Kunnithi Treecharoensomboon ID 63070503461

CPE383 Machine Learning: Quiz 7

1. 15 points. 1 hour. Show that the OR, AND will work for a single layer perceptron, but the XOR will not find a solution.

   The AND and OR logical operations can be modeled using a single-layer perceptron with linearly separable input data. For example, in the case of the AND operation, if we represent the two input variables as x1 and x2, we can set the weights w1 and w2 of the perceptron such that w1 + w2 > 1.0, and set the bias term b to be negative. This will cause the perceptron to output a positive value only when both inputs are positive, which is the desired behavior for the AND operation. Similarly, we can set the weights and bias term for the OR operation to produce the desired output.

   However, the XOR logical operation is not linearly separable, and cannot be modeled by a single-layer perceptron. This is because no matter how we set the weights and biases, we cannot draw a straight line that can perfectly separate the two classes of input data (i.e., those that produce a 0 output and those that produce a 1 output). Therefore, we need to use a multi-layer perceptron or some other non-linear classifier to model the XOR operation.

   OR

In1	In2	out
0	0	0
0	1	1
1	0	1
1	1	1

   AND

In1	In2	out
0	0	0
0	1	0
1	0	0
1	1	1

   XOR

   the XOR logical function is not linearly separable. The inputs are binary values, and the output is 1 only if one of the inputs is 1, but not both. the OR and AND logical functions will work for a single layer perceptron because they are linearly separable, but the XOR function will not find a solution because it is not linearly separable.

In1	In2	out
0	0	0
0	1	1
1	0	1
1	1	0

2. 15 points. 1 hour. Use multilayer perceptron on sklearn's diabetes data meant for regression using 25% test data, find the R-square.

```
[1] #copied from https://towardsdatascience.com/deep-neural-multilayer-perceptron-mlp-with-scikit-learn-2698e77155e
    from sklearn.neural_network import MLPRegressor
    from sklearn.model_selection import train_test_split
    from sklearn.datasets import load_diabetes
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import r2_score
```

```
[2] # Load diabetes data
    diabetes_data = load_diabetes()
    X = diabetes_data.data
    y = diabetes_data.target

    # Split test and train data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

```
[3] # Scale the data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

```
[4] # Train the model
    reg = MLPRegressor(hidden_layer_sizes=(64, 64, 64), activation="relu", random_state=1, max_iter=2000).fit(X_train_scaled, y_train)
```

```
    # Make predictions on the test set
    y_pred = reg.predict(X_test_scaled)

    # Calculate R-square
    r_square = r2_score(y_test, y_pred)
    print("R-square: ", r_square)
```

```
R-square:  -0.28060369491997483
```

3. 20 points. 2 hours. Follow the instructions in the following site for MLPClassifier for MNIST data. https://towardsdatascience.com/classifying-handwritten-digits-using-a-multilayer-perceptron-class ifier-mlp-bc8453655880 . Use 50,000 data samples for training and 20,000 for testing. Note you should scale your data to be 0-1 by dividing it by 255. Report your accuracy and print out the confusion matrix.

```
# Load data
X, y = fetch_openml("mnist_784", version=1, return_X_y=True)

# Normalize intensity of images to make it in the range [0,1] since 255 is the max (white).
X = X / 255.0

# Split the data into train/test sets
X_train, X_test = X[:50000], X[20000:]
y_train, y_test = y[:50000], y[20000:]

classifier = MLPClassifier(
    hidden_layer_sizes=(50,20,10),
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)
```

```
classifier = MLPClassifier(
    hidden_layer_sizes=(50,20,10),
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)

# fit the model on the training data
classifier.fit(X_train, y_train)
```

```
Iteration 52, loss = 0.02205017
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

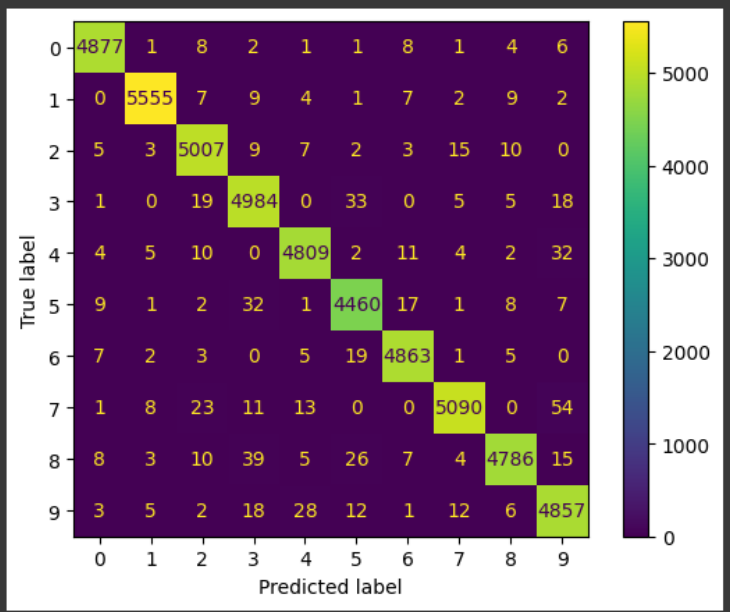
MLPClassifier

```
MLPClassifier(hidden_layer_sizes=(50, 20, 10), learning_rate_init=0.1,
              max_iter=100, random_state=1, solver='sgd', verbose=10)
```

[31]
```
# make predictions
print("Training set score: %f" % classifier.score(X_train, y_train))
print("Test set score: %f" % classifier.score(X_test, y_test))
```

```
Training set score: 0.996200
Test set score: 0.985760
```

[32]
```
# print confusion matrix
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[str(i) for i in classifier.classes_])
disp.plot()
plt.show()
```



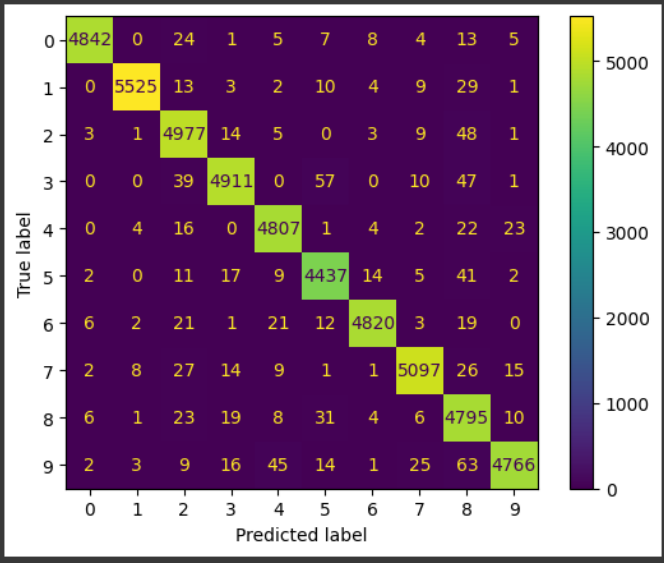
4. 10 points. 0.5 hours. Repeat problem #3, but also normalize the input data by dividing by variance on the scaled 0-1 data by using sklearn's StandardScaler. Report your accuracy and see if it improves from problem #3.

```
# Load data
X, y = fetch_openml("mnist_784", version=1, return_X_y=True)

# Normalize input data using StandardScaler
scaler = StandardScaler()
X = X / 255
X = scaler.fit_transform(X)

# Split the data into train/test sets
X_train, X_test = X[:50000], X[20000:]
y_train, y_test = y[:50000], y[20000:]

classifier = MLPClassifier(
    hidden_layer_sizes=(50,20,10),
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)
```

```
[39] # make predictions
print("Training set score: %f" % classifier.score(X_train, y_train))
print("Test set score: %f" % classifier.score(X_test, y_test))
```

```
Training set score: 0.989700
Test set score: 0.979540
```

```
# print confusion matrix
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[str(i) for i in classifier.classes_])
disp.plot()
plt.show()
```



ANS The accuracy drops a little bit