Problem Set 6

To be submitted on 14th of January 2020

Part A

Diabetes dataset

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

Data Set Characteristics:

Number of Instances

442

Number of Attributes

First 10 columns are numeric predictive values

Target

Column 11 is a quantitative measure of disease progression one year after baseline

Attribute Information

- Age
- Sex
- Body mass index
- Average blood pressure
- S1
- S2
- S3
- S4
- S5
- S6

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times `n_samples` (i.e. the sum of squares of each column totals 1).

Required:

Load the dataset using the Python library 'Sklearn' (it's embedded in the library) and figure out the best fitting to have a good regression model.

Measure the quality of your model by the metrics you studied (MSE, R-square, …).

Part (B)

Iris plants dataset

Data Set Characteristics:

Number of Instances

150 (50 in each of three classes)

Number of Attributes

4 numeric, predictive attributes and the class

Attribute Information

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- **class:**
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

Summary Statistics

sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

Class Distribution

33.3% for each of the 3 classes.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Required:

Load the dataset using the Python library 'Sklearn' (it's embedded in the library) and try different classification algorithms to classify the 3 types of plants (KNN, Logistic Regression, Random Forest, ...).

Measure the quality of your model by the metrics you studied (Confusion matrix, ROC, …).

Optional:

Keras is a famous library for deep learning. It uses Tensorflow as a backend. Install Tensorflow, you can install the CPU version or the GPU version. If you install the GPU version make sure to have a supported Nvidia driver and Cuda toolkit. After installing Tensorflow, it will be easier to install keras. And there are later versions of Tensorflow that come with keras built-in.

The following code helps you to begin in building a neural network

Make sure to read the documentation

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import sklearn

from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras import metrics


iris_data = load_iris() # load the iris dataset


x = iris_data.data
y_ = iris_data.target.reshape(-1, 1) # Convert data to a single column

# One Hot encode the class labels
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform(y_)

train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.30)
```

```
# Build the model

model = Sequential()

model.add(Dense(10, input_shape=(4,), activation='relu', name='fc1'))
model.add(Dense(10, activation='relu', name='fc2'))
model.add(Dense(3, activation='softmax', name='output'))

# Adam optimizer with learning rate of 0.001
optimizer = Adam(lr=0.001)
model.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

print('Neural Network Model Summary: ')
print(model.summary())

model.fit(train_x, train_y, verbose=2, batch_size=5, epochs=200,
validation_split= 0.2)

# Test on unseen data

results = model.evaluate(test_x, test_y)

print('Final test set loss: {:4f}'.format(results[0]))
print('Final test set accuracy: {:4f}'.format(results[1]))

#Confution Matrix and Classification Report
y_pred = model.predict(test_x)
matrix = sklearn.metrics.confusion_matrix(test_y.argmax(axis=1),
y_pred.argmax(axis=1))
print('CM', matrix)
```

Edit the above code to see the effect of the following on the performance of the model:

Adding more layers.

Adding more neurons in each layer.

Using a different optimizers.