Homework 4

April 1, 2024

Aaron Vo

Step 1: Read in Titanic.csv and observe a few samples, some features are categorical, and others are numerical. If some features are missing, fill them in using the average of the same feature of other samples. Take a random 80% samples for training and the rest 20% for test.

[1]:
```
# Import the packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
```

[2]:
```
def getValueCounts(df, column):
    print(df[column].value_counts())
```

[3]:
```
#Read the Titanic CSV
df = pd.read_csv("Titanic.csv")
df.head()
```

[3]:
```
   Unnamed: 0 pclass  survived                             name     sex  \
0           1    1st         1     Allen, Miss. Elisabeth Walton  female
1           2    1st         1    Allison, Master. Hudson Trevor    male
2           3    1st         0      Allison, Miss. Helen Loraine  female
3           4    1st         0   Allison, Mr. Hudson Joshua Crei    male
4           5    1st         0   Allison, Mrs. Hudson J C (Bessi  female

       age  sibsp  parch  ticket        fare    cabin    embarked boat  \
0  29.0000      0      0   24160  211.337494       B5  Southampton    2
1   0.9167      1      2  113781  151.550003  C22 C26  Southampton   11
2   2.0000      1      2  113781  151.550003  C22 C26  Southampton  NaN
3  30.0000      1      2  113781  151.550003  C22 C26  Southampton  NaN
4  25.0000      1      2  113781  151.550003  C22 C26  Southampton  NaN

    body                         home.dest
0    NaN                      St Louis, MO
1    NaN  Montreal, PQ / Chesterville, ON
2    NaN  Montreal, PQ / Chesterville, ON
```

```
3  135.0  Montreal, PQ / Chesterville, ON
4    NaN  Montreal, PQ / Chesterville, ON
```

[4]:
```
df['sex'] = df['sex'].replace({'female': 0, 'male': 1})
getValueCounts(df, 'sex')
```

```
1    843
0    466
Name: sex, dtype: int64
```

[5]:
```
df['pclass'] = df['pclass'].replace({'1st':1, '2nd':2, '3rd': 3})
getValueCounts(df, 'pclass')
```

```
3    709
1    323
2    277
Name: pclass, dtype: int64
```

[6]:
```
def getNaNCount(df):
    nan_count = df.isna().sum()
    print(nan_count)
```

[7]:
```
age_mean = df['age'].mean()
df['age'].fillna(age_mean, inplace=True)
getNaNCount(df)
```

```
Unnamed: 0       0
pclass           0
survived         0
name             0
sex              0
age              0
sibsp            0
parch            0
ticket           0
fare             1
cabin         1014
embarked         2
boat           823
body          1188
home.dest      564
dtype: int64
```

[8]:
```
data = df[['pclass', 'sex', 'age', 'sibsp', 'survived']]
data.head()
```

[8]:

	pclass	sex	age	sibsp	survived
0	1	0	29.0000	0	1
1	1	1	0.9167	1	1

```
2        1    0   2.0000      1          0
3        1    1  30.0000      1          0
4        1    0  25.0000      1          0
```

[9]:
```
cleaned_data=data.dropna(axis = 0 )
cleaned_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pclass    1309 non-null   int64
 1   sex       1309 non-null   int64
 2   age       1309 non-null   float64
 3   sibsp     1309 non-null   int64
 4   survived  1309 non-null   int64
dtypes: float64(1), int64(4)
memory usage: 51.3 KB
```

[10]:
```
X_train, X_test, y_train, y_test = train_test_split(cleaned_data[['pclass',␣
↪'sex', 'age', 'sibsp']],
                                                    cleaned_data['survived'],
                                                    test_size = 0.2,
                                                    random_state = 5)
```

Step 2: Fit a neural network using independent variables 'pclass + sex + age + sibsp' and dependent variable 'survived'. Fill in n/a attributes with the average of the same attributes from other training examples. Use 2 hidden layers and set the activation functions for both the hidden and output layer to be the sigmoid function. Set "solver" parameter as either SGD (stochastic gradient descend) or Adam (similar to SGD but optimized performance with mini batches). You can adjust parameter "alpha" for regularization (to control overfitting) and other parameters such as "learning rate" and "momentum" as needed.

[11]:
```
clf_adam_1 = MLPClassifier(hidden_layer_sizes=(4, 4), activation='logistic',␣
↪alpha=1e-4, learning_rate='constant', solver='adam',␣
↪max_iter=5000,momentum=0.9, random_state=1)
```

[12]:
```
clf_adam_2 = MLPClassifier(hidden_layer_sizes=(8, 4), activation='logistic',␣
↪alpha=1e-4,learning_rate='constant',solver='adam',max_iter=5000,momentum=0.
↪9,random_state=1)
```

Step 3: Check the performance of the model with out-of- sample accuracy, defined as

- out-of-sample percent survivors correctly predicted (on test set)

- out-of-sample percent fatalities correctly predicted (on test set)

Please try two different network structures (i.e., number of neurons at each hidden layer) and show their respective accuracy.

[13]:
```
def printConfusionMatrix(y_test,y_pred):
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(conf_matrix)

    percent_survivors_correct = conf_matrix[0, 0] / (conf_matrix[0, 0] +␣
 ↪conf_matrix[0, 1])
    percent_fatalities_correct = conf_matrix[1, 1] / (conf_matrix[1, 0] +␣
 ↪conf_matrix[1, 1])

    print(f"Percent Survivors Correctly Predicted: {percent_survivors_correct:.
 ↪2%}")
    print(f"Percent Fatalities Correctly Predicted: {percent_fatalities_correct:
 ↪.2%}")
```

[14]:
```
clf_adam_1.fit(X_train, y_train)
```

[14]:
```
MLPClassifier(activation='logistic', hidden_layer_sizes=(4, 4), max_iter=5000,
              random_state=1)
```

[15]:
```
printConfusionMatrix(y_test, clf_adam_1.predict(X_test))
```

```
[[141  21]
 [ 35  65]]
Percent Survivors Correctly Predicted: 87.04%
Percent Fatalities Correctly Predicted: 65.00%
```

[16]:
```
print(classification_report(y_test, clf_adam_1.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       162
           1       0.76      0.65      0.70       100

    accuracy                           0.79       262
   macro avg       0.78      0.76      0.77       262
weighted avg       0.78      0.79      0.78       262
```

[17]:
```
clf_adam_2.fit(X_train, y_train)
```

[17]:
```
MLPClassifier(activation='logistic', hidden_layer_sizes=(8, 4), max_iter=5000,
              random_state=1)
```

[18]:
```
printConfusionMatrix(y_test, clf_adam_2.predict(X_test))
```

```
[[140  22]
 [ 38  62]]
Percent Survivors Correctly Predicted: 86.42%
Percent Fatalities Correctly Predicted: 62.00%
```

[19]:
```
print(classification_report(y_test, clf_adam_2.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       162
           1       0.74      0.62      0.67       100

    accuracy                           0.77       262
   macro avg       0.76      0.74      0.75       262
weighted avg       0.77      0.77      0.77       262
```

Step 4: Compare the out-of-sample accuracy (as defined in step 3) with the random forest obtained in homework #3. (You can either use a table or plot the results of the two algorithms in one figure). Explain any difference in accuracy.

In terms of correctly predicting survivors, both neural network architectures outperform the Random Forest model. The Neural Network with hidden layer sizes (4,4) achieves the highest accuracy at 87.04%, closely followed by the Neural Network with hidden layer sizes (8,4) at 86.42%.

For correctly predicting fatalities, the Random Forest model outperforms both neural network architectures. Among the neural network models, the one with hidden layer sizes (4,4) performs slightly better than the one with hidden layer sizes (8,4), with accuracies of 65.00% and 62.00% respectively.

	Random Forest	Neural Network (4,4)	Neural Network (8,4)
Survivors Correctly Predicted	79.63%	87.04%	86.42%
Fatalities Correctly Predicted	68.00%	65.00%	62.00%