NaN         NaN  1987
1173    NaN         NaN         NaN   858
1187    NaN         NaN         NaN  1981
1200    NaN         NaN         NaN   838

[91 rows x 4 columns]
```

[8]:
```
#let's drop all rows with at least one nan value
df = df.dropna()
```

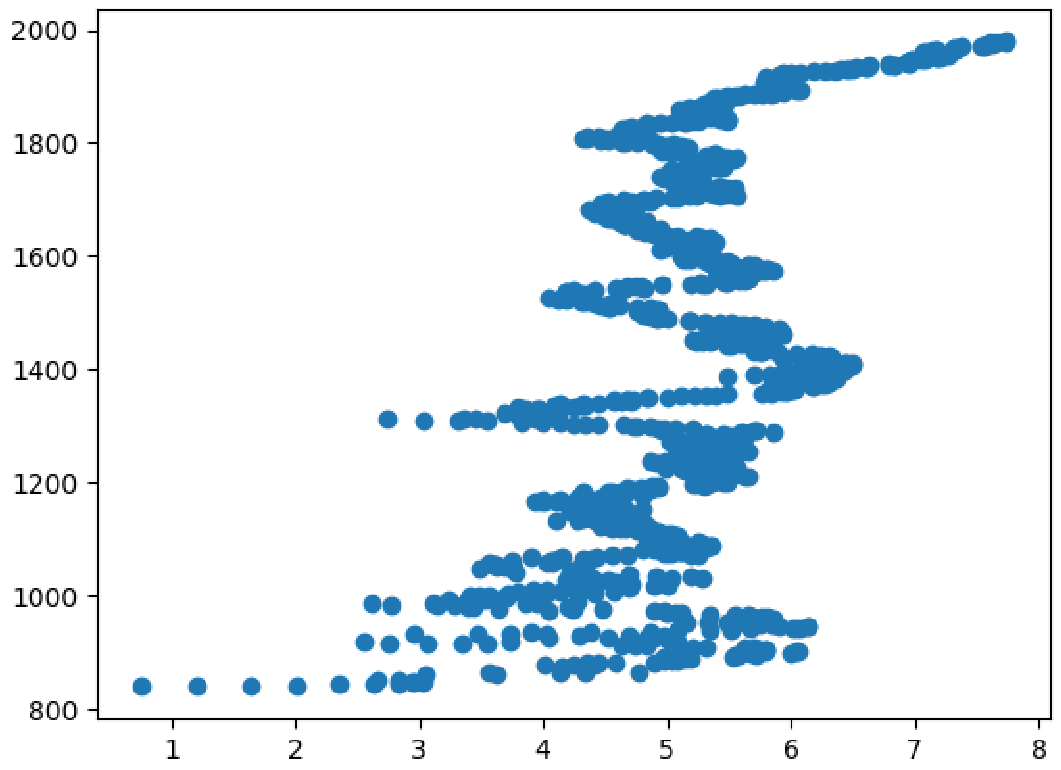
[9]:
```
print(df.head())
print(df.info())
```

```
   temp  temp_lower  temp_upper  year
0  6.46        4.76        8.16  1300
1  5.63        4.90        6.37  1638
2  5.81        4.68        6.95  1347
3  5.70        4.87        6.53  1187
4  6.20        5.31        7.09  1617
<class 'pandas.core.frame.DataFrame'>
Index: 1124 entries, 0 to 1214
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   temp        1124 non-null   float32
 1   temp_lower  1124 non-null   float32
 2   temp_upper  1124 non-null   float32
 3   year        1124 non-null   int32
dtypes: float32(3), int32(1)
memory usage: 26.3 KB
None
```

0.4 2. Remove outlier points from the set using a standardized score (Z-score) so that outlier points make up 5% to 10% of all points in the data set. Visualize the points of the original data set on a plane as a scatterplot (X-axis is the independent feature temp_lower, Y-axis is the dependent feature year), showing the points left in the set and the points removed in different colors, labeling the axes and figure, and creating a legend.

[10]:
```
#let's see the scatter plot
plt.scatter(df['temp_lower'], df['year'])
```

[10]:
```
<matplotlib.collections.PathCollection at 0x1ef59ca8150>
```



[59]:
```
#let's standardize independent feeature
df['temp_lower'] = (df['temp_lower'] - np.mean(df['temp_lower']))/np.
 ↪std(df['temp_lower'])
df['temp'] = (df['temp'] - np.mean(df['temp']))/np.std(df['temp'])
df['temp_upper'] = (df['temp_upper'] - np.mean(df['temp_upper']))/np.
 ↪std(df['temp_upper'])

print(np.mean(df['temp_lower']), np.std(df['temp_lower']))
print(np.mean(df['temp']), np.std(df['temp']))
print(np.mean(df['temp_upper']), np.std(df['temp_upper']))
```

```
-7.636182e-09 1.0
-1.01815765e-08 1.0
5.0907882e-09 1.0
```

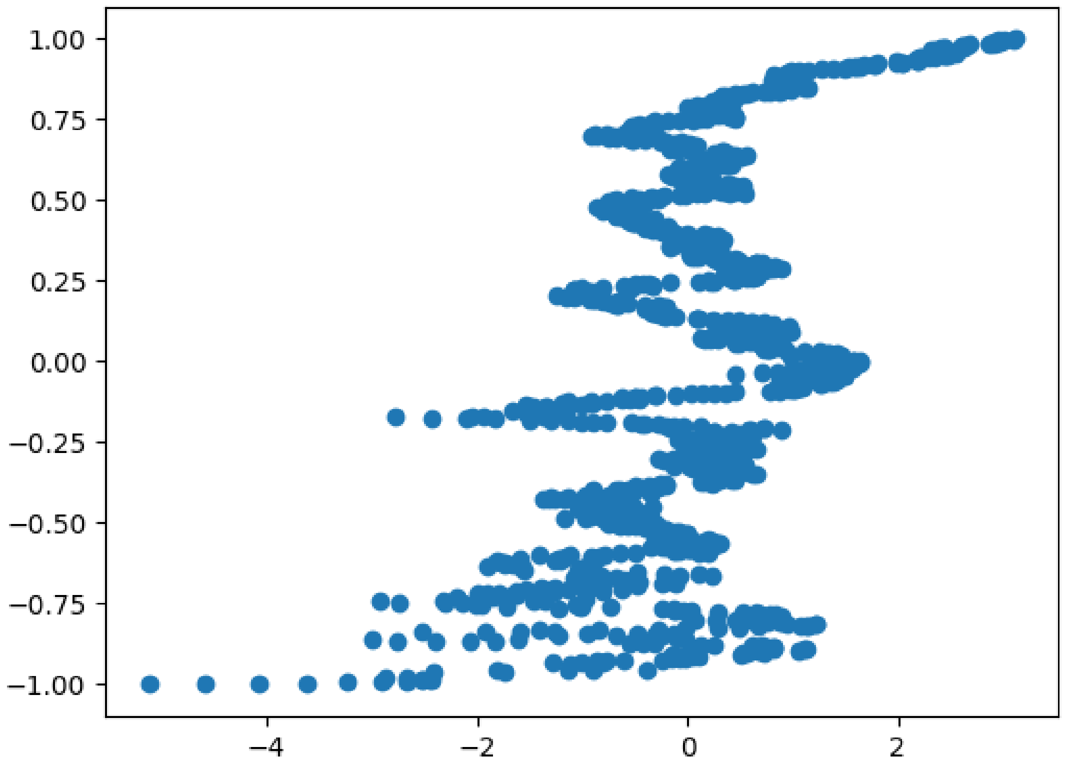
[60]:
```
#let's scale the dependent feature on the interval [-1,1]
df['year'] = (2 * (df['year'] - np.min(df['year'])) / (np.max(df['year']) - np.
 ↪min(df['year']))) - 1
```

[61]:
```
np.min(df['temp_lower']), np.max(df['temp_lower']), np.min(df['year']), np.
 ↪max(df['year'])
```

```
[61]: (-5.116575, 3.1072328, -1.0, 1.0)
```

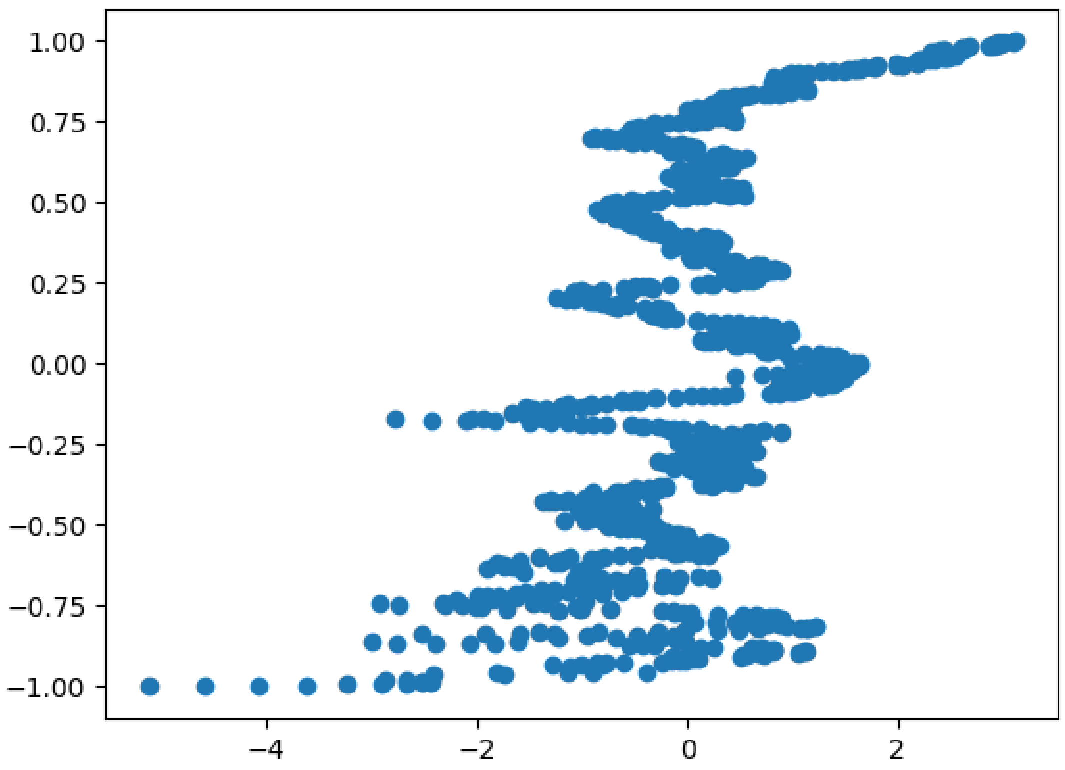
```
[216]: plt.scatter(df['temp_lower'], df['year'])
```

```
[216]: <matplotlib.collections.PathCollection at 0x1ef6be90c10>
```



```
[63]: plt.scatter(df['temp_lower'], df['year'])
```

```
[63]: <matplotlib.collections.PathCollection at 0x1ef606f15d0>
```



[64]:
```
x_out = df
pd.DataFrame(x_out).to_numpy()
```

[64]:
```
array([[ 0.47955483, -0.39876768,  0.98223627, -0.1919369 ],
       [-0.7716651 , -0.23405635, -0.82132876,  0.40052585],
       [-0.50031644, -0.49288896, -0.23693347, -0.10955302],
       …,
       [ 0.81120318,  1.15422559,  0.08549175,  0.90359334],
       [ 0.01223177,  0.22478254, -0.17647836,  0.37598598],
       [ 1.39912629,  1.06010485,  0.96208525, -0.028922  ]])
```

[65]:
```
x_out.shape
```

[65]:
```
(1124, 4)
```

[82]:
```
x_out.info
```

[82]:
```
<bound method DataFrame.info of           temp  temp_lower  temp_upper      year
0     0.479555   -0.398768    0.982236 -0.191937
1    -0.771665   -0.234056   -0.821329  0.400526
2    -0.500316   -0.492889   -0.236933 -0.109553
3    -0.666141   -0.269352   -0.660116 -0.390009
```

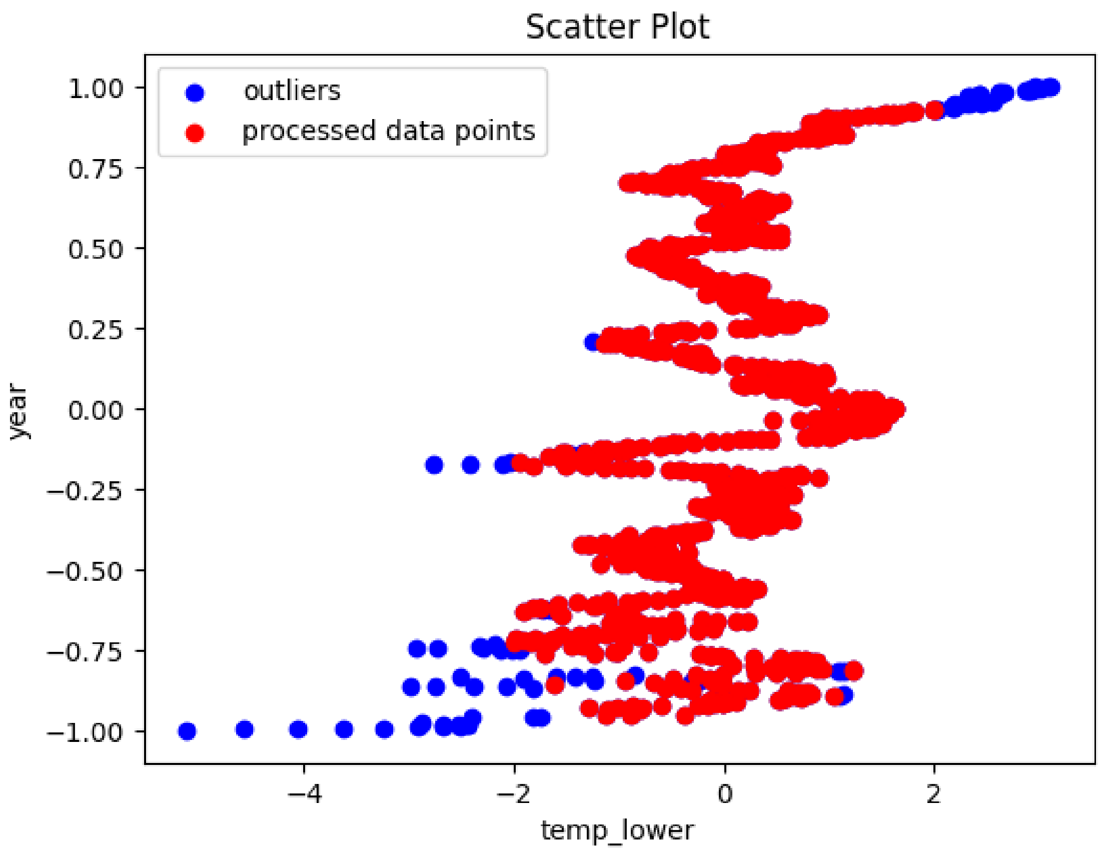
```
4     0.087606    0.248313   -0.095872  0.363716
…        …          …          …         …
1210  0.464480    0.836568   -0.085797  0.293602
1211 -0.651066   -0.151701   -0.750798  0.404032
1212  0.811203    1.154226    0.085492  0.903593
1213  0.012232    0.224783   -0.176478  0.375986
1214  1.399126    1.060105    0.962085 -0.028922

[1124 rows x 4 columns]>
```

[91]:
```python
z_score = 2
print('                    = %d' % (x_out.shape[0]))
x2_out = x_out.loc[((x_out >= -z_score).sum(axis=1)==4) & ((x_out <= z_score).
 ↪sum(axis=1)==4),:] # NB .loc
print('                      = %d' % (x2_out.shape[0]))
```

```
                   = 1124
                     = 1028
```

[94]:
```python
#About 8.5% of data pouint were removed
#Let's see the difference
plt.scatter(x_out['temp_lower'], x_out['year'], color='blue', label='outliers')
plt.scatter(x2_out['temp_lower'], x2_out['year'], color='red', label='processed␣
 ↪data points')
plt.xlabel('temp_lower')  # Customize x-axis label
plt.ylabel('year')  # Customize y-axis label
plt.title('Scatter Plot')
plt.legend()
plt.legend(loc='upper left')
```

[94]:
```
<matplotlib.legend.Legend at 0x1ef6029a5d0>
```


Scatter Plot
outliers
processed data points
year
temp_lower
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−4
−2
0
2

0.5 3. Standardize the independent feature and scale by the interval [-1, 1] of the dependent feature. Solve linear regression and polynomial regression problems for the polynomial degree specified in the individual assignment using neural networks with one neuron and evaluate the quality of the resulting models according to the indicator specified in the individual assignment. Monitor the training of neural networks, changing hyperparameters (loss function, optimizer, training step, etc.) as necessary or applying regularization.

0.6 4. Plot learning curves for the constructed neural networks depending on the number of epochs. Create a legend on the visualization.

0.7 5. Visualize the data set points on a plane as a scatterplot (X-axis - independent feature, Y-axis - dependent feature), as well as linear and polynomial regression lines (in different colors), labeling the axes and figure and creating a legend.

[95]:
```python
#Let's define a class `RegressionSGD` that uses stochastic gradient descent
class RegressionSGD:

    def __init__(self):
        self.coef_ = None
        self.intercept_ = None
        self._theta = None

    def fit(self, X_train, y_train, n_iters=50, t0=5, t1=50):
        assert X_train.shape[0] == y_train.shape[0], \
            "    X_train                    y_train"
        assert n_iters >= 1

        def dJ_sgd(theta, X_b_i, y_i):
            return X_b_i * (X_b_i.dot(theta) - y_i) * 2.

        def sgd(X_b, y, initial_theta, n_iters=5, t0=5, t1=50):

            def learning_rate(t):
                return t0 / (t + t1)

            theta = initial_theta
            m = len(X_b)
            for i_iter in range(n_iters):
                indexes = np.random.permutation(m)
                X_b_new = X_b[indexes,:]
                y_new = y[indexes]
                for i in range(m):
                    gradient = dJ_sgd(theta, X_b_new[i], y_new[i])
                    theta = theta - learning_rate(i_iter * m + i) * gradient
            return theta

        X_b = np.hstack([np.ones((len(X_train), 1)), X_train])
        initial_theta = np.random.randn(X_b.shape[1])
        self._theta = sgd(X_b, y_train, initial_theta, n_iters, t0, t1)

        self.intercept_ = self._theta[0]
        self.coef_ = self._theta[1:]

        return self

    def predict(self, X_predict):
        assert self.intercept_ is not None and self.coef_ is not None, \
            "                        !"
        assert X_predict.shape[1] == len(self.coef_), \
            "  -         X_predict              -         X_train"

        X_b = np.hstack([np.ones((len(X_predict), 1)), X_predict])
        return X_b.dot(self._theta)

    def score(self, X_test, y_test):
        y_predict = self.predict(X_test)
        return r2_score(y_test, y_predict)

    def __repr__(self):
        return "RegressionSGD()"
```

[98]:
```
X = x2_out['temp_lower'].values.reshape(-1,1)           #                  ␣
 ↪
y = x2_out['year']
```

[105]:
```
X = pd.DataFrame(X).to_numpy()
```

[109]:
```
y.info
```

[109]:
```
<bound method Series.info of 0       -0.191937
1        0.400526
2       -0.109553
3       -0.390009
4        0.363716
           …
1210     0.293602
1211     0.404032
1212     0.903593
1213     0.375986
1214    -0.028922
Name: year, Length: 1028, dtype: float64>
```

[110]:
```
y = pd.Series(y).to_numpy()
```

[112]:
```
y.shape
```

[112]:
```
(1028,)
```

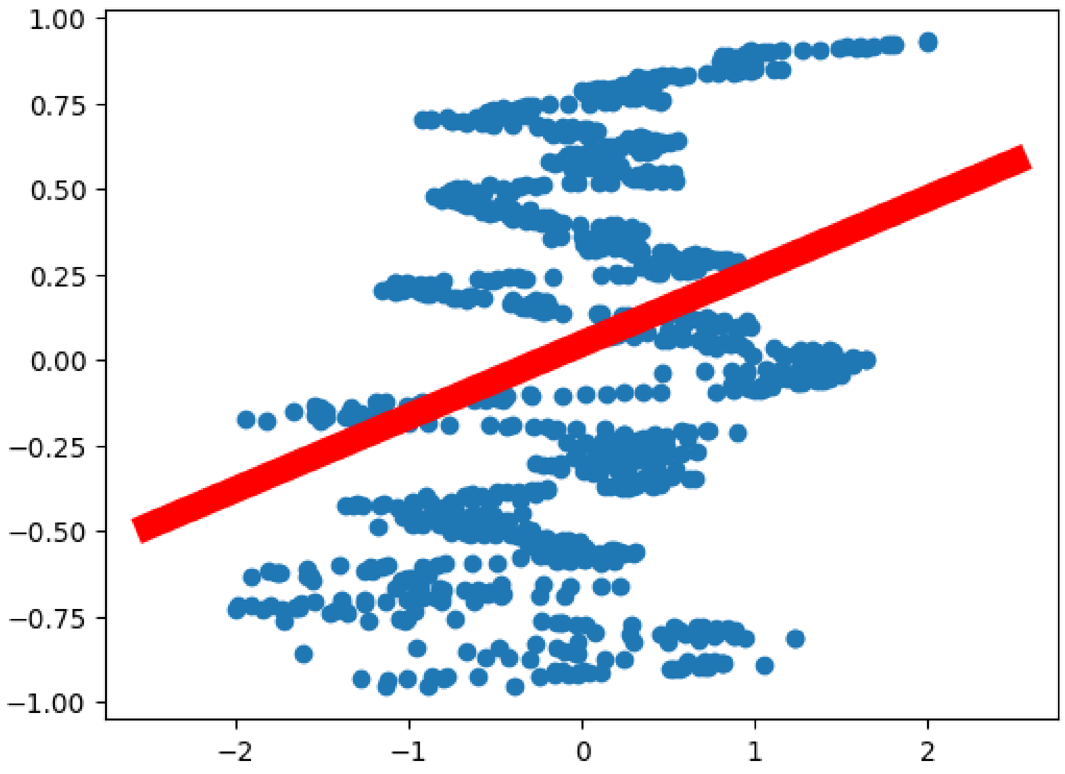
[107]:
```
X.shape
```

[107]:
```
(1028, 1)
```

[113]:
```
reg = RegressionSGD()
reg.fit(X, y, n_iters=2)
reg.coef_, reg.intercept_
```

[113]:
```
(array([0.21296096]), 0.046003204395O6115)
```

[116]:
```
plt.scatter(x2_out['temp_lower'], x2_out['year'])
plot_x = np.linspace(-2.5, 2.5, 101)
plt.plot(plot_x, reg.predict(plot_x.reshape(-1,1)), c='r', lw=10);
```



R^2 score for linear model

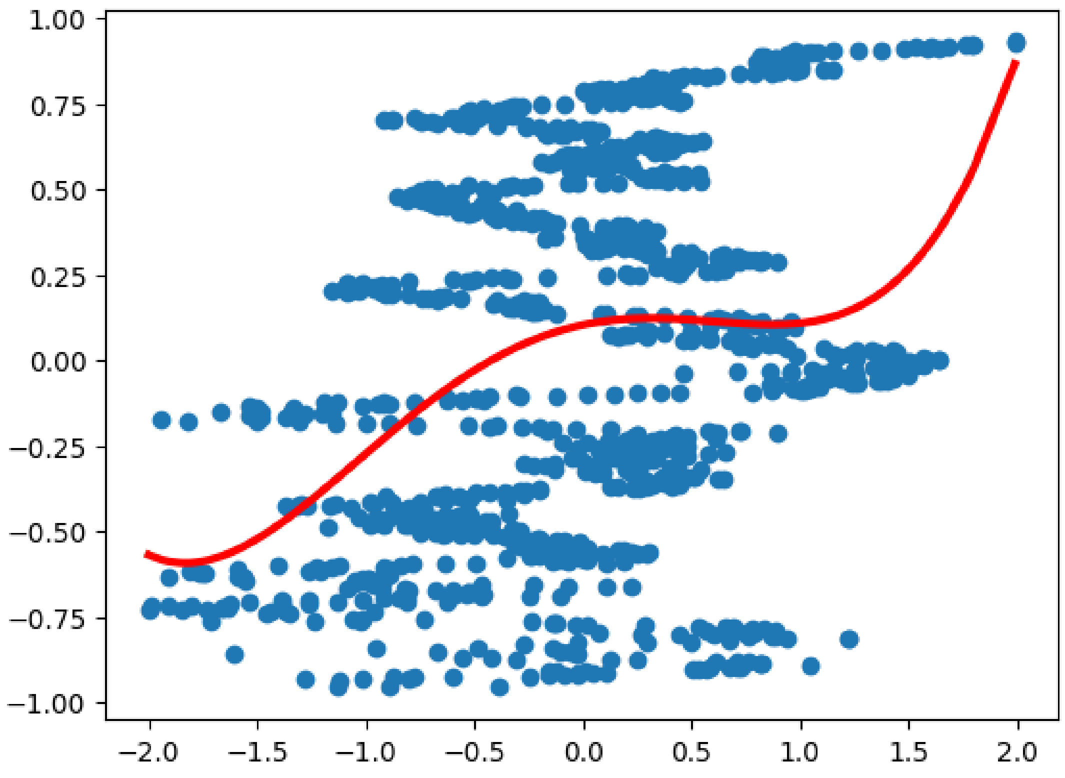
```
[186]: #We can that the model only explains 9% of the variance which is very low
       r2 = r2_score(y, reg.predict(X))
       print(r2)
```

```
0.09201340260472668
```

```
[150]: #
       #Let's prepare input data for the regression model with two characteristics -␣
        ↪linear and 4th power dependence on the independent variable:
       X4 = np.hstack([X, X**2, X**3, X**4]) #
       X4.shape
```

```
[150]: (1028, 4)
```

```
[152]: reg2 = RegressionSGD()
       reg2.fit(X4, y, n_iters=2000)
       y_predict2 = reg2.predict(X4)

       plt.scatter(x2_out['temp_lower'], y)
       plt.plot(np.sort(x2_out['temp_lower']), y_predict2[np.
        ↪argsort(x2_out['temp_lower'])], c='r', lw=3);
```



```
[153]: reg2.coef_, reg2.intercept_
```

```
[153]: (array([ 0.133859  , -0.25286321,  0.05737394,  0.06668921]),
        0.10283383497566828)
```

```
[162]: #Let's create a simple neural network with one layer of one neuron and two␣
        ↪input neurons:
       reg2_model = tf.keras.Sequential([
           tf.keras.Input(shape=(4,)), #Polynomial degree
           tf.keras.layers.Dense(units=1)
       ])
```

```
[163]: reg2_model.summary()
```

```
Model: "sequential_3"



 Layer (type)                          Output Shape                        ␣
 ↪Param #

 dense_3 (Dense)                       (None, 1)                               ␣
 ↪  5



Total params: 5 (20.00 B)


Trainable params: 5 (20.00 B)


Non-trainable params: 0 (0.00 B)
```

```
[164]: from sklearn.metrics import r2_score
```

```
[187]: # Custom metric function for R2 score
       def r2_metric(y_true, y_pred):
           r2 = r2_score(y_true, y_pred)
           return r2
```

```
[196]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
[198]: reg2_model.compile(
           optimizer=tf.optimizers.AdamW(learning_rate=0.01),
           loss='mean_absolute_error')
```

```
[199]: history = reg2_model.fit(
           X4, y,
           epochs=100,
           #
           verbose=1,
           #      (     )   30%
           validation_split = 0.3)
```

```
Epoch 1/100
23/23            0s 4ms/step - loss:
0.4450 - val_loss: 0.4067
Epoch 2/100
23/23            0s 1ms/step - loss:
0.4275 - val_loss: 0.4038
Epoch 3/100
23/23            0s 1ms/step - loss:
0.4405 - val_loss: 0.4028
Epoch 4/100
23/23            0s 1ms/step - loss:
0.4467 - val_loss: 0.4019
Epoch 5/100
23/23            0s 1ms/step - loss:
0.4303 - val_loss: 0.4028
Epoch 6/100
23/23            0s 1ms/step - loss:
0.4395 - val_loss: 0.3999
Epoch 7/100
23/23            0s 1ms/step - loss:
0.4245 - val_loss: 0.4042
Epoch 8/100
23/23            0s 1ms/step - loss:
0.4273 - val_loss: 0.4026
Epoch 9/100
23/23            0s 1ms/step - loss:
0.4328 - val_loss: 0.4002
Epoch 10/100
23/23            0s 1ms/step - loss:
0.4393 - val_loss: 0.4011
Epoch 11/100
23/23            0s 1ms/step - loss:
0.4204 - val_loss: 0.4028
Epoch 12/100
23/23            0s 2ms/step - loss:
0.4402 - val_loss: 0.4021
Epoch 13/100
23/23            0s 1ms/step - loss:
0.4440 - val_loss: 0.3990
Epoch 14/100
23/23            0s 1ms/step - loss:
0.4352 - val_loss: 0.3995
```

```
Epoch 15/100
23/23            0s 1ms/step - loss:
0.4217 - val_loss: 0.3979
Epoch 16/100
23/23            0s 1ms/step - loss:
0.4267 - val_loss: 0.4010
Epoch 17/100
23/23            0s 1ms/step - loss:
0.4296 - val_loss: 0.4007
Epoch 18/100
23/23            0s 1ms/step - loss:
0.4306 - val_loss: 0.3984
Epoch 19/100
23/23            0s 1ms/step - loss:
0.4416 - val_loss: 0.3985
Epoch 20/100
23/23            0s 1ms/step - loss:
0.4312 - val_loss: 0.3991
Epoch 21/100
23/23            0s 1ms/step - loss:
0.4338 - val_loss: 0.3999
Epoch 22/100
23/23            0s 1ms/step - loss:
0.4326 - val_loss: 0.3993
Epoch 23/100
23/23            0s 1ms/step - loss:
0.4324 - val_loss: 0.3984
Epoch 24/100
23/23            0s 1ms/step - loss:
0.4386 - val_loss: 0.4019
Epoch 25/100
23/23            0s 1ms/step - loss:
0.4379 - val_loss: 0.3984
Epoch 26/100
23/23            0s 1ms/step - loss:
0.4135 - val_loss: 0.4001
Epoch 27/100
23/23            0s 1ms/step - loss:
0.4269 - val_loss: 0.3982
Epoch 28/100
23/23            0s 1ms/step - loss:
0.4342 - val_loss: 0.4003
Epoch 29/100
23/23            0s 1ms/step - loss:
0.4433 - val_loss: 0.3999
Epoch 30/100
23/23            0s 1ms/step - loss:
0.4433 - val_loss: 0.4000
```

```
Epoch 31/100
23/23             0s 1ms/step - loss:
0.4233 - val_loss: 0.3996
Epoch 32/100
23/23             0s 1ms/step - loss:
0.4298 - val_loss: 0.4002
Epoch 33/100
23/23             0s 1ms/step - loss:
0.4377 - val_loss: 0.3998
Epoch 34/100
23/23             0s 1ms/step - loss:
0.4089 - val_loss: 0.3974
Epoch 35/100
23/23             0s 1ms/step - loss:
0.4260 - val_loss: 0.3998
Epoch 36/100
23/23             0s 1ms/step - loss:
0.4317 - val_loss: 0.3976
Epoch 37/100
23/23             0s 1ms/step - loss:
0.4208 - val_loss: 0.4042
Epoch 38/100
23/23             0s 1ms/step - loss:
0.4352 - val_loss: 0.3991
Epoch 39/100
23/23             0s 1ms/step - loss:
0.4330 - val_loss: 0.3982
Epoch 40/100
23/23             0s 1ms/step - loss:
0.4268 - val_loss: 0.3968
Epoch 41/100
23/23             0s 2ms/step - loss:
0.4270 - val_loss: 0.3990
Epoch 42/100
23/23             0s 1ms/step - loss:
0.4266 - val_loss: 0.3996
Epoch 43/100
23/23             0s 1ms/step - loss:
0.4457 - val_loss: 0.4057
Epoch 44/100
23/23             0s 1ms/step - loss:
0.4294 - val_loss: 0.3976
Epoch 45/100
23/23             0s 1ms/step - loss:
0.4202 - val_loss: 0.4042
Epoch 46/100
23/23             0s 1ms/step - loss:
0.4375 - val_loss: 0.4011
```

```
Epoch 47/100
23/23            0s 1ms/step - loss:
0.4307 - val_loss: 0.3986
Epoch 48/100
23/23            0s 1ms/step - loss:
0.4268 - val_loss: 0.3980
Epoch 49/100
23/23            0s 1ms/step - loss:
0.4228 - val_loss: 0.3996
Epoch 50/100
23/23            0s 1ms/step - loss:
0.4358 - val_loss: 0.3982
Epoch 51/100
23/23            0s 1ms/step - loss:
0.4250 - val_loss: 0.3984
Epoch 52/100
23/23            0s 1ms/step - loss:
0.4504 - val_loss: 0.3985
Epoch 53/100
23/23            0s 1ms/step - loss:
0.4046 - val_loss: 0.4006
Epoch 54/100
23/23            0s 1ms/step - loss:
0.4242 - val_loss: 0.4034
Epoch 55/100
23/23            0s 1ms/step - loss:
0.4335 - val_loss: 0.4032
Epoch 56/100
23/23            0s 1ms/step - loss:
0.4412 - val_loss: 0.4004
Epoch 57/100
23/23            0s 1ms/step - loss:
0.4297 - val_loss: 0.3986
Epoch 58/100
23/23            0s 1ms/step - loss:
0.4367 - val_loss: 0.3984
Epoch 59/100
23/23            0s 1ms/step - loss:
0.4307 - val_loss: 0.3978
Epoch 60/100
23/23            0s 1ms/step - loss:
0.4480 - val_loss: 0.3985
Epoch 61/100
23/23            0s 1ms/step - loss:
0.4320 - val_loss: 0.3984
Epoch 62/100
23/23            0s 1ms/step - loss:
0.4387 - val_loss: 0.3993
```

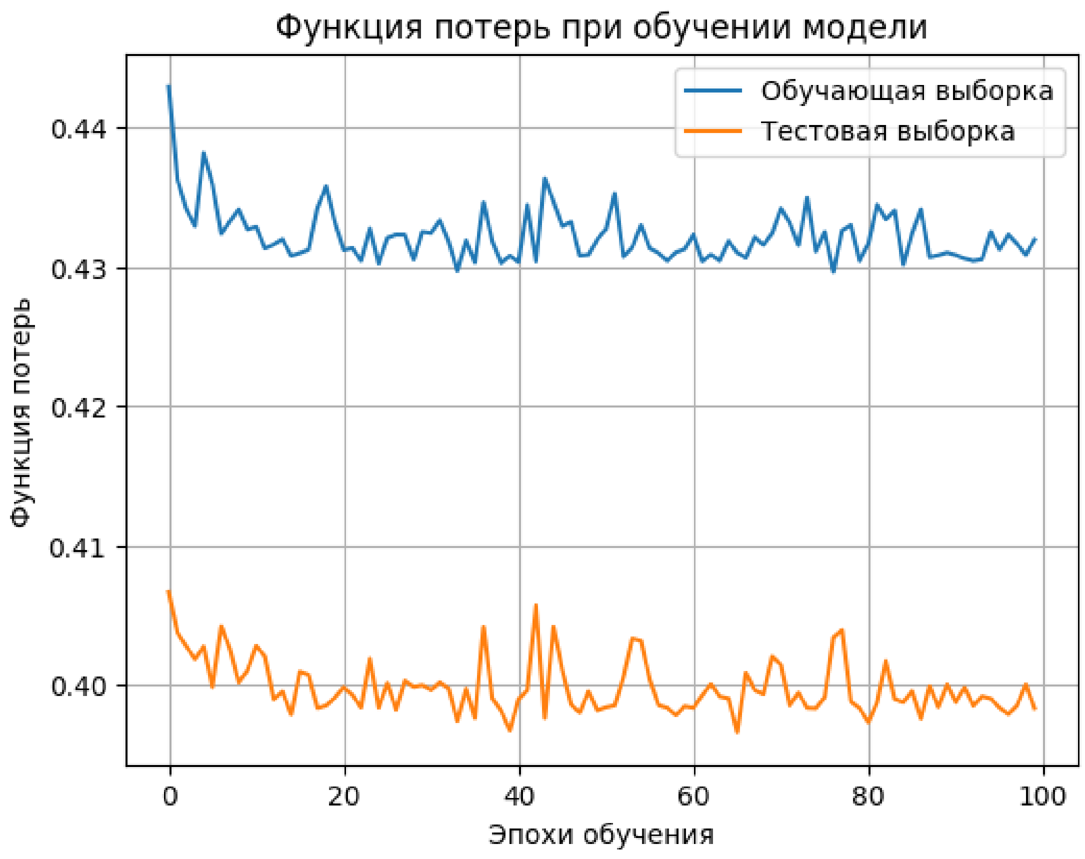
```
Epoch 63/100
23/23            0s 1ms/step - loss:
0.4331 - val_loss: 0.4001
Epoch 64/100
23/23            0s 1ms/step - loss:
0.4280 - val_loss: 0.3992
Epoch 65/100
23/23            0s 1ms/step - loss:
0.4345 - val_loss: 0.3990
Epoch 66/100
23/23            0s 1ms/step - loss:
0.4248 - val_loss: 0.3966
Epoch 67/100
23/23            0s 1ms/step - loss:
0.4307 - val_loss: 0.4009
Epoch 68/100
23/23            0s 1ms/step - loss:
0.4170 - val_loss: 0.3997
Epoch 69/100
23/23            0s 1ms/step - loss:
0.4391 - val_loss: 0.3994
Epoch 70/100
23/23            0s 1ms/step - loss:
0.4221 - val_loss: 0.4021
Epoch 71/100
23/23            0s 1ms/step - loss:
0.4373 - val_loss: 0.4015
Epoch 72/100
23/23            0s 1ms/step - loss:
0.4324 - val_loss: 0.3985
Epoch 73/100
23/23            0s 1ms/step - loss:
0.4350 - val_loss: 0.3995
Epoch 74/100
23/23            0s 1ms/step - loss:
0.4476 - val_loss: 0.3984
Epoch 75/100
23/23            0s 1ms/step - loss:
0.4339 - val_loss: 0.3983
Epoch 76/100
23/23            0s 1ms/step - loss:
0.4328 - val_loss: 0.3991
Epoch 77/100
23/23            0s 1ms/step - loss:
0.4251 - val_loss: 0.4034
Epoch 78/100
23/23            0s 1ms/step - loss:
0.4369 - val_loss: 0.4039
```

```
Epoch 79/100
23/23            0s 1ms/step - loss:
0.4325 - val_loss: 0.3988
Epoch 80/100
23/23            0s 1ms/step - loss:
0.4281 - val_loss: 0.3983
Epoch 81/100
23/23            0s 1ms/step - loss:
0.4307 - val_loss: 0.3973
Epoch 82/100
23/23            0s 1ms/step - loss:
0.4190 - val_loss: 0.3988
Epoch 83/100
23/23            0s 2ms/step - loss:
0.4415 - val_loss: 0.4017
Epoch 84/100
23/23            0s 1ms/step - loss:
0.4343 - val_loss: 0.3990
Epoch 85/100
23/23            0s 1ms/step - loss:
0.4359 - val_loss: 0.3988
Epoch 86/100
23/23            0s 1ms/step - loss:
0.4514 - val_loss: 0.3996
Epoch 87/100
23/23            0s 1ms/step - loss:
0.4270 - val_loss: 0.3976
Epoch 88/100
23/23            0s 1ms/step - loss:
0.4203 - val_loss: 0.3999
Epoch 89/100
23/23            0s 1ms/step - loss:
0.4399 - val_loss: 0.3984
Epoch 90/100
23/23            0s 1ms/step - loss:
0.4424 - val_loss: 0.4001
Epoch 91/100
23/23            0s 1ms/step - loss:
0.4330 - val_loss: 0.3988
Epoch 92/100
23/23            0s 1ms/step - loss:
0.4308 - val_loss: 0.3998
Epoch 93/100
23/23            0s 1ms/step - loss:
0.4336 - val_loss: 0.3985
Epoch 94/100
23/23            0s 1ms/step - loss:
0.4377 - val_loss: 0.3992
```

```
Epoch 95/100
23/23              0s 1ms/step - loss:
0.4516 - val_loss: 0.3990
Epoch 96/100
23/23              0s 1ms/step - loss:
0.4381 - val_loss: 0.3984
Epoch 97/100
23/23              0s 1ms/step - loss:
0.4355 - val_loss: 0.3979
Epoch 98/100
23/23              0s 1ms/step - loss:
0.4464 - val_loss: 0.3986
Epoch 99/100
23/23              0s 1ms/step - loss:
0.4402 - val_loss: 0.4001
Epoch 100/100
23/23              0s 1ms/step - loss:
0.4223 - val_loss: 0.3983
```

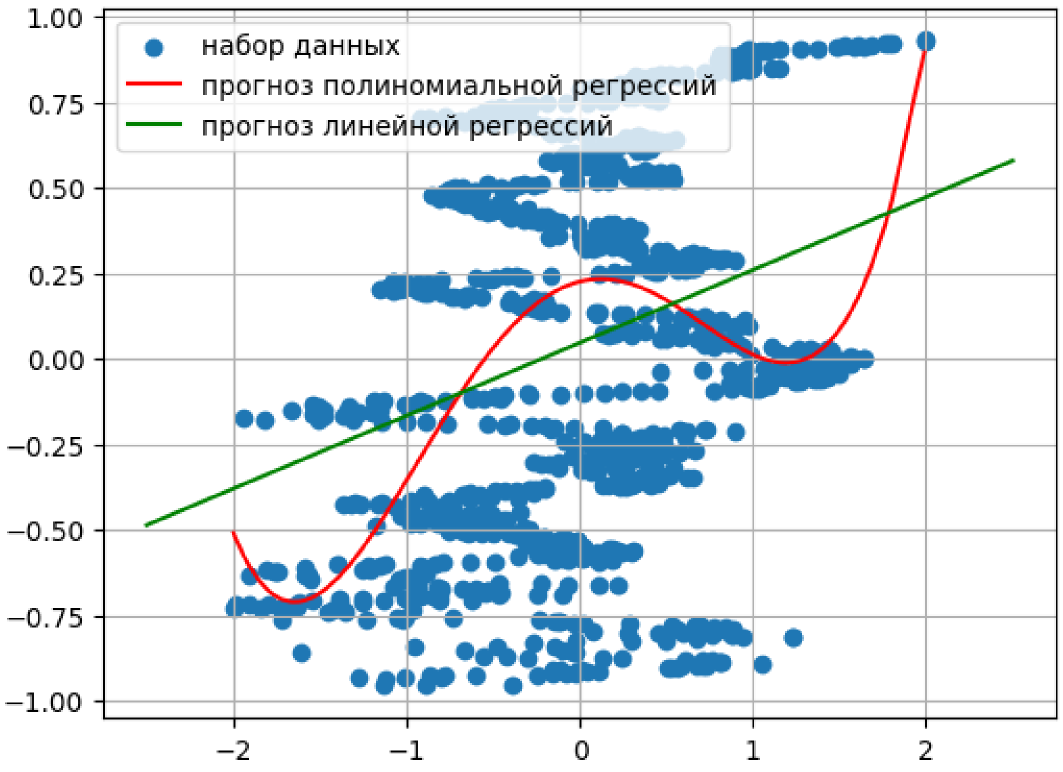
[200]:
```
#The fit method returns a history object, which typically has the 'loss' and␣
 ↪'val_loss' keys for the regression task.
#You can visualize your learning history using the following function:

def plot_loss(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    #plt.ylim([0, max(history.history['loss']*0.5)])
    plt.title('                    ')
    plt.xlabel('        ')
    plt.ylabel('        ')
    plt.legend(['         ', '         '], loc='upper right')
    plt.grid(True)
```

[201]:
```
plot_loss(history)
```



```
[209]: y_predict_reg2 = reg2_model.predict(X4)

plt.scatter(x2_out['temp_lower'], x2_out['year'], label='        ')
plt.plot(np.sort(x2_out['temp_lower']), y_predict_reg2[np.
  ↪argsort(x2_out['temp_lower'])], color='r', label='                    ')
plt.plot(plot_x, reg.predict(plot_x.reshape(-1,1)), color='g', label='          ␣
  ↪     ');
plt.legend(loc='upper left')
plt.grid();
```

```
33/33                 0s 375us/step
```



0.8 6. Define a feature in the original data set (different from the independent and dependent features) that takes continuous values and has the properties specified in the individual task. Add. feature: having minimal covariance with the independent variable temp_lower

[211]:
```
x2_out.cov()
```

[211]:
```
                 temp  temp_lower  temp_upper      year
temp         0.744762    0.474040    0.589334 -0.106223
temp_lower   0.474040    0.549550    0.162979  0.121716
temp_upper   0.589334    0.162979    0.647971 -0.246076
year        -0.106223    0.121716   -0.246076  0.283241
```

[212]:
```
# Yeaar has lowest cov with temp_lower but since it is our dependent variable␣
↪we will use the 2nd lowest independent feature which is temp_upper
x2_out
```

[212]:
```
          temp  temp_lower  temp_upper      year
0     0.479555   -0.398768    0.982236 -0.191937
1    -0.771665   -0.234056   -0.821329  0.400526
2    -0.500316   -0.492889   -0.236933 -0.109553
3    -0.666141   -0.269352   -0.660116 -0.390009
```

```
4     0.087606    0.248313   -0.095872  0.363716
…        …          …          …         …
1210  0.464480    0.836568   -0.085797  0.293602
1211 -0.651066   -0.151701   -0.750798  0.404032
1212  0.811203    1.154226    0.085492  0.903593
1213  0.012232    0.224783   -0.176478  0.375986
1214  1.399126    1.060105    0.962085 -0.028922

[1028 rows x 4 columns]
```

[217]:
```
#let's standardize independent feeature (We already did that)
x2_out
```

[217]:
```
          temp  temp_lower  temp_upper      year
0     0.479555   -0.398768    0.982236 -0.191937
1    -0.771665   -0.234056   -0.821329  0.400526
2    -0.500316   -0.492889   -0.236933 -0.109553
3    -0.666141   -0.269352   -0.660116 -0.390009
4     0.087606    0.248313   -0.095872  0.363716
…        …          …          …         …
1210  0.464480    0.836568   -0.085797  0.293602
1211 -0.651066   -0.151701   -0.750798  0.404032
1212  0.811203    1.154226    0.085492  0.903593
1213  0.012232    0.224783   -0.176478  0.375986
1214  1.399126    1.060105    0.962085 -0.028922

[1028 rows x 4 columns]
```

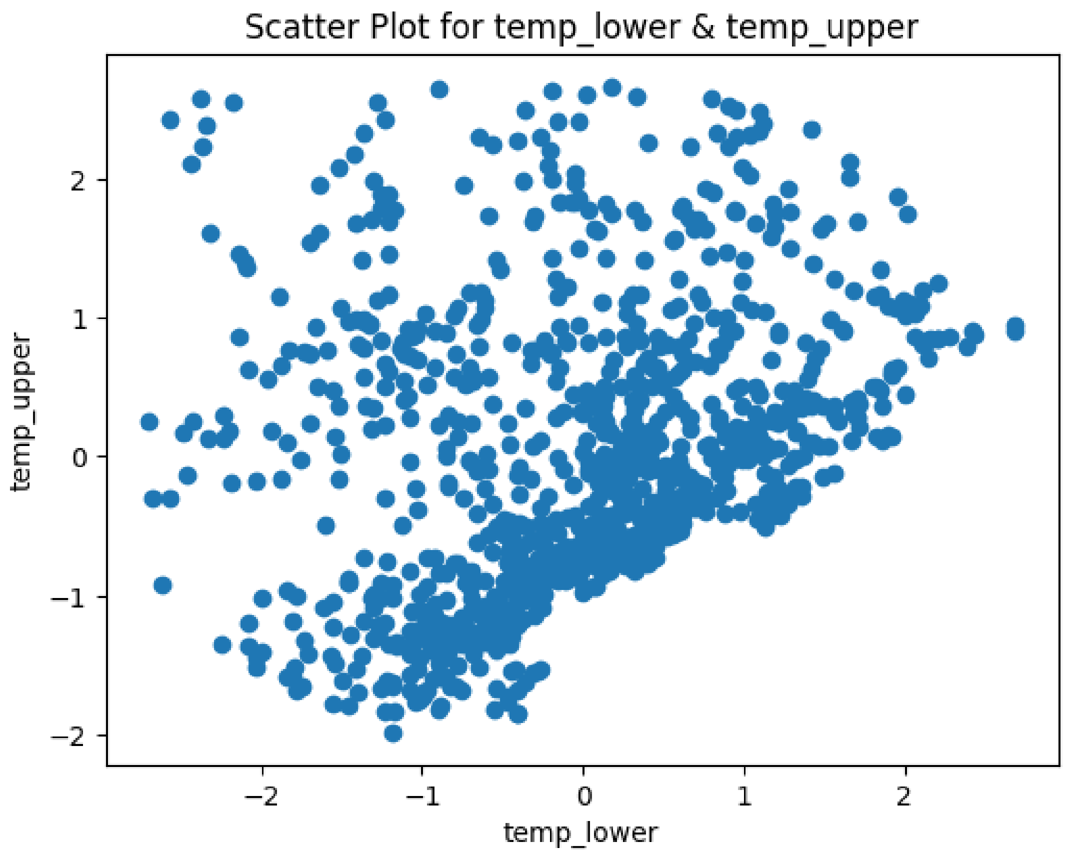
[218]:
```
np.mean(x2_out['temp_lower']), np.std(x2_out['temp_lower']), np.
 ↪mean(x2_out['temp_upper']), np.std(x2_out['temp_upper'])
```

[218]:
```
(-0.0015926871, 0.7409557, -0.15235747, 0.8045746)
```

[219]:
```
x_s, y_s = (x2_out['temp_lower'] - np.mean(x2_out['temp_lower']))/np.
 ↪std(x2_out['temp_lower']), (x2_out['temp_upper']- np.
 ↪mean(x2_out['temp_upper']))/np.std(x2_out['temp_upper'])
```

[222]:
```
plt.scatter(x_s, y_s)
plt.xlabel('temp_lower')  # Customize x-axis label
plt.ylabel('temp_upper')  # Customize y-axis label
plt.title('Scatter Plot for temp_lower & temp_upper')  # Customize y-axis label
np.mean(x_s), np.std(x_s), np.mean(y_s), np.std(y_s)
```

[222]:
```
(-5.5661924e-09, 1.0, -8.349288e-09, 1.0)
```



0.9 7. Standardize this feature and visualize it according to the individual task. Visualization additional characteristic – empirical distribution function

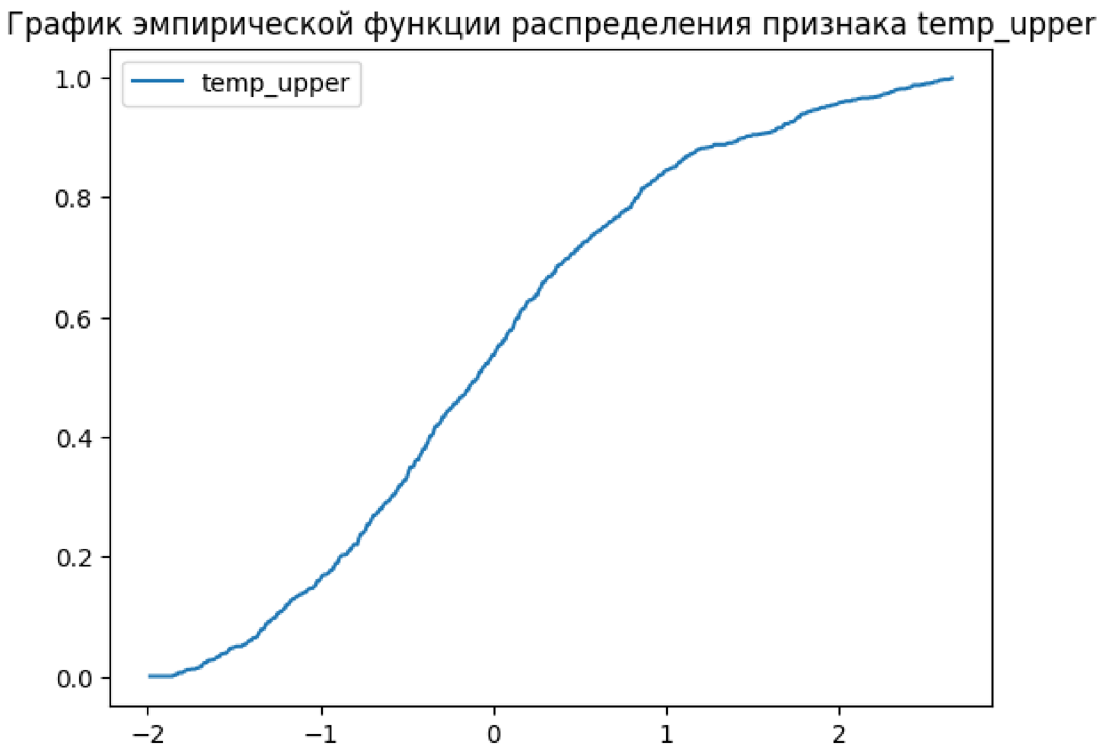
[223]:
```
def ECDF(data, x):
    counter = 0
    for v in data:
        if v <= x:
            counter += 1
    return counter / len(data)
```

[233]:
```
samples = y_s # upper temp
npoints = 500
dx = (samples.max()-samples.min())/npoints

xlist = [samples.min()+dx*i for i in range(npoints)]
ylist = [ECDF(samples, x) for x in xlist]
```

[234]:
```
df_ECDF = pd.DataFrame(ylist, columns=['temp_upper'],index=xlist)
df_ECDF
```

```
[234]:             temp_upper
       -1.983587    0.000973
       -1.974295    0.000973
       -1.965002    0.000973
       -1.955710    0.000973
       -1.946418    0.000973
       …                …
        2.616030    0.997082
        2.625322    0.997082
        2.634614    0.997082
        2.643906    0.998054
        2.653198    0.999027

       [500 rows x 1 columns]
```

```
[235]: df_ECDF.plot.line(title='                              temp_upper');
```



0.10 **8. Generate a set of input data from two standardized features of the data set (an independent feature and a specific feature), build a neural network (nonlinear regressor) with the number of hidden layers, the number of neurons and the activation function specified in the individual task, and one neuron in the output layer and train it on a data set of two features and a response. Monitor the training of the neural network, changing hyperparameters (loss function, optimizer, training step, etc.) as necessary or applying regularization.**

0.11 **9. Visualize the dataset as a scatterplot and the neural network prediction as a surface in 3D space, labeling the axes and drawing.**

0.12 **10. Split the two-feature-response data set into training and test sets and plot learning curves for a given quality metric as a function of the number of points in the training set, labeling the axes and figure and creating a legend.**

0.13 **Parameters of the deep neural network: number of hidden layers – 5, number of neurons in the hidden layer – 32, activation function – hyperbolic tangent.**

[236]:
```
x2_out.info
```

[236]:
```
<bound method DataFrame.info of            temp  temp_lower  temp_upper      year
0     0.479555   -0.398768    0.982236 -0.191937
1    -0.771665   -0.234056   -0.821329  0.400526
2    -0.500316   -0.492889   -0.236933 -0.109553
3    -0.666141   -0.269352   -0.660116 -0.390009
4     0.087606    0.248313   -0.095872  0.363716
…          …          …          …        …
1210  0.464480    0.836568   -0.085797  0.293602
1211 -0.651066   -0.151701   -0.750798  0.404032
1212  0.811203    1.154226    0.085492  0.903593
1213  0.012232    0.224783   -0.176478  0.375986
1214  1.399126    1.060105    0.962085 -0.028922

[1028 rows x 4 columns]>
```

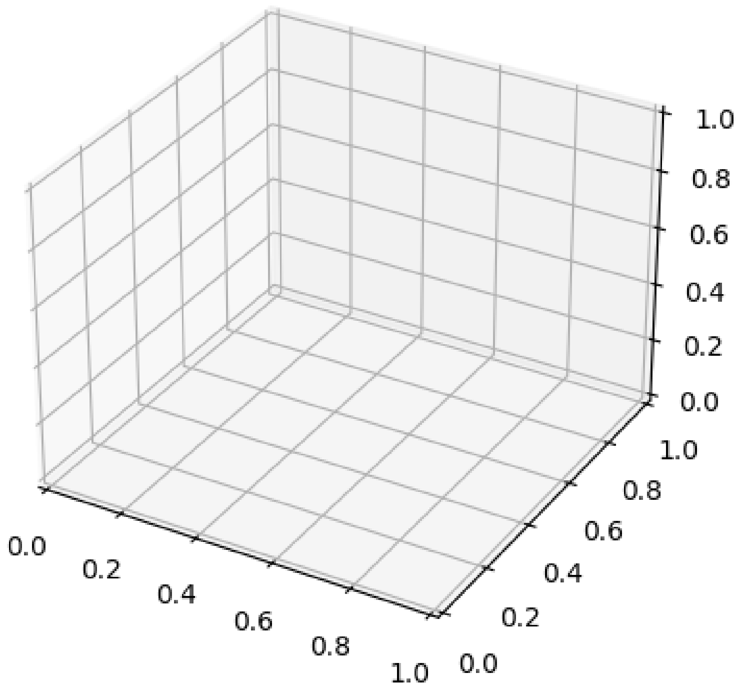
[241]:
```
#Let's study the dependence of temp_upper on temp and temp_lower
X = np.array(x2_out[['temp','temp_lower']])
y = np.array(x2_out[['temp_upper']]).reshape(-1)
```

[242]:
```
X.shape, y.shape
```

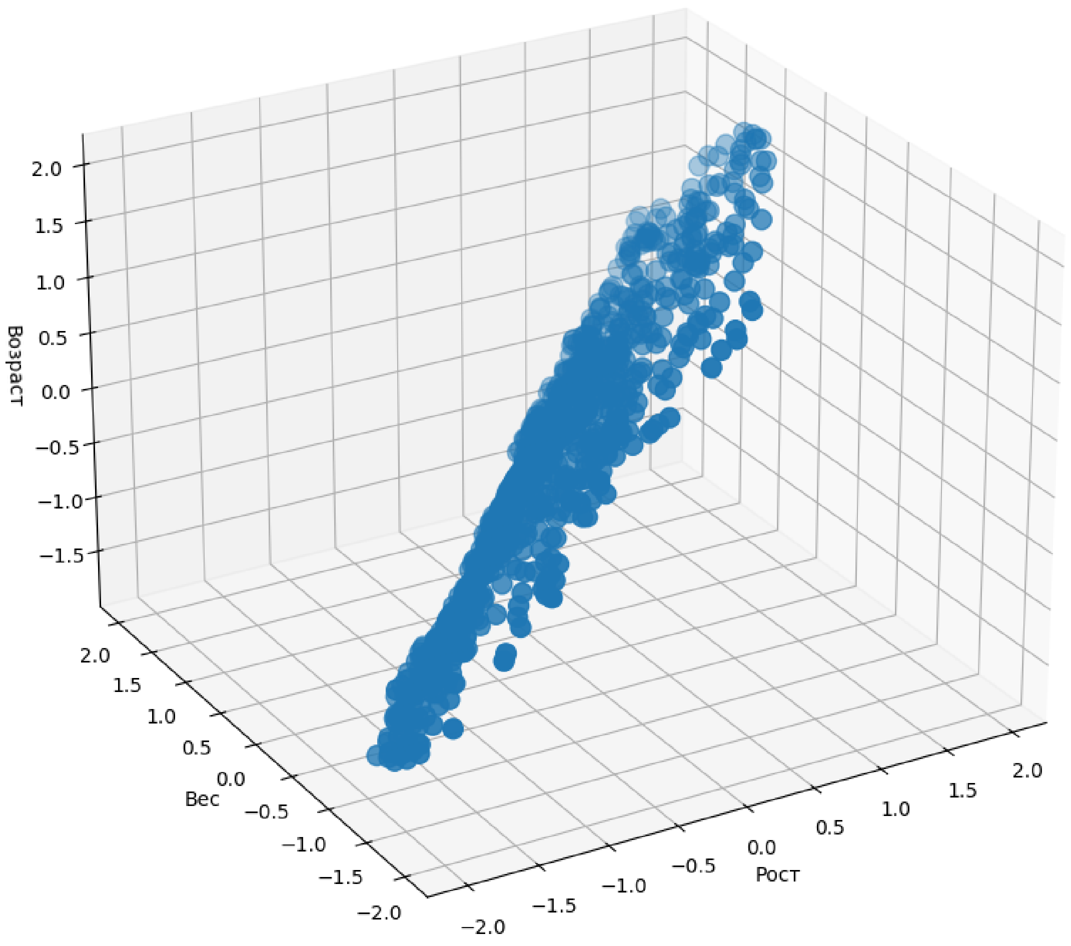
[242]:
```
((1028, 2), (1028,))
```

[243]:
```
from mpl_toolkits import mplot3d
```

[244]:
```
fig = plt.figure()
ax = plt.axes(projection='3d')
```



[245]:
```
fig = plt.figure(figsize=(12,10))
ax = plt.axes(projection='3d')

xs = X[:,0]
ys = X[:,1]
zs = y
```



```
[278]: ax.scatter( xs, ys, zs, s=100 )
       ax.set_xlabel('temp')
       ax.set_ylabel('temp_lower')
       ax.set_zlabel('temp_upper')
       ax.view_init( azim=-120, elev=25 )
```

```
[246]: feature_normalizer = tf.keras.layers.Normalization(axis=None,input_shape=(2,))
       feature_normalizer.adapt(X)
```

```
C:\Users\Mo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kf
ra8p0\LocalCache\local-packages\Python311\site-
packages\keras\src\layers\preprocessing\normalization.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
```

```
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(**kwargs)
```

[247]:
```
#Let's create a neural network with a normalization layer,
#five hidden dense layers with 64 neurons and an activation function ReLu and␣
 ↪an output layer of one neuron:
large_model = tf.keras.Sequential([
    feature_normalizer,
    tf.keras.layers.Dense(units=32, activation='tanh'),
    tf.keras.layers.Dense(units=32, activation='tanh'),
    tf.keras.layers.Dense(units=32, activation='tanh'),
    tf.keras.layers.Dense(units=32, activation='tanh'),
    tf.keras.layers.Dense(units=32, activation='tanh'),
    tf.keras.layers.Dense(units=1)
])

large_model.summary()
```

```
Model: "sequential_4"


 Layer (type)                           Output Shape                      ␣
 ↪Param #

 normalization (Normalization)          (None, 2)                            ␣
 ↪  3

 dense_4 (Dense)                        (None, 32)                           ␣
 ↪ 96

 dense_5 (Dense)                        (None, 32)                         ␣
 ↪1,056

 dense_6 (Dense)                        (None, 32)                         ␣
 ↪1,056

 dense_7 (Dense)                        (None, 32)                         ␣
 ↪1,056

 dense_8 (Dense)                        (None, 32)                         ␣
 ↪1,056

 dense_9 (Dense)                        (None, 1)                            ␣
 ↪ 33
```

```
Total params: 4,356 (17.02 KB)

Trainable params: 4,353 (17.00 KB)

Non-trainable params: 3 (16.00 B)
```

[248]:
```
#Let's compile the model using the root mean square error MSE with the default␣
↪optimizer (RmsProp) as the loss function:
large_model.compile(optimizer='adam', loss='mse', metrics=['mae', 'mse'])
```

[249]:
```
history = large_model.fit(
    X, y,
    epochs=100,
    #
    verbose=1,
    #       (     )    30%
    validation_split = 0.3)
```

```
Epoch 1/100
23/23              1s 6ms/step - loss:
0.2192 - mae: 0.3373 - mse: 0.2192 - val_loss: 0.0308 - val_mae: 0.1354 -
val_mse: 0.0308
Epoch 2/100
23/23              0s 1ms/step - loss:
0.0219 - mae: 0.1139 - mse: 0.0219 - val_loss: 0.0107 - val_mae: 0.0755 -
val_mse: 0.0107
Epoch 3/100
23/23              0s 1ms/step - loss:
0.0120 - mae: 0.0819 - mse: 0.0120 - val_loss: 0.0075 - val_mae: 0.0646 -
val_mse: 0.0075
Epoch 4/100
23/23              0s 1ms/step - loss:
0.0079 - mae: 0.0673 - mse: 0.0079 - val_loss: 0.0062 - val_mae: 0.0604 -
val_mse: 0.0062
Epoch 5/100
23/23              0s 1ms/step - loss:
0.0076 - mae: 0.0685 - mse: 0.0076 - val_loss: 0.0050 - val_mae: 0.0464 -
val_mse: 0.0050
Epoch 6/100
23/23              0s 1ms/step - loss:
0.0044 - mae: 0.0454 - mse: 0.0044 - val_loss: 0.0047 - val_mae: 0.0549 -
val_mse: 0.0047
Epoch 7/100
23/23              0s 1ms/step - loss:
0.0054 - mae: 0.0543 - mse: 0.0054 - val_loss: 0.0034 - val_mae: 0.0446 -
val_mse: 0.0034
```

```
Epoch 8/100
23/23           0s 1ms/step - loss:
0.0035 - mae: 0.0444 - mse: 0.0035 - val_loss: 0.0038 - val_mae: 0.0440 -
val_mse: 0.0038
Epoch 9/100
23/23           0s 1ms/step - loss:
0.0034 - mae: 0.0407 - mse: 0.0034 - val_loss: 0.0027 - val_mae: 0.0369 -
val_mse: 0.0027
Epoch 10/100
23/23           0s 1ms/step - loss:
0.0028 - mae: 0.0349 - mse: 0.0028 - val_loss: 0.0020 - val_mae: 0.0281 -
val_mse: 0.0020
Epoch 11/100
23/23           0s 1ms/step - loss:
0.0022 - mae: 0.0293 - mse: 0.0022 - val_loss: 0.0019 - val_mae: 0.0271 -
val_mse: 0.0019
Epoch 12/100
23/23           0s 1ms/step - loss:
0.0024 - mae: 0.0327 - mse: 0.0024 - val_loss: 0.0016 - val_mae: 0.0258 -
val_mse: 0.0016
Epoch 13/100
23/23           0s 1ms/step - loss:
0.0017 - mae: 0.0268 - mse: 0.0017 - val_loss: 0.0016 - val_mae: 0.0259 -
val_mse: 0.0016
Epoch 14/100
23/23           0s 1ms/step - loss:
0.0019 - mae: 0.0263 - mse: 0.0019 - val_loss: 0.0015 - val_mae: 0.0252 -
val_mse: 0.0015
Epoch 15/100
23/23           0s 1ms/step - loss:
0.0017 - mae: 0.0275 - mse: 0.0017 - val_loss: 0.0013 - val_mae: 0.0242 -
val_mse: 0.0013
Epoch 16/100
23/23           0s 1ms/step - loss:
0.0011 - mae: 0.0227 - mse: 0.0011 - val_loss: 0.0016 - val_mae: 0.0279 -
val_mse: 0.0016
Epoch 17/100
23/23           0s 1ms/step - loss:
0.0016 - mae: 0.0281 - mse: 0.0016 - val_loss: 9.8823e-04 - val_mae: 0.0224 -
val_mse: 9.8823e-04
Epoch 18/100
23/23           0s 1ms/step - loss:
0.0011 - mae: 0.0218 - mse: 0.0011 - val_loss: 0.0010 - val_mae: 0.0231 -
val_mse: 0.0010
Epoch 19/100
23/23           0s 1ms/step - loss:
0.0015 - mae: 0.0253 - mse: 0.0015 - val_loss: 0.0012 - val_mae: 0.0263 -
val_mse: 0.0012
```

```
Epoch 20/100
23/23            0s 1ms/step - loss:
0.0010 - mae: 0.0221 - mse: 0.0010 - val_loss: 9.0354e-04 - val_mae: 0.0224 -
val_mse: 9.0354e-04
Epoch 21/100
23/23            0s 1ms/step - loss:
9.4977e-04 - mae: 0.0215 - mse: 9.4977e-04 - val_loss: 8.5110e-04 - val_mae:
0.0189 - val_mse: 8.5110e-04
Epoch 22/100
23/23            0s 1ms/step - loss:
0.0012 - mae: 0.0242 - mse: 0.0012 - val_loss: 0.0011 - val_mae: 0.0238 -
val_mse: 0.0011
Epoch 23/100
23/23            0s 1ms/step - loss:
0.0012 - mae: 0.0252 - mse: 0.0012 - val_loss: 8.4827e-04 - val_mae: 0.0201 -
val_mse: 8.4827e-04
Epoch 24/100
23/23            0s 1ms/step - loss:
7.5670e-04 - mae: 0.0204 - mse: 7.5670e-04 - val_loss: 7.3643e-04 - val_mae:
0.0197 - val_mse: 7.3643e-04
Epoch 25/100
23/23            0s 1ms/step - loss:
8.3337e-04 - mae: 0.0207 - mse: 8.3337e-04 - val_loss: 0.0010 - val_mae: 0.0253
- val_mse: 0.0010
Epoch 26/100
23/23            0s 1ms/step - loss:
7.1752e-04 - mae: 0.0191 - mse: 7.1752e-04 - val_loss: 6.4017e-04 - val_mae:
0.0196 - val_mse: 6.4017e-04
Epoch 27/100
23/23            0s 1ms/step - loss:
5.2384e-04 - mae: 0.0175 - mse: 5.2384e-04 - val_loss: 5.1263e-04 - val_mae:
0.0165 - val_mse: 5.1263e-04
Epoch 28/100
23/23            0s 1ms/step - loss:
6.3875e-04 - mae: 0.0185 - mse: 6.3875e-04 - val_loss: 7.4943e-04 - val_mae:
0.0223 - val_mse: 7.4943e-04
Epoch 29/100
23/23            0s 1ms/step - loss:
7.4672e-04 - mae: 0.0211 - mse: 7.4672e-04 - val_loss: 0.0013 - val_mae: 0.0279
- val_mse: 0.0013
Epoch 30/100
23/23            0s 1ms/step - loss:
0.0010 - mae: 0.0259 - mse: 0.0010 - val_loss: 0.0015 - val_mae: 0.0302 -
val_mse: 0.0015
Epoch 31/100
23/23            0s 1ms/step - loss:
0.0015 - mae: 0.0287 - mse: 0.0015 - val_loss: 0.0011 - val_mae: 0.0264 -
val_mse: 0.0011
```

```
Epoch 32/100
23/23            0s 2ms/step - loss:
0.0013 - mae: 0.0273 - mse: 0.0013 - val_loss: 6.4309e-04 - val_mae: 0.0205 -
val_mse: 6.4309e-04
Epoch 33/100
23/23            0s 1ms/step - loss:
5.4746e-04 - mae: 0.0187 - mse: 5.4746e-04 - val_loss: 5.1972e-04 - val_mae:
0.0164 - val_mse: 5.1972e-04
Epoch 34/100
23/23            0s 3ms/step - loss:
4.0639e-04 - mae: 0.0146 - mse: 4.0639e-04 - val_loss: 6.1218e-04 - val_mae:
0.0191 - val_mse: 6.1218e-04
Epoch 35/100
23/23            0s 1ms/step - loss:
5.8373e-04 - mae: 0.0180 - mse: 5.8373e-04 - val_loss: 5.6846e-04 - val_mae:
0.0200 - val_mse: 5.6846e-04
Epoch 36/100
23/23            0s 2ms/step - loss:
4.4047e-04 - mae: 0.0158 - mse: 4.4047e-04 - val_loss: 3.5772e-04 - val_mae:
0.0143 - val_mse: 3.5772e-04
Epoch 37/100
23/23            0s 1ms/step - loss:
3.8652e-04 - mae: 0.0145 - mse: 3.8652e-04 - val_loss: 3.4476e-04 - val_mae:
0.0148 - val_mse: 3.4476e-04
Epoch 38/100
23/23            0s 1ms/step - loss:
3.1842e-04 - mae: 0.0133 - mse: 3.1842e-04 - val_loss: 9.1655e-04 - val_mae:
0.0231 - val_mse: 9.1655e-04
Epoch 39/100
23/23            0s 1ms/step - loss:
6.6599e-04 - mae: 0.0202 - mse: 6.6599e-04 - val_loss: 7.0069e-04 - val_mae:
0.0208 - val_mse: 7.0069e-04
Epoch 40/100
23/23            0s 1ms/step - loss:
5.3161e-04 - mae: 0.0178 - mse: 5.3161e-04 - val_loss: 6.2475e-04 - val_mae:
0.0194 - val_mse: 6.2475e-04
Epoch 41/100
23/23            0s 1ms/step - loss:
4.8710e-04 - mae: 0.0172 - mse: 4.8710e-04 - val_loss: 9.7940e-04 - val_mae:
0.0255 - val_mse: 9.7940e-04
Epoch 42/100
23/23            0s 1ms/step - loss:
7.2159e-04 - mae: 0.0199 - mse: 7.2159e-04 - val_loss: 3.6171e-04 - val_mae:
0.0143 - val_mse: 3.6171e-04
Epoch 43/100
23/23            0s 1ms/step - loss:
3.1453e-04 - mae: 0.0133 - mse: 3.1453e-04 - val_loss: 3.3076e-04 - val_mae:
0.0133 - val_mse: 3.3076e-04
```

```
Epoch 44/100
23/23            0s 1ms/step - loss:
2.5922e-04 - mae: 0.0122 - mse: 2.5922e-04 - val_loss: 2.9445e-04 - val_mae:
0.0136 - val_mse: 2.9445e-04
Epoch 45/100
23/23            0s 1ms/step - loss:
3.0029e-04 - mae: 0.0133 - mse: 3.0029e-04 - val_loss: 2.9925e-04 - val_mae:
0.0136 - val_mse: 2.9925e-04
Epoch 46/100
23/23            0s 1ms/step - loss:
3.1523e-04 - mae: 0.0142 - mse: 3.1523e-04 - val_loss: 3.1030e-04 - val_mae:
0.0132 - val_mse: 3.1030e-04
Epoch 47/100
23/23            0s 1ms/step - loss:
3.3529e-04 - mae: 0.0142 - mse: 3.3529e-04 - val_loss: 4.3256e-04 - val_mae:
0.0163 - val_mse: 4.3256e-04
Epoch 48/100
23/23            0s 1ms/step - loss:
3.6055e-04 - mae: 0.0148 - mse: 3.6055e-04 - val_loss: 3.1860e-04 - val_mae:
0.0142 - val_mse: 3.1860e-04
Epoch 49/100
23/23            0s 1ms/step - loss:
2.9913e-04 - mae: 0.0132 - mse: 2.9913e-04 - val_loss: 2.8793e-04 - val_mae:
0.0135 - val_mse: 2.8793e-04
Epoch 50/100
23/23            0s 1ms/step - loss:
3.1034e-04 - mae: 0.0139 - mse: 3.1034e-04 - val_loss: 2.5050e-04 - val_mae:
0.0125 - val_mse: 2.5050e-04
Epoch 51/100
23/23            0s 1ms/step - loss:
2.4905e-04 - mae: 0.0119 - mse: 2.4905e-04 - val_loss: 3.4984e-04 - val_mae:
0.0148 - val_mse: 3.4984e-04
Epoch 52/100
23/23            0s 1ms/step - loss:
2.7516e-04 - mae: 0.0128 - mse: 2.7516e-04 - val_loss: 1.8468e-04 - val_mae:
0.0107 - val_mse: 1.8468e-04
Epoch 53/100
23/23            0s 1ms/step - loss:
1.9806e-04 - mae: 0.0108 - mse: 1.9806e-04 - val_loss: 2.2955e-04 - val_mae:
0.0120 - val_mse: 2.2955e-04
Epoch 54/100
23/23            0s 1ms/step - loss:
2.3642e-04 - mae: 0.0117 - mse: 2.3642e-04 - val_loss: 3.2925e-04 - val_mae:
0.0142 - val_mse: 3.2925e-04
Epoch 55/100
23/23            0s 1ms/step - loss:
5.3876e-04 - mae: 0.0177 - mse: 5.3876e-04 - val_loss: 4.7937e-04 - val_mae:
0.0176 - val_mse: 4.7937e-04
```

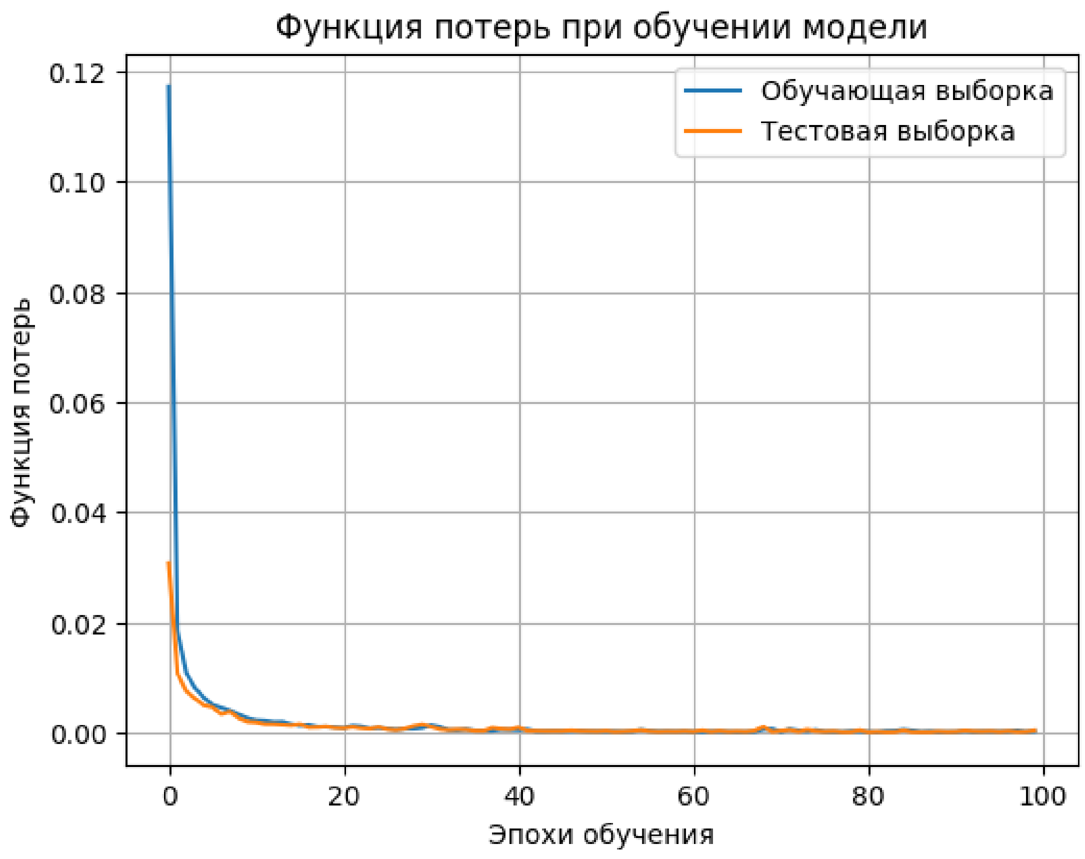
```
Epoch 56/100
23/23            0s 1ms/step - loss:
4.0945e-04 - mae: 0.0155 - mse: 4.0945e-04 - val_loss: 2.8159e-04 - val_mae:
0.0134 - val_mse: 2.8159e-04
Epoch 57/100
23/23            0s 1ms/step - loss:
2.9345e-04 - mae: 0.0132 - mse: 2.9345e-04 - val_loss: 1.9203e-04 - val_mae:
0.0112 - val_mse: 1.9203e-04
Epoch 58/100
23/23            0s 1ms/step - loss:
2.5262e-04 - mae: 0.0121 - mse: 2.5262e-04 - val_loss: 2.4303e-04 - val_mae:
0.0127 - val_mse: 2.4303e-04
Epoch 59/100
23/23            0s 1ms/step - loss:
2.1189e-04 - mae: 0.0113 - mse: 2.1189e-04 - val_loss: 2.1952e-04 - val_mae:
0.0116 - val_mse: 2.1952e-04
Epoch 60/100
23/23            0s 1ms/step - loss:
2.6010e-04 - mae: 0.0126 - mse: 2.6010e-04 - val_loss: 2.9055e-04 - val_mae:
0.0136 - val_mse: 2.9055e-04
Epoch 61/100
23/23            0s 1ms/step - loss:
1.9301e-04 - mae: 0.0112 - mse: 1.9301e-04 - val_loss: 2.0889e-04 - val_mae:
0.0113 - val_mse: 2.0889e-04
Epoch 62/100
23/23            0s 1ms/step - loss:
1.6035e-04 - mae: 0.0102 - mse: 1.6035e-04 - val_loss: 4.4198e-04 - val_mae:
0.0173 - val_mse: 4.4198e-04
Epoch 63/100
23/23            0s 1ms/step - loss:
2.8050e-04 - mae: 0.0132 - mse: 2.8050e-04 - val_loss: 1.8998e-04 - val_mae:
0.0105 - val_mse: 1.8998e-04
Epoch 64/100
23/23            0s 1ms/step - loss:
2.4182e-04 - mae: 0.0118 - mse: 2.4182e-04 - val_loss: 3.5032e-04 - val_mae:
0.0154 - val_mse: 3.5032e-04
Epoch 65/100
23/23            0s 1ms/step - loss:
2.1219e-04 - mae: 0.0115 - mse: 2.1219e-04 - val_loss: 1.9660e-04 - val_mae:
0.0113 - val_mse: 1.9660e-04
Epoch 66/100
23/23            0s 1ms/step - loss:
1.8638e-04 - mae: 0.0109 - mse: 1.8638e-04 - val_loss: 2.6213e-04 - val_mae:
0.0128 - val_mse: 2.6213e-04
Epoch 67/100
23/23            0s 1ms/step - loss:
1.9906e-04 - mae: 0.0111 - mse: 1.9906e-04 - val_loss: 1.9953e-04 - val_mae:
0.0114 - val_mse: 1.9953e-04
```

```
Epoch 68/100
23/23            0s 1ms/step - loss:
1.8131e-04 - mae: 0.0106 - mse: 1.8131e-04 - val_loss: 3.8750e-04 - val_mae:
0.0161 - val_mse: 3.8750e-04
Epoch 69/100
23/23            0s 1ms/step - loss:
5.1068e-04 - mae: 0.0179 - mse: 5.1068e-04 - val_loss: 0.0011 - val_mae: 0.0269
- val_mse: 0.0011
Epoch 70/100
23/23            0s 1ms/step - loss:
8.9525e-04 - mae: 0.0233 - mse: 8.9525e-04 - val_loss: 1.9035e-04 - val_mae:
0.0106 - val_mse: 1.9035e-04
Epoch 71/100
23/23            0s 1ms/step - loss:
1.7693e-04 - mae: 0.0106 - mse: 1.7693e-04 - val_loss: 3.6047e-04 - val_mae:
0.0153 - val_mse: 3.6047e-04
Epoch 72/100
23/23            0s 1ms/step - loss:
4.7724e-04 - mae: 0.0168 - mse: 4.7724e-04 - val_loss: 5.2851e-04 - val_mae:
0.0193 - val_mse: 5.2851e-04
Epoch 73/100
23/23            0s 1ms/step - loss:
4.4983e-04 - mae: 0.0167 - mse: 4.4983e-04 - val_loss: 2.1572e-04 - val_mae:
0.0117 - val_mse: 2.1572e-04
Epoch 74/100
23/23            0s 1ms/step - loss:
2.1886e-04 - mae: 0.0114 - mse: 2.1886e-04 - val_loss: 5.7098e-04 - val_mae:
0.0195 - val_mse: 5.7098e-04
Epoch 75/100
23/23            0s 1ms/step - loss:
6.0067e-04 - mae: 0.0194 - mse: 6.0067e-04 - val_loss: 3.1005e-04 - val_mae:
0.0145 - val_mse: 3.1005e-04
Epoch 76/100
23/23            0s 1ms/step - loss:
2.1247e-04 - mae: 0.0115 - mse: 2.1247e-04 - val_loss: 2.1282e-04 - val_mae:
0.0123 - val_mse: 2.1282e-04
Epoch 77/100
23/23            0s 1ms/step - loss:
2.0441e-04 - mae: 0.0114 - mse: 2.0441e-04 - val_loss: 2.9334e-04 - val_mae:
0.0147 - val_mse: 2.9334e-04
Epoch 78/100
23/23            0s 1ms/step - loss:
1.8842e-04 - mae: 0.0112 - mse: 1.8842e-04 - val_loss: 1.3672e-04 - val_mae:
0.0093 - val_mse: 1.3672e-04
Epoch 79/100
23/23            0s 1ms/step - loss:
2.5123e-04 - mae: 0.0121 - mse: 2.5123e-04 - val_loss: 1.3903e-04 - val_mae:
0.0092 - val_mse: 1.3903e-04
```

```
Epoch 80/100
23/23            0s 1ms/step - loss:
2.3482e-04 - mae: 0.0119 - mse: 2.3482e-04 - val_loss: 4.7740e-04 - val_mae:
0.0174 - val_mse: 4.7740e-04
Epoch 81/100
23/23            0s 1ms/step - loss:
2.5391e-04 - mae: 0.0120 - mse: 2.5391e-04 - val_loss: 1.1601e-04 - val_mae:
0.0086 - val_mse: 1.1601e-04
Epoch 82/100
23/23            0s 1ms/step - loss:
1.3747e-04 - mae: 0.0092 - mse: 1.3747e-04 - val_loss: 1.8859e-04 - val_mae:
0.0108 - val_mse: 1.8859e-04
Epoch 83/100
23/23            0s 1ms/step - loss:
2.9175e-04 - mae: 0.0130 - mse: 2.9175e-04 - val_loss: 1.1910e-04 - val_mae:
0.0087 - val_mse: 1.1910e-04
Epoch 84/100
23/23            0s 1ms/step - loss:
2.8812e-04 - mae: 0.0132 - mse: 2.8812e-04 - val_loss: 1.3925e-04 - val_mae:
0.0095 - val_mse: 1.3925e-04
Epoch 85/100
23/23            0s 1ms/step - loss:
3.9971e-04 - mae: 0.0152 - mse: 3.9971e-04 - val_loss: 4.4842e-04 - val_mae:
0.0169 - val_mse: 4.4842e-04
Epoch 86/100
23/23            0s 1ms/step - loss:
4.7642e-04 - mae: 0.0171 - mse: 4.7642e-04 - val_loss: 1.3812e-04 - val_mae:
0.0095 - val_mse: 1.3812e-04
Epoch 87/100
23/23            0s 1ms/step - loss:
1.7043e-04 - mae: 0.0103 - mse: 1.7043e-04 - val_loss: 1.2099e-04 - val_mae:
0.0086 - val_mse: 1.2099e-04
Epoch 88/100
23/23            0s 1ms/step - loss:
2.0408e-04 - mae: 0.0110 - mse: 2.0408e-04 - val_loss: 1.2620e-04 - val_mae:
0.0092 - val_mse: 1.2620e-04
Epoch 89/100
23/23            0s 2ms/step - loss:
1.7237e-04 - mae: 0.0101 - mse: 1.7237e-04 - val_loss: 2.4315e-04 - val_mae:
0.0126 - val_mse: 2.4315e-04
Epoch 90/100
23/23            0s 1ms/step - loss:
1.8342e-04 - mae: 0.0106 - mse: 1.8342e-04 - val_loss: 1.5849e-04 - val_mae:
0.0102 - val_mse: 1.5849e-04
Epoch 91/100
23/23            0s 2ms/step - loss:
1.4322e-04 - mae: 0.0094 - mse: 1.4322e-04 - val_loss: 2.0052e-04 - val_mae:
0.0114 - val_mse: 2.0052e-04
```

```
Epoch 92/100
23/23              0s 2ms/step - loss:
3.3888e-04 - mae: 0.0137 - mse: 3.3888e-04 - val_loss: 3.8467e-04 - val_mae:
0.0158 - val_mse: 3.8467e-04
Epoch 93/100
23/23              0s 2ms/step - loss:
2.3008e-04 - mae: 0.0121 - mse: 2.3008e-04 - val_loss: 2.7435e-04 - val_mae:
0.0136 - val_mse: 2.7435e-04
Epoch 94/100
23/23              0s 2ms/step - loss:
3.0665e-04 - mae: 0.0138 - mse: 3.0665e-04 - val_loss: 2.3194e-04 - val_mae:
0.0123 - val_mse: 2.3194e-04
Epoch 95/100
23/23              0s 2ms/step - loss:
2.1181e-04 - mae: 0.0114 - mse: 2.1181e-04 - val_loss: 2.9439e-04 - val_mae:
0.0138 - val_mse: 2.9439e-04
Epoch 96/100
23/23              0s 1ms/step - loss:
2.4060e-04 - mae: 0.0124 - mse: 2.4060e-04 - val_loss: 1.7552e-04 - val_mae:
0.0108 - val_mse: 1.7552e-04
Epoch 97/100
23/23              0s 1ms/step - loss:
1.7743e-04 - mae: 0.0102 - mse: 1.7743e-04 - val_loss: 3.0660e-04 - val_mae:
0.0142 - val_mse: 3.0660e-04
Epoch 98/100
23/23              0s 1ms/step - loss:
3.7806e-04 - mae: 0.0153 - mse: 3.7806e-04 - val_loss: 1.7403e-04 - val_mae:
0.0111 - val_mse: 1.7403e-04
Epoch 99/100
23/23              0s 1ms/step - loss:
1.9591e-04 - mae: 0.0112 - mse: 1.9591e-04 - val_loss: 1.8070e-04 - val_mae:
0.0108 - val_mse: 1.8070e-04
Epoch 100/100
23/23              0s 1ms/step - loss:
2.8745e-04 - mae: 0.0137 - mse: 2.8745e-04 - val_loss: 4.3239e-04 - val_mae:
0.0175 - val_mse: 4.3239e-04
```

[250]:
```
plot_loss(history)
```



```
[251]: n_plot = 51

       x_plot = np.linspace(np.min(xs), np.max(xs), n_plot)
       y_plot = np.linspace(np.min(ys), np.max(ys), n_plot)
```

```
[252]: x_mesh, y_mesh = np.meshgrid(x_plot, y_plot)
       x_mesh.shape, y_mesh.shape
```

```
[252]: ((51, 51), (51, 51))
```

```
[253]: x_plot2 = np.reshape(x_mesh, [n_plot**2,1])
       y_plot2 = np.reshape(y_mesh, [n_plot**2,1])
       xy_2 = np.hstack([x_plot2, y_plot2])
       xy_2.shape
```

```
[253]: (2601, 2)
```

```
[254]: z = large_model.predict(xy_2)
       z.shape
```

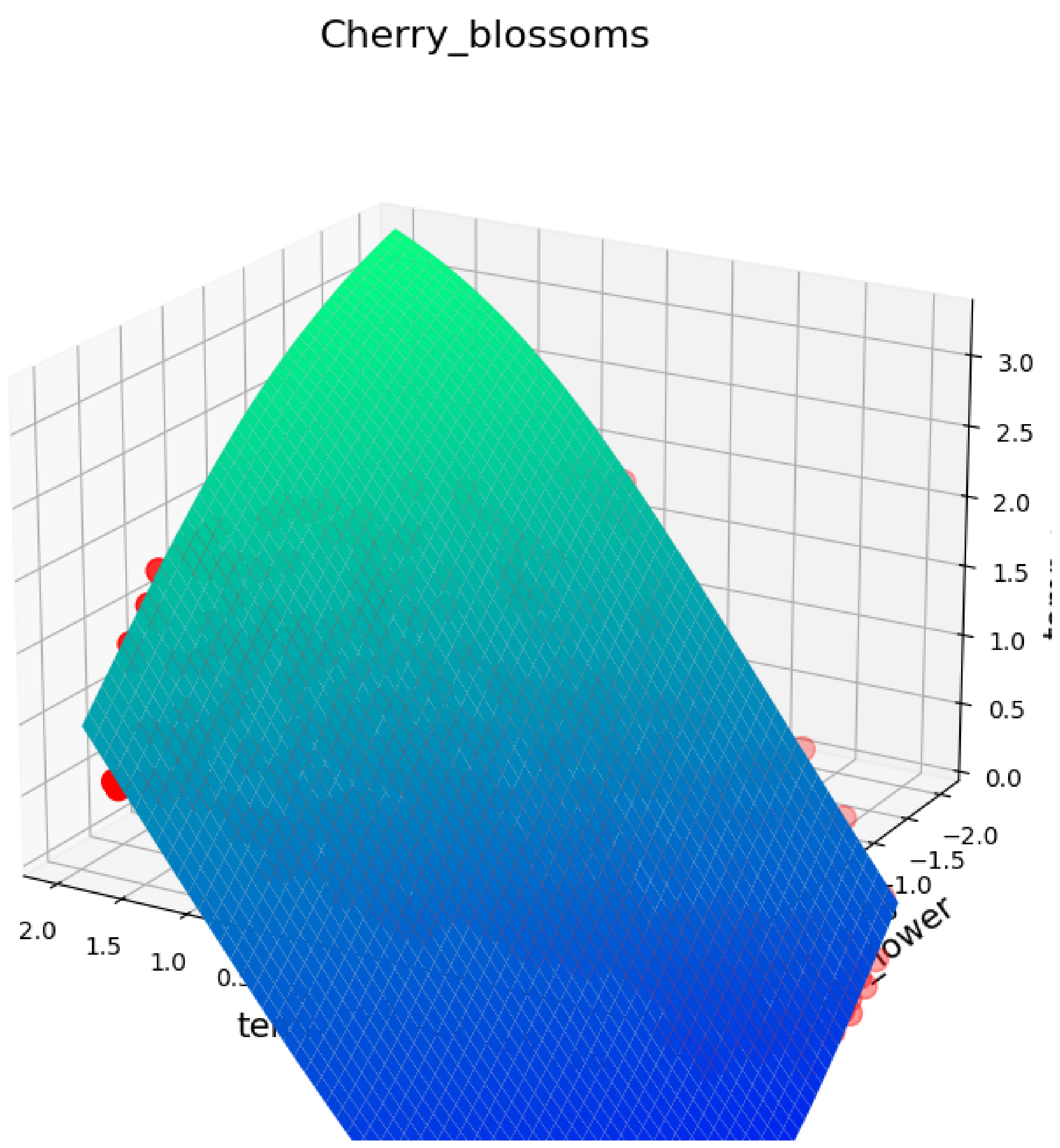
```
82/82                 0s 817us/step
```

```
[254]: (2601, 1)
```

```
[255]: z_mesh = z.reshape((n_plot, n_plot))
       z_mesh.shape
```

```
[255]: (51, 51)
```

```
[258]: from matplotlib import cm

       fig = plt.figure(figsize=(16, 8))
       ax = fig.add_subplot(111, projection='3d')

       surf = ax.plot_surface(x_mesh, y_mesh, z_mesh, \
              rstride=1, cstride=1, linewidth=0.05, cmap=cm.winter, antialiased=True, \
              edgecolors='gray')
       ax.scatter( xs, ys, zs, s=100, c='r' )

       ax.set_xlabel('temp', fontsize=14)
       ax.set_ylabel('temp_lower', fontsize=14)
       ax.set_zlabel('temp_upper', fontsize=14)
       ax.set_title('Cherry_blossoms', fontsize=16)

       ax.set_zlim(0., z_mesh.max())
       ax.view_init(elev = 20, azim = 120)
```



```
[259]: #Learning curves
       def train_test_split(X, y, test_ratio=0.2, seed=None):
           """       X_train, X_test, y_train, y_test"""
           assert X.shape[0] == y.shape[0], \
               "    X                 y"
           assert 0.0 <= test_ratio <= 1.0, \
               "          test_ratio"

           if seed:
               np.random.seed(seed)

           shuffled_indexes = np.random.permutation(len(X))
```

```
    test_size = int(len(X) * test_ratio)
    test_indexes = shuffled_indexes[:test_size]
    train_indexes = shuffled_indexes[test_size:]

    X_train = X[train_indexes]
    y_train = y[train_indexes]

    X_test = X[test_indexes]
    y_test = y[test_indexes]

    return X_train, X_test, y_train, y_test
```

[260]:
```
#Let's split the data arrays `X` and `y` into training and test data:
X_train, X_test, y_train, y_test = train_test_split(X, y, 0.3)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[260]:
```
((720, 2), (308, 2), (720,), (308,))
```

[274]:
```
#We will use the MSE indicator for visualization
def my_mse(y_test, y_predict):
    return np.sum((y_predict - y_test)**2) / len(y_test)
```

[275]:
```
#                720    ,       11                   10                  :
train_score = []
test_score = []
for i in range(10, 720, 10):
    large_model = tf.keras.Sequential([
        feature_normalizer,
        tf.keras.layers.Dense(units=32, activation='tanh'),
        tf.keras.layers.Dense(units=32, activation='tanh'),
        tf.keras.layers.Dense(units=32, activation='tanh'),
        tf.keras.layers.Dense(units=32, activation='tanh'),
        tf.keras.layers.Dense(units=32, activation='tanh'),
        tf.keras.layers.Dense(units=1)
])
    large_model.compile(loss='mse')
    large_model.fit(X_train[:i], y_train[:i], epochs=50, verbose=0)

    y_train_predict = large_model.predict(X_train[:i])
    train_score.append(my_mse(y_train[:i], y_train_predict))

    y_test_predict = large_model.predict(X_test)
    test_score.append(my_mse(y_test, y_test_predict))
    print('-->', i, ' done')
```

```
1/1             0s 41ms/step
10/10             0s 555us/step
```

```
--> 10  done
1/1             0s 40ms/step
10/10             0s 555us/step
--> 20  done
1/1             0s 39ms/step
10/10             0s 555us/step
--> 30  done
2/2             0s 32ms/step
10/10             0s 445us/step
--> 40  done
2/2             0s 33ms/step
10/10             0s 555us/step
--> 50  done
2/2             0s 31ms/step
10/10             0s 444us/step
--> 60  done
3/3             0s 16ms/step
10/10             0s 444us/step
--> 70  done
3/3             0s 16ms/step
10/10             0s 556us/step
--> 80  done
3/3             0s 17ms/step
10/10             0s 556us/step
--> 90  done
4/4             0s 12ms/step
10/10             0s 444us/step
--> 100  done
4/4             0s 12ms/step
10/10             0s 556us/step
--> 110  done
4/4             0s 13ms/step
10/10             0s 556us/step
--> 120  done
5/5             0s 9ms/step
10/10             0s 556us/step
--> 130  done
5/5             0s 9ms/step
10/10             0s 556us/step
--> 140  done
5/5             0s 9ms/step
10/10             0s 571us/step
--> 150  done
5/5             0s 750us/step
10/10             0s 4ms/step
--> 160  done
6/6             0s 7ms/step
10/10             0s 556us/step
```

```
--> 170  done
6/6             0s 7ms/step
10/10             0s 556us/step
--> 180  done
6/6             0s 8ms/step
10/10             0s 555us/step
--> 190  done
7/7             0s 9ms/step
10/10             0s 555us/step
--> 200  done
7/7             0s 7ms/step
10/10             0s 445us/step
--> 210  done
7/7             0s 6ms/step
10/10             0s 556us/step
--> 220  done
8/8             0s 6ms/step
10/10             0s 543us/step
--> 230  done
8/8             0s 5ms/step
10/10             0s 556us/step
--> 240  done
8/8             0s 5ms/step
10/10             0s 556us/step
--> 250  done
9/9             0s 5ms/step
10/10             0s 667us/step
--> 260  done
9/9             0s 5ms/step
10/10             0s 556us/step
--> 270  done
9/9             0s 5ms/step
10/10             0s 556us/step
--> 280  done
10/10             0s 4ms/step
10/10             0s 556us/step
--> 290  done
10/10             0s 4ms/step
10/10             0s 667us/step
--> 300  done
10/10             0s 4ms/step
10/10             0s 555us/step
--> 310  done
10/10             0s 556us/step
10/10             0s 4ms/step
--> 320  done
11/11             0s 4ms/step
10/10             0s 501us/step
```

```
--> 330  done
11/11           0s 4ms/step
10/10           0s 556us/step
--> 340  done
11/11           0s 4ms/step
10/10           0s 555us/step
--> 350  done
12/12           0s 3ms/step
10/10           0s 556us/step
--> 360  done
12/12           0s 3ms/step
10/10           0s 444us/step
--> 370  done
12/12           0s 3ms/step
10/10           0s 556us/step
--> 380  done
13/13           0s 3ms/step
10/10           0s 555us/step
--> 390  done
13/13           0s 3ms/step
10/10           0s 556us/step
--> 400  done
13/13           0s 3ms/step
10/10           0s 556us/step
--> 410  done
14/14           0s 3ms/step
10/10           0s 556us/step
--> 420  done
14/14           0s 3ms/step
10/10           0s 555us/step
--> 430  done
14/14           0s 3ms/step
10/10           0s 556us/step
--> 440  done
15/15           0s 3ms/step
10/10           0s 556us/step
--> 450  done
15/15           0s 3ms/step
10/10           0s 556us/step
--> 460  done
15/15           0s 3ms/step
10/10           0s 445us/step
--> 470  done
15/15           0s 429us/step
10/10           0s 4ms/step
--> 480  done
16/16           0s 2ms/step
10/10           0s 444us/step
```

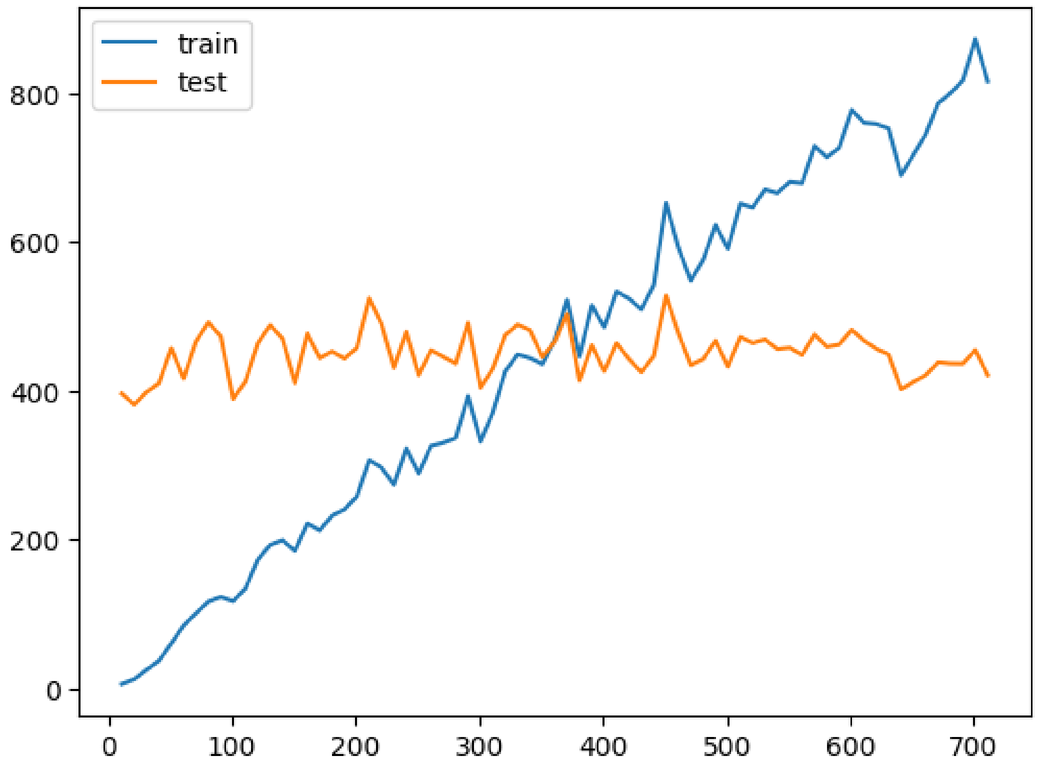
```
--> 490  done
16/16             0s 3ms/step
10/10             0s 445us/step
--> 500  done
16/16             0s 3ms/step
10/10             0s 556us/step
--> 510  done
17/17             0s 2ms/step
10/10             0s 444us/step
--> 520  done
17/17             0s 2ms/step
10/10             0s 444us/step
--> 530  done
17/17             0s 2ms/step
10/10             0s 445us/step
--> 540  done
18/18             0s 2ms/step
10/10             0s 556us/step
--> 550  done
18/18             0s 2ms/step
10/10             0s 444us/step
--> 560  done
18/18             0s 3ms/step
10/10             0s 445us/step
--> 570  done
19/19             0s 2ms/step
10/10             0s 444us/step
--> 580  done
19/19             0s 2ms/step
10/10             0s 444us/step
--> 590  done
19/19             0s 2ms/step
10/10             0s 444us/step
--> 600  done
20/20             0s 2ms/step
10/10             0s 444us/step
--> 610  done
20/20             0s 2ms/step
10/10             0s 506us/step
--> 620  done
20/20             0s 2ms/step
10/10             0s 444us/step
--> 630  done
20/20             0s 421us/step
10/10             0s 4ms/step
--> 640  done
21/21             0s 2ms/step
10/10             0s 444us/step
```

```
--> 650  done
21/21              0s 2ms/step
10/10              0s 444us/step
--> 660  done
21/21              0s 2ms/step
10/10              0s 444us/step
--> 670  done
22/22              0s 2ms/step
10/10              0s 444us/step
--> 680  done
22/22              0s 2ms/step
10/10              0s 556us/step
--> 690  done
22/22              0s 2ms/step
10/10              0s 556us/step
--> 700  done
23/23              0s 2ms/step
10/10              0s 444us/step
--> 710  done
```

[276]:
```
test_score
```

[276]:
```
[396.63643973214283,
 381.7247362012987,
 398.6898843344156,
 410.34448559253246,
 457.62560876623377,
 416.82142857142856,
 466.97955560064935,
 492.44617491883116,
 473.68075284090907,
 389.42291497564935,
 412.93039772727275,
 464.60480925324674,
 488.8656655844156,
 470.41700487012986,
 410.6696174918831,
 477.36414366883116,
 444.23584618506493,
 453.36399147727275,
 443.6841517857143,
 458.11820211038963,
 525.2320921266233,
 490.33203125,
 431.0861911525974,
 480.20393668831167,
 421.45769074675326,
454.86840503246754,
445.9845271915584,
437.0233867694805,
491.847554788961,
404.54791497564935,
430.4549005681818,
475.35049715909093,
489.45662540584414,
481.9957893668831,
445.9104099025974,
466.47402597402595,
503.81808035714283,
414.2267907873377,
461.69881290584414,
426.9865564123377,
464.44805194805195,
443.0960328733766,
425.5003804788961,
446.97367086038963,
528.6191152597403,
477.53865665584414,
434.97615665584414,
443.03596793831167,
467.76171875,
432.80078125,
472.9409496753247,
464.65711241883116,
469.49137581168833,
456.37987012987014,
458.31523944805195,
448.44921875,
476.3925020292208,
459.54043222402595,
462.73549107142856,
482.5932426948052,
467.9728084415584,
456.8206168831169,
449.0314021915584,
402.3925020292208,
412.2326247970779,
421.6662692775974,
438.77490868506493,
437.16512784090907,
436.91649756493507,
455.17116477272725,
421.54558137175326]
```

```
[277]: plt.plot([i for i in range(11, len(X_train)+1, 10)],
                                      train_score, label="train")
       plt.plot([i for i in range(11, len(X_train)+1, 10)],
                                      test_score, label="test")
       plt.legend();
```



```
[ ]:
```