EE4400 - Tutorial 1, Question 3

This question is on the Multi-Layer Perceptron (MLP) and using it to do classification. The aim is to find the best number of hidden nodes in the 3 hidden layers, assuming the same number of hidden nodes in each hidden layer. Cross-validation needs to be done on the training set. The MLP classifier with the best network size is then used for testing.

We shall use the Tensorflow Keras package to implement the MLP classifier. Obtain the data set “from sklearn.datasets import load_iris”. Import the necessary packages.

In [ ]:
```
## load data from scikit
import numpy as np
import pandas as pd
print("pandas version: {}".format(pd.__version__))
import sklearn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import metrics

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
```

```
pandas version: 1.1.5
```

(a) Load the data and split the database into two sets: 80% of samples for training, and 20% of samples for testing.

In [ ]:
```
## load data
iris_dataset = load_iris()
## split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'],
                                                    test_size=0.20,
                                                    random_state=0)
```

Change y_train and y_test to categorical values (required for classification).

In [ ]:
```
y_train = keras.utils.to_categorical(y_train, num_classes = 3)
y_test = keras.utils.to_categorical(y_test, num_classes = 3)
```

(b) Perform a 5-fold Cross-validation using only the training set to determine the best 3-layer MLP classifier with hidden_layer_sizes= (Nhidd,Nhidd,Nhidd) for Nhidd in range(1,11))^ for prediction. In other words, partition the training set into two sets, 4/5 for training and 1/5 for validation; and repeat this process until each of the 1/5 has been validated. ^ The assumption of hidden_layer_sizes= (Nhidd,Nhidd,Nhidd) is to reduce the search space in this exercise. In field applications, the search needs to consider different sizes for each hidden layer.

In [ ]:
```
def MLP_model(Nhidd):
  model = Sequential()
  model.add(Dense(Nhidd, activation='relu', input_shape=(4,)))
  model.add(Dense(Nhidd, activation='relu'))
  model.add(Dense(Nhidd, activation='relu'))
  model.add(Dense(3, activation='softmax'))
  model.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model
```

In [ ]:

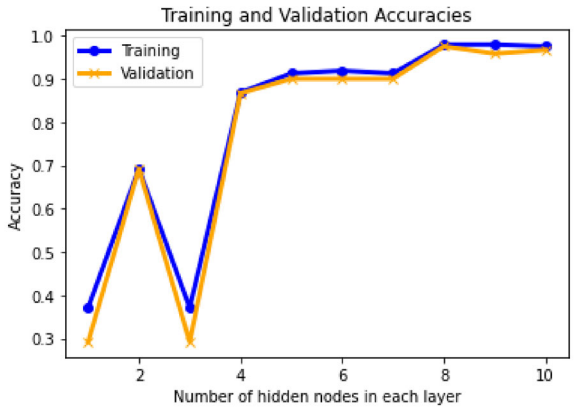
```
acc_train_array = []
acc_valid_array = []
for Nhidd in range(1,11):
    acc_train_array_fold = []
    acc_valid_array_fold = []
    ## Random permutation of data
    Idx = np.random.RandomState(seed=8).permutation(len(y_train))
    ## Tuning: perform 5-fold cross-validation on the training set to determine the best network size
    clf = MLP_model(Nhidd)
    for k in range(0,5):
        N = np.around((k+1)*len(y_train)/5)
        N = N.astype(int)
        Xvalid = X_train[Idx[N-24:N]] # validation features
        Yvalid = y_train[Idx[N-24:N]] # validation targets
        Idxtrn = np.setdiff1d(Idx, Idx[N-24:N])
        Xtrain = X_train[Idxtrn] # training features in tuning loop
        Ytrain = y_train[Idxtrn] # training targets in tuning loop
        ## MLP Classification with same size for each hidden-layer (specified in question)
        clf.fit(Xtrain, Ytrain, epochs = 100, verbose = 0)
        ## trained output
        y_est_p = clf.predict(Xtrain)
        Ytrain_class = np.argmax(Ytrain, axis=1)
        y_est_p_class = np.argmax(y_est_p, axis=1)
        acc_train_array_fold += [metrics.accuracy_score(y_est_p_class,Ytrain_class)]
        ## validation output
        yt_est_p = clf.predict(Xvalid)
        Yvalid_class = np.argmax(Yvalid, axis=1)
        yt_est_p_class = np.argmax(yt_est_p, axis=1)
        acc_valid_array_fold += [metrics.accuracy_score(yt_est_p_class,Yvalid_class)]
    acc_train_array += [np.mean(acc_train_array_fold)]
    acc_valid_array += [np.mean(acc_valid_array_fold)]
    clf.summary()
```

```
Total params: 255
Trainable params: 255
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 10)                50
_________________________________________________________________
dense_37 (Dense)             (None, 10)                110
_________________________________________________________________
dense_38 (Dense)             (None, 10)                110
_________________________________________________________________
dense_39 (Dense)             (None, 3)                 33
=================================================================
Total params: 303
Trainable params: 303
Non-trainable params: 0
_________________________________________________________________
```

(c) Provide a plot of the average 5-fold training and validation accuracies over the different network sizes, i.e. different number of nodes in the hidden layer. Determine the hidden layer size Nhidd that gives the best validation accuracy for the training set.

In [ ]:

```
## plotting
import matplotlib.pyplot as plt
hiddensize = [x for x in range(1,11)]
plt.plot(hiddensize, acc_train_array, color='blue', marker='o', linewidth=3, label='Training')
plt.plot(hiddensize, acc_valid_array, color='orange', marker='x', linewidth=3, label='Validation')
plt.xlabel('Number of hidden nodes in each layer')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracies')
plt.legend()
plt.show()
## find the best hidden layer size that gives the best validation accuracy using only the training se
Nhidden = np.argmax(acc_valid_array,axis=0)+1
print('best hidden layer size =', Nhidden, 'based on 5-fold cross-validation on training set')
```




```
best hidden layer size = 8 based on 5-fold cross-validation on training set
```

In [ ]:

```
print(acc_train_array)
print(acc_valid_array)
```

```
[0.3708333333333333, 0.6916666666666667, 0.3708333333333333, 0.86875, 0.9125, 0.91875, 0.9125, 0.9791666
666666667, 0.9791666666666666, 0.975]
[0.29166666666666663, 0.6916666666666667, 0.29166666666666663, 0.8666666666666668, 0.9, 0.9, 0.9, 0.975,
0.9583333333333333, 0.9666666666666668]
```

(d) Using the best hidden layer size Nhidd in the MLP classifier with hidden_layer_sizes=(Nhidd,Nhidd,Nhidd), evaluate the performance of the MLP by computing the prediction accuracy based on the 20% of samples for testing in part (a).

In [ ]:

```
## perform evaluation
clf = MLP_model(Nhidden)
history=clf.fit(X_train, y_train, epochs = 100, batch_size = 32, verbose = 0)
## trained output
y_test_predict = clf.predict(X_test)
y_test_class = np.argmax(y_test, axis=1)
y_test_predict_class = np.argmax(y_test_predict, axis=1)
test_accuracy = metrics.accuracy_score(y_test_predict_class,y_test_class)
print('test accuracy =', test_accuracy)
```

```
test accuracy = 0.8666666666666667
```

In [ ]:

```
plt.plot(history.history["loss"])
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x7f59fbc89908>]
```