Practical Activity 7

Classification Using Neural Networks

This notebook is an exercise for developing a Neural Network (NN) classifier for predicting presence of diabetes in patients. We apply the concepts discussed in Week 7. We walk through NN Classifier in this practical.

Note: this activity is unmarked. It develops your skills for predictive model development using NN.

Task

Our aim is to build a classification model to predict diabetes. We will be using the diabetes dataset which contains 768 observations and 9 variables, as below:

- Pregnancies - Number of times pregnant.
- Glucose - Plasma glucose concentration.
- BloodPressure - Diastolic blood pressure (mm Hg).
- SkinThickness - Skinfold thickness (mm).
- Insulin - Hour serum insulin (mu U/ml).
- BMI – Basal metabolic rate (weight in kg/height in m).
- DiabetesPedigreeFunction - Diabetes pedigree function.
- Age - Age in years.
- Outcome - "1" represents the presence of diabetes while "0" represents the absence of it.

The dataset is available at https://www.kaggle.com/uciml/pima-indians-diabetes-database

Evaluation Metric

We will evaluate the performance of the model using accuracy, which represents the percentage of correctly classified samples.

Step 1 - Loading the required libraries and modules.

In [1]:
```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
```

Step 2 - Reading the data and performing basic data checks.

In [2]:
```
#read in the data as pandas dataframe
df = pd.read_csv('diabetes.csv')

#prints the shape - 768 observations of 9 variables.
print(df.shape)

#summary statistics of the variables.
df.describe()
```

```
(768, 9)
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctior
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.47187
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.33132
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.07800
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.24375
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.37250
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.62625
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.42000

In [3]:
```
df.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

The above summary for the 'Outcome' variable, we observe that the mean value is 0.35, which means that around 35 percent of the observations in the dataset have diabetes.

Step 3 - Creating the training and test datasets.

In [4]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size = 0.3, random_state=42, stratify = df['Ou

X_train = train.drop('Outcome', axis=1)
y_train = train['Outcome']

X_test = test.drop('Outcome', axis = 1)
y_test = test['Outcome']

print(X_train.shape)
print(X_test.shape)
```

```
(537, 8)
(231, 8)
```

Step 4 - Building the neural network model.

In this step, we will build the neural network model using the sklearn's 'Multi-Layer Perceptron Classifier' library https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. We will use three hidden layers with the same number of neurons as the number of features in the dataset.

We will also select 'relu' as the activation function and 'adam' as the solver for weight optimization. To learn more about 'relu' and 'adam', please refer to the Deep Learning with Keras guides here .

In [5]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes = (8, 8, 8),
                    activation = 'relu',
                    solver = 'lbfgs',
                    verbose = 1,
                    random_state=42,
                    max_iter = 10000
                   )

mlp.fit(X_train,y_train)
```

Out[5]:
```
MLPClassifier(hidden_layer_sizes=(8, 8, 8), max_iter=10000, random_state=42,
              solver='lbfgs', verbose=1)
```

Step 5 - Evaluating the neural network model.

In [6]:
```
predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_train, predict_train))
```

```
0.8249534450651769
```

The above output shows the performance of the model on training data. The accuracy is around 0.70.

Next step is to evaluate the performance of the model on the test data that is done with the lines of code below.

In [7]:
```
print(accuracy_score(y_test, predict_test))
```

```
0.7272727272727273
```

Task

Try to find the best set of parameters for the NN model.

In [8]:
```
# set up a list of values for each parameter for cross-validation
parameter_space = {'activation': ['tanh', 'relu'],
                   'solver': ['lbfgs', 'adam'],
                  }
```

In [9]:
```
# import the library
from sklearn.model_selection import GridSearchCV


clf = GridSearchCV(mlp, parameter_space, n_jobs=-1)

# fitting the model for grid search
clf.fit(X_train,y_train)
```

Out[9]:
```
GridSearchCV(estimator=MLPClassifier(hidden_layer_sizes=(8, 8, 8),
                                     max_iter=10000, random_state=42,
                                     solver='lbfgs', verbose=1),
             n_jobs=-1,
             param_grid={'activation': ['tanh', 'relu'],
                         'solver': ['lbfgs', 'adam']})
```

In [10]:
```
# Best paramete set
print('Best parameters found:\n', clf.best_params_)

# All results
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']

for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

```
Best parameters found:
 {'activation': 'relu', 'solver': 'lbfgs'}
0.659 (+/-0.034) for {'activation': 'tanh', 'solver': 'lbfgs'}
0.657 (+/-0.030) for {'activation': 'tanh', 'solver': 'adam'}
0.745 (+/-0.053) for {'activation': 'relu', 'solver': 'lbfgs'}
0.661 (+/-0.044) for {'activation': 'relu', 'solver': 'adam'}
```

In [ ]: