Assignment 7.1 : Classifications and Regression

Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2 - Fundamentals of Data Science
1st Semester	AY 2023-2024

ACTIVITY NO.	**Assignment 7.1 : Classifications and Regression**
Name	William Laurence M. Ramos
Section	CPE32S1
Date Performed:	29/06/2024
Date Submitted:	30/06/2024
Instructor:	Engr. Roman M. Richard

Instructions:

1. Choose any dataset applicable to the classification problem, and also, choose any dataset applicable to the regression problem.
2. Explain your datasets and the problem being addressed.
3. For classification, do the following:
    - Create a base model
    - Evaluate the model with k-fold cross validation
    - Improve the accuracy of your model by applying additional hidden layers
4. For regression, do the following:
    - Create a base model
    - Improve the model by standardizing the dataset
    - Show tuning of layers and neurons (see evaluating small and larger networks)
5. Submit the link to your Google Colab (make sure that it is accessible to me)

Classification Problem

Iris Classification Dataset

In the fields of statistics and machine learning, the Iris flower data set is widely used. Ronald Fisher, a British biologist and statistician, first used it in 1936. 150 examples of iris blossoms from three different species—Setosa, Versicolor, and Virginica—make up the dataset. Four characteristics are measured for every sample:

The Iris dataset attempts to solve the classification problem of placing an iris flower into one of three species (Setosa, Versicolor, or Virginica) based on its four characteristics.

Stated otherwise, the objective is to develop a machine learning model that can identify the species of an iris flower with accuracy based on its measurements, by learning from the properties of the iris flowers in the dataset.

Install the required modules and import libraries

```
pip install keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.13.0-py3-none-any.whl (26 kB)
Collecting keras>=3.2.0 (from scikeras)
  Downloading keras-3.4.1-py3-none-any.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 8.4 MB/s eta 0:00:00
Collecting scikit-learn>=1.4.2 (from scikeras)
  Downloading scikit_learn-1.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 36.2 MB/s eta 0:00:00
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (1.25.2)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (13.7.1)
```

```
Collecting namex (from keras>=3.2.0->scikeras)
  Downloading namex-0.0.8-py3-none-any.whl (5.8 kB)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (3.9.0)
Collecting optree (from keras>=3.2.0->scikeras)
  Downloading optree-0.11.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (311 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.2/311.2 kB 33.7 MB/s eta 0:00:00
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (0.2.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (24.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.11.4
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.4.2
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from optree->keras>=3.2.0->scike
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.(
Installing collected packages: namex, optree, scikit-learn, keras, scikeras
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
  Attempting uninstall: keras
    Found existing installation: keras 2.15.0
    Uninstalling keras-2.15.0:
      Successfully uninstalled keras-2.15.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
tensorflow 2.15.0 requires keras<2.16,>=2.15.0, but you have keras 3.4.1 which is incompatible.
Successfully installed keras-3.4.1 namex-0.0.8 optree-0.11.0 scikeras-0.13.0 scikit-learn-1.5.0
```

```
pip install np_utils
```

```
Collecting np_utils
  Downloading np_utils-0.6.0.tar.gz (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 7.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.0 in /usr/local/lib/python3.10/dist-packages (from np_utils) (1.25.2)
Building wheels for collected packages: np_utils
  Building wheel for np_utils (setup.py) ... done
  Created wheel for np_utils: filename=np_utils-0.6.0-py3-none-any.whl size=56441 sha256=834875aee6b98c89cc03b8d6e5baeef0e157304051
  Stored in directory: /root/.cache/pip/wheels/b6/c7/50/2307607f44366dd021209f660045f8d51cb976514d30be7cc7
Successfully built np_utils
Installing collected packages: np_utils
Successfully installed np_utils-0.6.0
```

```
import pandas
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
```

Start the random number generator.

This step is crucial to guarantee that we can replicate the results precisely in the future. It allows us to reproduce the stochastic training process of a neural network model.

```
seed = 7
np.random.seed(seed)
```

Load the Dataset

```
# load dataset
import pandas

dataframe = pandas.read_csv("iris.csv", header=None, skiprows=1)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
```

```
print(X)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
```

```
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
```

```
print(Y)
```

```
['Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa'
 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa'
 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa'
 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa'
 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa'
 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa' 'Setosa'
 'Setosa' 'Setosa' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor' 'Versicolor'
 'Versicolor' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica']
```

Encoding variables

```
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = to_categorical(encoded_Y)
```

Defining the neural network model

```
def baseline():
    # Create model
    model = Sequential()
    model.add(Dense(8, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
estimator = KerasClassifier(build_fn=baseline, epochs=200, batch_size=5, verbose=1)
```

Evaluate the model with k-fold cross validation

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
```

```
results = cross_val_score(estimator, X, dummy_y, cv=kfold)
print("Baseline Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
Epoch 1/200
/usr/local/lib/python3.10/dist-packages/scikeras/wrappers.py:925: UserWarning: ``build_fn`` will be renamed to ``model`` in a fut
  X, y = self._initialize(X, y)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3408 - loss: 1.3009
Epoch 2/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3836 - loss: 1.2448
Epoch 3/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3168 - loss: 1.1979
Epoch 4/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3022 - loss: 1.1591
Epoch 5/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3726 - loss: 1.0917
Epoch 6/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3829 - loss: 1.0785
Epoch 7/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3815 - loss: 1.0594
Epoch 8/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3457 - loss: 1.0632
Epoch 9/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3595 - loss: 1.0558
Epoch 10/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3169 - loss: 1.0601
Epoch 11/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3488 - loss: 1.0236
Epoch 12/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3651 - loss: 1.0009
Epoch 13/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3580 - loss: 0.9746
Epoch 14/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3548 - loss: 0.9695
Epoch 15/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3641 - loss: 0.9465
Epoch 16/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3137 - loss: 0.9472
Epoch 17/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.2725 - loss: 0.9800
Epoch 18/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3655 - loss: 0.8819
Epoch 19/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4982 - loss: 0.8658
Epoch 20/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5850 - loss: 0.8719
Epoch 21/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6440 - loss: 0.8551
Epoch 22/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7118 - loss: 0.8046
Epoch 23/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6817 - loss: 0.8205
Epoch 24/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6999 - loss: 0.7805
Epoch 25/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7254 - loss: 0.7290
Epoch 26/200
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6784 - loss: 0.7697
Epoch 27/200
```

Improving the accuracy of my model by applying additional hidden layers

```
import pandas
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline

# load dataset
dataframe = pandas.read_csv("iris.csv", header=None, skiprows=1)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]


# Encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)


# Convert integers to dummy variables (i.e., one hot encoded)
dummy_y = to_categorical(encoded_Y)


# Define improved model
def improved_model():
    # Create model
    model = Sequential()
    model.add(Dense(16, input_dim=4, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

In this code, I add a one hidden layer in order to have a total of 2 hidden layers. There are 4 input variables and 3 output variables:

```
# Evaluate improved model
estimator = KerasClassifier(build_fn=improved_model, epochs=250, batch_size=5, verbose=1)
kfold = KFold(n_splits=10, shuffle=True)


improved_results = cross_val_score(estimator, X, dummy_y, cv=kfold)
print("Improved Baseline Accuracy: %.2f%% (%.2f%%)" % (improved_results.mean()*100, improved_results.std()*100))
```

```
/usr/local/lib/python3.10/dist-packages/scikeras/wrappers.py:925: UserWarning: ``build_fn`` will be renamed to ``model`` in a fut
  X, y = self._initialize(X, y)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.2646 - loss: 1.3849
Epoch 2/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.2051 - loss: 1.0388
Epoch 3/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5050 - loss: 0.9388
Epoch 4/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7007 - loss: 0.8537
Epoch 5/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7195 - loss: 0.7681
Epoch 6/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7376 - loss: 0.7181
Epoch 7/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6755 - loss: 0.6746
Epoch 8/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7670 - loss: 0.6289
Epoch 9/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8541 - loss: 0.5607
Epoch 10/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7377 - loss: 0.5159
Epoch 11/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9162 - loss: 0.5064
Epoch 12/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8080 - loss: 0.4954
Epoch 13/250
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9671 - loss: 0.4715
Epoch 14/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9632 - loss: 0.4554
Epoch 15/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8797 - loss: 0.4177
Epoch 16/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9445 - loss: 0.4146
Epoch 17/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9503 - loss: 0.3863
Epoch 18/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9114 - loss: 0.4158
Epoch 19/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9255 - loss: 0.3832
Epoch 20/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9297 - loss: 0.3606
Epoch 21/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9760 - loss: 0.3029
Epoch 22/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9697 - loss: 0.2900
Epoch 23/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9577 - loss: 0.3020
Epoch 24/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9719 - loss: 0.2871
Epoch 25/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9687 - loss: 0.2598
Epoch 26/250
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9656 - loss: 0.2875
Epoch 27/250
```

Analysis

The initial model we developed achieved an accuracy of 96.67% with a standard deviation of 4.47%. This higher standard deviation indicates that the values are more dispersed from each other and the mean, showing greater variability in performance.

On the other hand, our second model has improved to an accuracy of 98.00% and a lower standard deviation of 3.06%. This suggests that the values are more tightly clustered around the mean, indicating more consistent performance.

In conclusion, I enhanced the accuracy of my model by incorporating additional hidden layers. Furthermore, a lower standard deviation typically indicates that data points are more consistent and closer to the mean, whereas a higher standard deviation implies greater variability or spread within the dataset.

Regression Problem

Boston Housing Prices Regression Dataset

The data in this collection was gathered by the United States. Census Bureau regarding housing in the Boston, Massachusetts, region. Since its initial introduction by Harrison and Rubinfeld in 1978, machine learning and regression analysis have made extensive use of it.

There are 506 instances in the collection, each of which represents a distinct Boston suburb. Each instance has fourteen attributes, such as the property tax rate, average number of rooms per residence, crime rate, and median value of owner-occupied dwellings (which is the objective variable).

Using this dataset, our primary goal is to create a regression model that will allow us to test our regression and obtain the mean squared error, or MSE.

Importing Variables

```
import pandas
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
```

Load the Dataset

```
# load dataset

dataframe = pandas.read_csv("boston_housing.csv", header=None, skiprows=1)
dataset = dataframe.values
```

Splitting the data into two variables: X and Y

```
X = dataset[:,0:13]
Y = dataset[:,13]
```

Initialize random number generator.

```
seed = 7
np.random.seed(seed)
```

Defining the baseline model and compiling it

```
# define base model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(13, input_shape=(13,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

Evaluate the baseline model

```
estimator = KerasRegressor(model=baseline_model, epochs=100, batch_size=5, verbose=1)
kfold = KFold(n_splits=10)
results = cross_val_score(estimator, X, Y, cv=kfold, scoring='neg_mean_squared_error')
print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Epoch 1/100
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 506.0464
Epoch 2/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 141.3027
Epoch 3/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 101.6569
Epoch 4/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 89.1484
Epoch 5/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 61.4710
Epoch 6/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 74.7984
Epoch 7/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 71.1770
Epoch 8/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 55.1166
Epoch 9/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 74.4993
Epoch 10/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 56.8839
Epoch 11/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 63.1409
Epoch 12/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 70.3427
Epoch 13/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 62.6673
Epoch 14/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 58.4487
Epoch 15/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 50.6862
Epoch 16/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 46.1957
Epoch 17/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 56.2529
Epoch 18/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 52.2999
Epoch 19/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 47.0302
Epoch 20/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 52.4954
Epoch 21/100
```

```
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 48.1466
Epoch 22/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 41.9326
Epoch 23/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 47.4436
Epoch 24/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 37.9142
Epoch 25/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 48.9142
Epoch 26/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 45.8311
Epoch 27/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 42.9791
Epoch 28/100
```

Improve the model by standardizing the dataset

```python
import pandas
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# load dataset
dataframe = pandas.read_csv("boston_housing.csv", header=None, skiprows=1)
dataset = dataframe.values

# splitting dataset
X = dataset[:,0:13]
Y = dataset[:,13]

# initialize random generator
seed = 7
np.random.seed(seed)

# define standardized model
def standardized_model():
    # create model
    model = Sequential()
    model.add(Dense(13, input_shape=(13,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=standardized_model, epochs=70, batch_size=5, verbose=1)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold, scoring='neg_mean_squared_error')
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Epoch 1/70
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 674.7215
Epoch 2/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 591.1982
Epoch 3/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 542.1035
Epoch 4/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 388.9992
Epoch 5/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 273.3217
Epoch 6/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 198.7250
Epoch 7/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 124.3130
Epoch 8/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 80.7908
Epoch 9/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 59.3224
Epoch 10/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 60.4238
Epoch 11/70
```

```
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 49.9474
Epoch 12/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 39.3056
Epoch 13/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 35.2110
Epoch 14/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 35.9814
Epoch 15/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 28.9472
Epoch 16/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 36.3537
Epoch 17/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 27.3512
Epoch 18/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 25.9128
Epoch 19/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 27.4394
Epoch 20/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 24.8791
Epoch 21/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 27.4887
Epoch 22/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 22.7499
Epoch 23/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.8319
Epoch 24/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 23.0783
Epoch 25/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 27.0628
Epoch 26/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 21.9207
Epoch 27/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 22.2858
Epoch 28/70
```

The MSE from the baseline model was improved by standardising the dataset. After standardisation, the mean MSE dropped to -23.43, suggesting that, on average, the model's predictions are more accurate than the actual data.

Evaluating the model in smaller network

```
import pandas
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# load dataset
dataframe = pandas.read_csv("boston_housing.csv", header=None, skiprows=1)
dataset = dataframe.values

# splitting dataset
X = dataset[:,0:13]
Y = dataset[:,13]

# initialize random generator
seed = 7
np.random.seed(seed)

# define standardized model
def small_model():
    # create model
    model = Sequential()
    model.add(Dense(6, input_shape=(13,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=small_model, epochs=70, batch_size=5, verbose=1)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold, scoring='neg_mean_squared_error')
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Epoch 1/70
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 640.3375
Epoch 2/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 628.6452
Epoch 3/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 604.8555
Epoch 4/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 555.1995
Epoch 5/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 541.6552
Epoch 6/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 498.8346
Epoch 7/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 380.1591
Epoch 8/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 320.0621
Epoch 9/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 253.4726
Epoch 10/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 194.5652
Epoch 11/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 133.6217
Epoch 12/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 98.6225
Epoch 13/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 67.1956
Epoch 14/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 44.9599
Epoch 15/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 52.5641
Epoch 16/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 32.7157
Epoch 17/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 28.2262
Epoch 18/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 27.5706
Epoch 19/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 25.8627
Epoch 20/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 25.0721
Epoch 21/70
```

```
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 32.4400
Epoch 22/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 31.1670
Epoch 23/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 24.2488
Epoch 24/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 24.6163
Epoch 25/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 23.0597
Epoch 26/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 21.7633
Epoch 27/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 20.4860
Epoch 28/70
```

We constructed a smaller neural network with six neurons in its single hidden layer and evaluated the results. In comparison to the normal model, the smaller neural network with 6 neurons in the hidden layer had a higher mean MSE of -31.40. This suggests that improving performance might not necessarily follow from simplifying the model.

Evaluating the model in larger network

```
import pandas
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# load dataset
dataframe = pandas.read_csv("boston_housing.csv", header=None, skiprows=1)
dataset = dataframe.values

# splitting dataset
X = dataset[:,0:13]
Y = dataset[:,13]

# initialize random generator
seed = 7
np.random.seed(seed)

# define standardized model
def large_model():
    # create model
    model = Sequential()
    model.add(Dense(22, input_shape=(13,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=large_model, epochs=70, batch_size=5, verbose=1)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold, scoring='neg_mean_squared_error')
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.4487
Epoch 26/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 18.3762
Epoch 27/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 18.7729
Epoch 28/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 21.8244
Epoch 29/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 27.6380
Epoch 30/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 21.2564
Epoch 31/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 14.1407
Epoch 32/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 18.8132
Epoch 33/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 17.5684
Epoch 34/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 19.2127
Epoch 35/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 18.5483
Epoch 36/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 14.4121
Epoch 37/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 18.1662
Epoch 38/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 13.9494
Epoch 39/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 15.4418
Epoch 40/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 14.5349
Epoch 41/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 16.9687
Epoch 42/70
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 15.9804
Epoch 43/70
```