Eraste Boko

Machine Learning: Multilayer Perceptron Regression

Neural Networks for Regression Tasks

Dataset: Housing

•Import the data from the provided "housing.csv" file and pre-process it by applying one hot encoding to replace the categorical features with binary features and by replacing NaN values with the mean value from that column. Do not apply feature expansion; this is a technique we use when trying to supply our linear models with additional data but not

neural networks. For a neural network, we will attempt other solutions.

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display
```

In [2]:
```
# import the Multilayer Perceptron Regression package
from sklearn.neural_network import MLPRegressor
```

• Create your training and test set based on your cleansed and pre-processed data. Use this same training and test data consistently in all of your experiments.

• As a first attempt at building a model, create and train a multilayer perceptron neural network using its default values to try to predict the median house value. Here and throughout your notebook, after you fit your model to the training data, print out the model's accuracy on both the training set and the test set.

Remember to increase the "max_iter" parameter from the default if needed and that, here and throughout, models that do not converge should be treated as failed runs.

In [3]:
```
housing = pd.read_csv("housing.csv")
housing_input = housing.drop("median_house_value", axis=1)
housing_output = housing["median_house_value"].copy()
encoded_input = pd.get_dummies(housing_input)
cleaned_input = encoded_input.fillna(encoded_input.mean())
```

In [4]:
```
print("cleaned_input shape: ", cleaned_input.shape)
encoded_input.head()
```

```
cleaned_input shape:  (20640, 13)
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0

In [5]:
```
# create a multilayer perceptron regressor
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(cleaned_input, housing_output,
mlp = MLPRegressor(max_iter=3000,random_state=42)

# fit the regressor to the training data
mlp.fit(X_train, y_train)

# print its accuracy on the training set
print("Accuracy on training set: ", mlp.score(X_train, y_train))

# print its accuracy on the test set
print("Accuracy on test set: ", mlp.score(X_test, y_test))
```

```
Accuracy on training set:  0.6935422269035172
Accuracy on test set:  0.6826178247440557
```

•Next, rescale your features so they all have a mean of 0 and a variance of 1. Create and train a new MLPRegressor model. Observe that, even with a reasonably large value for "max_iter", your model is unlikely to converge now

In [6]:
```
# compute the mean value for each feature of the training set
mean_on_train = X_train.mean(axis=0)

# compute the standard deviation for each feature of the training set
std_on_train = X_train.std(axis=0)

# apply the same transformation to both the training and test set
X_train_scaled = (X_train - mean_on_train)/std_on_train
X_test_scaled = (X_test - mean_on_train)/std_on_train
```

•Review the scikit learn documentation at https://scikit-learn.org/ for the MLPRegressor model and observe that the default MLPRegressor has a single hidden layer of 100 units. In order to train a neural network of this shape to learn our data, it is likely that you will need to experiment with both the "alpha" regularization factor (which we have worked with in class)

and the "learning_rate_init" parameter (which is new to this assignment). Check the documentation for the default values for both of these

parameters and identify a setting for both of these parameters, as well as the "max_iter" parameter that does allow convergence.

Note that randomly changing these values will not be as effective as researching and making sure you understand the role of these parameters and then considering what changes may help.

In [7]:
```
mlp = MLPRegressor(max_iter=6000,alpha=0.001,learning_rate_init=0.01, hidden_layer_
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
```

```
Accuracy on training set:  0.8243325520752796
Accuracy on test set:  0.7936743196561722
```

In [8]:
```
mlp = MLPRegressor(max_iter=6000,alpha=0.001,learning_rate_init=0.01, hidden_layer_
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
```

```
Accuracy on training set:  0.8255529122847771
Accuracy on test set:  0.7892249972588324
```

In [9]:
```
mlp = MLPRegressor(max_iter=6000,alpha=0.001,learning_rate_init=0.01, hidden_layer_
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
```

```
Accuracy on training set:  0.8107744409240205
Accuracy on test set:  0.7839434760860771
```

In [10]:
```
mlp = MLPRegressor(max_iter=6000,alpha=0.001,learning_rate_init=0.01, hidden_layer_
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
```

```
Accuracy on training set:  0.820708366360485
Accuracy on test set:  0.7919947526400402
```

• Proceed to experiment with the following possible changes to the shape of your neural network. Start by using the values for the max_iter, learning_rate_init and alpha parameters that you have found allow you to reach convergence for a default network with a single layer of 100 hidden units, but make adjustments as needed to be able to convince

yourself you understand whether these different neural network shapes produce better quality models or not.

o Maintain a single hidden layer and explore the impact of increasing or decreasing the number of units in that single layer.

o Add a second hidden layer and explore the impact of having a larger first layer versus a smaller first layer and how many units seem to be needed in these layers.

o Explore whether adding a third hidden layer can be used to improve the quality of the learned model

In [11]:
```
max_iter=[3000, 5000, 8000, 10000]
```

In [12]:
```
alpha=[0.001, 0.01, 0.1, 1, 10]
```

In [13]:
```
learning_rate_init=[ 0.01, 0.1, 1, 10]
```

In [14]:
```
for mi in max_iter:
    mlp = MLPRegressor(max_iter=mi,alpha=0.001,learning_rate_init=0.01, hidden_laye
    mlp.fit(X_train_scaled, y_train)

    train_accuracy = mlp.score(X_train_scaled, y_train)
    test_accuracy = mlp.score(X_test_scaled, y_test)
    print("max_iter:", mi)
    print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
    print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
    print()
```

```
max_iter: 3000
Accuracy on training set:  0.8255529122847771
Accuracy on test set:  0.7892249972588324

max_iter: 5000
Accuracy on training set:  0.8255529122847771
Accuracy on test set:  0.7892249972588324

max_iter: 8000
Accuracy on training set:  0.8255529122847771
Accuracy on test set:  0.7892249972588324

max_iter: 10000
Accuracy on training set:  0.8255529122847771
Accuracy on test set:  0.7892249972588324
```

In [15]:
```
for a in alpha:
    mlp = MLPRegressor(max_iter=6000,alpha=a,learning_rate_init=0.01, hidden_layer_
    mlp.fit(X_train_scaled, y_train)


    train_accuracy = mlp.score(X_train_scaled, y_train)
    test_accuracy = mlp.score(X_test_scaled, y_test)
    print("alpha:", a)
    print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
    print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
    print()
```

```
alpha: 0.001
Accuracy on training set:  0.8255529122847771
Accuracy on test set:  0.7892249972588324

alpha: 0.01
Accuracy on training set:  0.8166510868001509
Accuracy on test set:  0.7790189458213668

alpha: 0.1
Accuracy on training set:  0.8249492733280843
Accuracy on test set:  0.7902282537508376

alpha: 1
Accuracy on training set:  0.8415514292417421
Accuracy on test set:  0.7973070278159309

alpha: 10
Accuracy on training set:  0.8206360215971069
Accuracy on test set:  0.7889406347016861
```

In [16]:
```
for lri  in learning_rate_init:
    mlp = MLPRegressor(max_iter=6000,alpha=0.001,learning_rate_init=lri, hidden_lay
    mlp.fit(X_train_scaled, y_train)

    train_accuracy = mlp.score(X_train_scaled, y_train)
    test_accuracy = mlp.score(X_test_scaled, y_test)
    print("learning_rate_init:", lri)
    print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
    print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
    print()
```

```
learning_rate_init: 0.01
Accuracy on training set:  0.8255529122847771
Accuracy on test set:  0.7892249972588324

learning_rate_init: 0.1
Accuracy on training set:  0.7046575622896709
Accuracy on test set:  0.6959159502863956

learning_rate_init: 1
Accuracy on training set:  0.7921353634682831
Accuracy on test set:  0.7707740659436444

learning_rate_init: 10
Accuracy on training set:  -2.844117497069476e-08
Accuracy on test set:  -4.09134355552343e-05
```

•Based on your experiments, pick the neural network shape that produces the best model overall, avoiding both overfitting and underfitting. Experiment more comprehensively with the values for the max_iter, learning_rate_init, and alpha parameters until you are convinced you have the best quality model you can train.

In [17]:
```
alpha2=[1, 2,3,4,5,6,7]
```

In [18]:
```
for a in alpha2:
    mlp = MLPRegressor(max_iter=6000,alpha=a,learning_rate_init=0.01, hidden_layer_
    mlp.fit(X_train_scaled, y_train)

    # print out the accuracy on the training and test sets, remembering to
    # use the scaled versions here as well
    train_accuracy = mlp.score(X_train_scaled, y_train)
    test_accuracy = mlp.score(X_test_scaled, y_test)
    print("alpha:", a)
    print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
    print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
    print()
```

```
alpha: 1
Accuracy on training set:  0.8208644570998662
Accuracy on test set:  0.7901836552577707

alpha: 2
Accuracy on training set:  0.819174585698654
Accuracy on test set:  0.7895282653558293

alpha: 3
Accuracy on training set:  0.8045266237932318
Accuracy on test set:  0.7732788678345154

alpha: 4
Accuracy on training set:  0.8061784512661796
Accuracy on test set:  0.7713286821973875

alpha: 5
Accuracy on training set:  0.8200300085210637
Accuracy on test set:  0.7900751136114335

alpha: 6
Accuracy on training set:  0.8247760809543074
Accuracy on test set:  0.7915920365469766

alpha: 7
Accuracy on training set:  0.8005236845683075
Accuracy on test set:  0.7708984327717952
```

In [19]:
```
alpha3=[0.00001, 0.000001,0.000001,0.0000001]
```

In [20]:
```
for a in alpha3:
    mlp = MLPRegressor(max_iter=6000,alpha=a,learning_rate_init=0.01, hidden_layer_
    mlp.fit(X_train_scaled, y_train)

    # print out the accuracy on the training and test sets, remembering to
    # use the scaled versions here as well
    train_accuracy = mlp.score(X_train_scaled, y_train)
    test_accuracy = mlp.score(X_test_scaled, y_test)
    print("alpha:", a)
```

```
    print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
    print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
    print()
```

```
alpha: 1e-05
Accuracy on training set:  0.8132655096258569
Accuracy on test set:  0.7890547446508376

alpha: 1e-06
Accuracy on training set:  0.8225003075177546
Accuracy on test set:  0.792106400985379

alpha: 1e-06
Accuracy on training set:  0.8225003075177546
Accuracy on test set:  0.792106400985379

alpha: 1e-07
Accuracy on training set:  0.8244733677665421
Accuracy on test set:  0.7918937681622842
```

The neural network shape that produces the best model overall avoiding both overfitting and underfitting.

In [21]:
```
mlp = MLPRegressor(max_iter=6000,alpha=1,learning_rate_init=0.01, hidden_layer_size
mlp.fit(X_train_scaled, y_train)

# print out the accuracy on the training and test sets, remembering to
# use the scaled versions here as well
print("Accuracy on training set: ", mlp.score(X_train_scaled, y_train))
print("Accuracy on test set: ", mlp.score(X_test_scaled, y_test))
```

```
Accuracy on training set:  0.8415514292417421
Accuracy on test set:  0.7973070278159309
```