Deniz Aytemiz

ECE5984: HOMEWORK 5

Analysis&Discussion

The best 10 architectures depending on error_rate is shown in Figure 1.

	hidden layers	activation	error_rate	auroc
26	(1, 9)	tanh	0.049460	0.951469
56	(3, 1)	tanh	0.050157	0.950653
47	(2, 7)	tanh	0.050505	0.950446
61	(3, 3)	identity	0.050853	0.949894
69	(3, 6)	relu	0.051202	0.949343
43	(2, 6)	identity	0.051550	0.949250
14	(1, 5)	tanh	0.051550	0.948963
24	(1, 9)	relu	0.051898	0.948412
34	(2, 3)	identity	0.051898	0.948928
65	(3, 4)	tanh	0.051898	0.948699

Figure 1

The worst 10 architectures depending on error_rate is shown in Figure 2.

	hidden layers	activation	error_rate	auroc
29	(2, 1)	tanh	0.540927	0.5
63	(3, 4)	relu	0.540927	0.5
3	(1, 2)	relu	0.540927	0.5
30	(2, 2)	relu	0.540927	0.5
39	(2, 5)	relu	0.459073	0.5
59	(3, 2)	tanh	0.459073	0.5
60	(3, 3)	relu	0.459073	0.5
0	(1, 1)	relu	0.459073	0.5
15	(1, 6)	relu	0.459073	0.5
2	(1, 1)	tanh	0.459073	0.5

Figure 2

The worst results according to AUROC is given in Figure 3.

	hidden layers	activation	error_rate	auroc
0	(1, 1)	relu	0.459073	0.5
39	(2, 5)	relu	0.459073	0.5
15	(1, 6)	relu	0.459073	0.5
30	(2, 2)	relu	0.540927	0.5
59	(3, 2)	tanh	0.459073	0.5
60	(3, 3)	relu	0.459073	0.5
63	(3, 4)	relu	0.540927	0.5
29	(2, 1)	tanh	0.540927	0.5
6	(1, 3)	relu	0.459073	0.5
3	(1, 2)	relu	0.540927	0.5

Figure 3

The best results according to AUROC is given in Figure 4.

	hidden layers	activation	error_rate	auroc
26	(1, 9)	tanh	0.049460	0.951469
56	(3, 1)	tanh	0.050157	0.950653
47	(2, 7)	tanh	0.050505	0.950446
61	(3, 3)	identity	0.050853	0.949894
69	(3, 6)	relu	0.051202	0.949343
43	(2, 6)	identity	0.051550	0.949250
8	(1, 3)	tanh	0.051898	0.949158
54	(3, 1)	relu	0.051898	0.948986
14	(1, 5)	tanh	0.051550	0.948963
34	(2, 3)	identity	0.051898	0.948928

Figure 4

The best performing model considering error rate has 1 layer and 9 nodes with activation mode tanh. Looking at Figure 1 it is hard to make a general assumption about best node and layer numbers because some are more complex and some are simpler but one can suggest tanh seems like the best activation mode.

Considering the worst models depending on error rate which can be seen in Figure 2, one can say relu is the least good performing activation mode in this case. In addition it can be said that relatively simpler models in terms of number of layers and nodes in this case resulted in a worse performance when constarsting to Figure 1's node and layer numbers.

Considering the models we sorted by AUROC metric, relatively simpler models has the best performance when again the best activation mode seems like tanh. Similar to results sorted with error_rate again relu seems like worst performing activation mode. Both metrics agree on the same best architecture and have multiple best architectures in common.

APPENDIX:

```
#import libraries
#import libraries
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_auc_score
import random

#load data
df = pd.read_excel('ccpp.xlsx')
#scaling dataframe
scaler = preprocessing.StandardScaler()
df_scaled = scaler.fit_transform(df.to_numpy())
df_scaled = pd.DataFrame(df_scaled, columns=[
 'ID', 'AT', 'V', 'AP', 'RH', 'TG'])
# column ID and TG don't need to be scaled so I dropped them and added
them from dataframe df
df_scaled = df_scaled.drop('TG', axis= 'columns')
df_scaled = df_scaled.drop('ID', axis= 'columns')
extracted_col1 = df["TG"]
extracted_col2 = df["ID"]
df_scaled = df_scaled.join(extracted_col1)
df_scaled = df_scaled.join(extracted_col2)
# ID is dropped since it is not a feature and TG is the target variable
X = df_scaled.drop(['ID','TG'], axis='columns')
y = df_scaled['TG']
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1)
# Solve the problem using an artificial neural network
regpenalty = 0.001
hl = (5, 5)
clf = MLPClassifier(hidden_layer_sizes=hl, activation='tanh',
solver='adam', \
alpha=regpenalty, early_stopping=True, validation_fraction=0.42)
```

```
clf.fit(X_train,y_train)
annPredY = clf.predict(X_test)
print("\n\rANN: %d mislabeled out of %d points"
% ((y_test != annPredY).sum(), X_test.shape[0]))
print(metrics.confusion_matrix(y_test, annPredY))
trainingLoss = np.asarray(clf.loss_curve_)
validationLoss = np.sqrt(1 - np.asarray(clf.validation_scores_))
factor = trainingLoss[1] / validationLoss[1]
validationLoss = validationLoss*factor
# create figure and axis objects with subplots()
xlabel = "epochs (hl=" + str(hl) + ")"
fig,ax = plt.subplots()
ax.plot(trainingLoss, color="blue")
ax.set_xlabel(xlabel,fontsize=10)
ax.set_ylabel("training loss",color="blue",fontsize=10)
ax2=ax.twinx()
ax2.plot(validationLoss,color="red")
ax2.set_ylabel("validation score",color="red",fontsize=10)
plt.show()

#finding best 10 and worst 10 architectures using AUC as metric

# hidden layers are defined for every possible matching of numbers from 1
to 3
# one_tuple stores 1-tuples as 1,2,3 whereas two_tuples store every
possible combination of numbers in the range
# as (1,2), (3,2) etc. Likewise three_tuple is the 3 numbered version of
it as (2,1,3), (3,1,1) etc
# therefore total stores all these possible hidden layer architectures we
are going to train and
# evaluate performance of

from itertools import product
total =
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8),(1,9),(2,1),(2,2),(2,3),(
2,4),(2,5),(2,6),(2,7),(2,8),(2,9),(3,1),(3,2),(3,3),(3,4),(3,5),(3,6),(3,
7),(3,8),(3,9)]


# all possible activation states
```

```
activations = ('relu', 'identity', 'tanh')

# creating empty dataframe to analyze the results later

results = pd.DataFrame(data=None, columns=['hidden layers',
                                           'activation', 'error_rate',
'auroc'])
# trying all possible combinations of hidden layers and
# activation modes
# adam is selected as solver as default
for i in total:
 for j in activations:
   alphas = 0.001
   clf = MLPClassifier(solver='adam',
                       activation = j,
                       alpha = alphas,
                       hidden_layer_sizes = i,
                       early_stopping = True,
                       validation_fraction=0.42)

   # Train the model with current values
   clf.fit(X_train, y_train)

   # test model on test set
   predictions = clf.predict(X_test)
   actual = y_test

   # find out misclassification error
   # Misclassification Rate = # incorrect predictions / # total
predictions
   error_rate = ((predictions ==
actual).value_counts()[False]/actual.count())


   # Determine the AUROC
   auroc = roc_auc_score(actual, predictions)

   # print architecture properties
   print("Running ", str(i), " hidden layers for the solver: ", str(j))
```

```
    # Store all results for later
    results = results.append({'hidden layers': i,
                              'activation': j,
                              'error_rate': error_rate,
                              'auroc': auroc
                              }, ignore_index=True)

results
table = results.to_excel("output.xlsx")
## best results dependong on error_rate
df1=results.sort_values(by=['error_rate'])
df1.head(10)
## worst results dependong on error_rate
df2 = results.sort_values(by='error_rate', ascending=False)
df2.head(10)
#wost results for auroc
df3 = results.sort_values(by=['auroc'])
df3.head(10)
## best results dependong on auroc
df4 = results.sort_values(by='auroc', ascending=False)
df4.head(10)
```