6. Deep Learning – Evaluate Model Performance

Contents

1. Model evaluation 2

2. Data splitting - Automatic Verification Dataset 3

3. Data splitting - Use a Manual Verification Dataset 7

4. Manual k-Fold Cross-Validation 11

6. Deep Learning – Evaluate Model Performance

1. Model evaluation

- ✓ We can evaluate the model by using below ways,
    - o Data Splitting
        - ▪ Use an automatic verification dataset
        - ▪ Use a manual verification dataset
    - o Manual k-Fold Cross-Validation

2. Data splitting - Automatic Verification Dataset

- ✓ Once model compiling is done then we need to train the model.
- ✓ validation_split keyword argument helps in automation verification dataset.
- ✓ We can train the model by using fit() method
  - For fit(validation_split = 0.33) method we can provide validation_split = 0.33 as keyword argument.
- ✓ We can give 0.2 or 0.33 for 20% or 33% of your training data held back for validation.

Program	Model evaluation training
Name	demo6.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()


print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]



print("Step 4: Splitting the dataset: Optional")
```

```
print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

model.compile(
   loss = 'binary_crossentropy',
   optimizer = 'adam',
   metrics = ['accuracy']
)

print("Step 7: Model training")

model.fit(
   X,
   y,
   validation_split = 0.33,
   epochs = 150,
   batch_size = 10,
)
```

Output

```
Epoch 147/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7713 - loss: 0.4595 - val_accuracy:
 0.7441 - val_loss: 0.5430
Epoch 148/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7874 - loss: 0.4698 - val_accuracy:
 0.7283 - val_loss: 0.6000
Epoch 149/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7521 - loss: 0.4930 - val_accuracy:
 0.7520 - val_loss: 0.5419
Epoch 150/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7752 - loss: 0.4899 - val_accuracy:
 0.7008 - val_loss: 0.5801
```

Note

- ✓ Running the example, you can see that the verbose output on each epoch shows the loss and accuracy on both the training dataset and the validation dataset.

3. Data splitting - Use a Manual Verification Dataset

- ✓ train_test_split() function helps in manual verification of the dataset.
- ✓ Let me reminder we already worked with train_test_split() function in machine learning.

Program	Model evaluation training
Name	demo6.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split


import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]
```

```
print("Step 4: Splitting the dataset: Optional")

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size = 0.33,
    random_state = 7
)


print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)
```

```
print("Step 7: Model training")

model.fit(
    X_train,
    y_train,
    validation_data = (X_test, y_test),
    epochs = 150,
    batch_size = 10
)
```

Output

```
Epoch 148/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7712 - loss: 0.4804 - val_accuracy:
 0.7283 - val_loss: 0.5412
Epoch 149/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7858 - loss: 0.4621 - val_accuracy:
 0.7638 - val_loss: 0.5658
Epoch 150/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7714 - loss: 0.4710 - val_accuracy:
 0.7362 - val_loss: 0.5260
```

4. Manual k-Fold Cross-Validation

- ✓ By using k fold cross validation we can evaluate the model.
- ✓ Let me remind, we already worked with k fold cross validation in machine learning.
- ✓ It provides very good performance on the model.
- ✓ It splits the dataset into k subsets and used to apply training and validate on subsets.
- ✓ Finally it used to get average score from all models.

Program	K fold cross validation
Name	demo3.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import StratifiedKFold
import numpy as np

import warnings
warnings.filterwarnings("ignore")


# load the dataset
dataset = np.loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state
= 7)

cvscores = []

for train, test in kfold.split(X, y):
        # create model
        model = Sequential()

        model.add(Dense(12, input_shape=(8,), activation='relu'))
        model.add(Dense(8, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

        # Compile model
        model.compile(loss='binary_crossentropy',
```

```
            optimizer='adam', metrics = ['accuracy'])

            # Fit the model
            model.fit(X[train], y[train], epochs=150, batch_size=10, verbose=0)

            # evaluate the model
            scores = model.evaluate(X[test], y[test])

            print(("Accuracy:", scores[1]*100))

            cvscores.append(scores[1] * 100)

print("Mean Accuracy", np.mean(cvscores))
```

Output

```
('Accuracy:', 59.740257263183594)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 0s/step - accuracy: 0.7113 - loss: 0.5755
('Accuracy:', 68.83116960525513)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 0s/step - accuracy: 0.7152 - loss: 0.5587
('Accuracy:', 72.72727489471436)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 8ms/step - accuracy: 0.7212 - loss: 0.6130
('Accuracy:', 72.36841917037964)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 0s/step - accuracy: 0.7644 - loss: 0.4472
('Accuracy:', 76.31579041481018)
Mean Accuracy 71.36192798614502
```