

i) Running the example code results in the following confusion matrix

The screenshot displays a Jupyter Notebook environment with the following components:

- Code Editor:** Contains Python code for training a Perceptron model. The code includes data splitting, model training, prediction, and visualization of the confusion matrix and accuracy.
- Variable Explorer:** Shows the variables 'accuracy' (float64, 1) and 'fx_test' (Array of int32, (188,)).
- Console:** Displays the output of the code, including the confusion matrix and the weighted average accuracy.

The confusion matrix is as follows:

	0	1	2	3	4	5	6	7	8	9
0	20	0	0	0	0	0	0	0	0	0
1	0	18	0	0	1	0	0	0	0	0
2	0	0	21	0	0	0	0	0	0	0
3	0	0	0	21	0	0	0	0	1	0
4	0	1	0	0	17	0	0	0	0	0
5	0	1	0	0	0	15	0	0	0	1
6	0	0	0	0	0	0	15	0	0	0
7	0	0	0	1	0	0	0	19	1	0
8	0	0	0	0	1	0	0	0	14	0
9	0	1	0	0	0	1	0	0	1	8

The weighted average accuracy is 0.94.

The second line is for '1' class

Precision = TP / (TP+FP)

TP (True Positives) = 18

FP (False Positives) = 3

Precision= 18/(18+3) = 0.857

ii)

from sklearn.model_selection import train_test_split

from sklearn.linear_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

import matplotlib.pyplot as plt

from sklearn import datasets

import numpy as np

import cv2

load the MNIST digits dataset

mnist = datasets.load_digits()

X = mnist.data

```

y = mnist.target

# Standard normalization
scaler = StandardScaler()
X = scaler.fit_transform(X)

# split dataset into training and test set
(X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.10, random_state=1)

# show the sizes of each data split
print("Number of train samples: {}".format(len(X_train)))
print("Number of test samples: {}".format(len(X_test)))

# Parameter tuning
best_accuracy = 0
best_max_iter = 0
best_tol = 0

for max_iter in [100, 500, 1000, 1500]:
    for tol in [0.0001, 0.001, 0.01, 0.1]:
        model = Perceptron(max_iter=max_iter, tol=tol)
        model.fit(X_train, y_train)

        fx_test = model.predict(X_test)
        accuracy = accuracy_score(y_test, fx_test)

        print(f"Max_iter: {max_iter}, Tol: {tol}, Accuracy: {accuracy}")

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_max_iter = max_iter

```

```

best_tol = tol

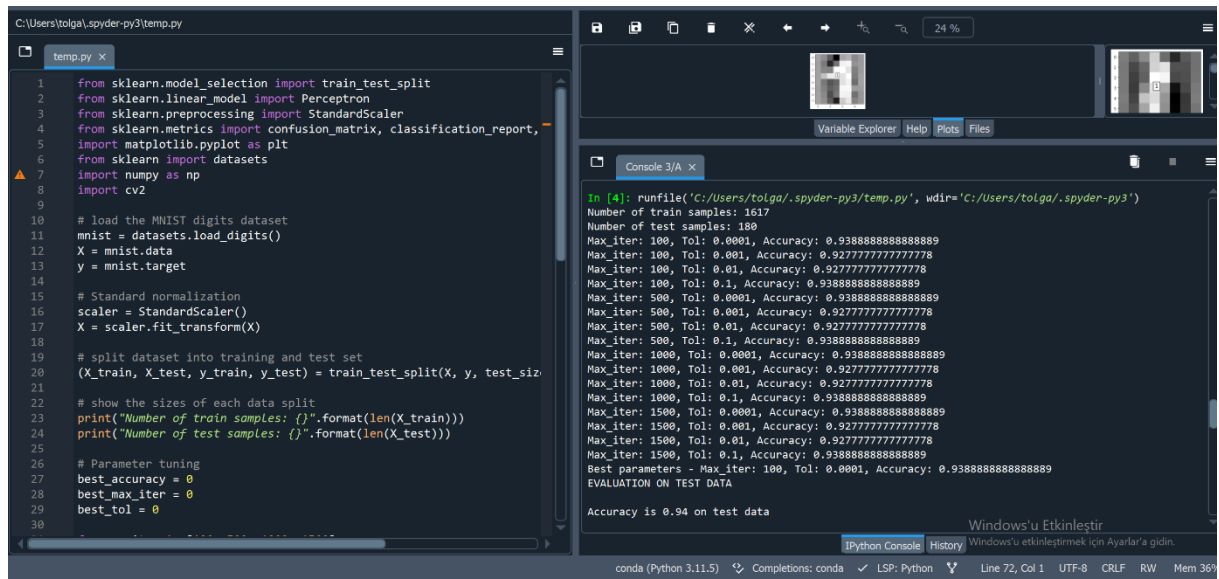
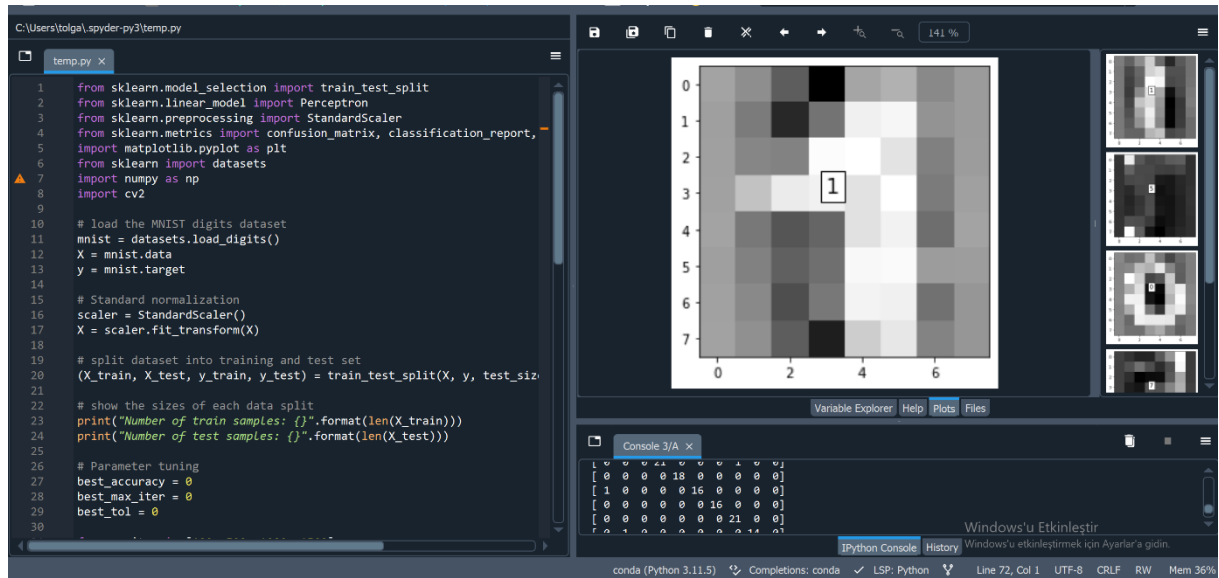
# Report the best parameters
print(f"Best parameters - Max_iter: {best_max_iter}, Tol: {best_tol}, Accuracy: {best_accuracy}")

# train the model with the best parameters
model = Perceptron(max_iter=best_max_iter, tol=best_tol)
model.fit(X_train, y_train)

# test and analyze the model
fx_test = model.predict(X_test)
print("EVALUATION ON TEST DATA\n")
accuracy = accuracy_score(y_test, fx_test)
print("Accuracy is %.2f on test data\n" % accuracy)
print(classification_report(y_test, fx_test))
print("Confusion matrix")
print(confusion_matrix(y_test, fx_test))

# visualize the first 5 digits and annotate them
for i in range(5):
    # get the data and classify it
    label = model.predict(X_test[i:i+1])
    image = X_test[i].reshape((8, 8))
    plt.imshow(image, cmap='gray')
    plt.annotate(label[0], (3, 3), bbox={'facecolor': 'white'}, fontsize=16)
    print("I think the digit is : {}".format(label[0]))
    plt.show()
    cv2.waitKey(0)

```



```

1 from sklearn.model_selection import train_test_split
2 from sklearn.linear_model import Perceptron
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.metrics import confusion_matrix, classification_report,
5 import matplotlib.pyplot as plt
6 from sklearn import datasets
7 import numpy as np
8 import cv2
9
10 # load the MNIST digits dataset
11 mnist = datasets.load_digits()
12 X = mnist.data
13 y = mnist.target
14
15 # Standard normalization
16 scaler = StandardScaler()
17 X = scaler.fit_transform(X)
18
19 # split dataset into training and test set
20 (X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.2)
21
22 # show the sizes of each data split
23 print("Number of train samples: {}".format(len(X_train)))
24 print("Number of test samples: {}".format(len(X_test)))
25
26 # Parameter tuning
27 best_accuracy = 0
28 best_max_iter = 0
29 best_tol = 0
30

```

Best parameters - Max_iter: 100, Tol: 0.0001, Accuracy: 0.9388888888888889

EVALUATION ON TEST DATA

Accuracy is 0.94 on test data

	precision	recall	f1-score	support
0	0.95	1.00	0.98	20
1	0.89	0.84	0.86	19
2	1.00	0.95	0.98	21
3	0.95	0.95	0.95	22
4	0.95	1.00	0.97	18
5	1.00	0.94	0.97	17
6	1.00	1.00	1.00	16
7	0.91	1.00	0.95	21
8	0.78	0.93	0.85	15
9	1.00	0.64	0.78	11
accuracy			0.94	180
macro avg	0.94	0.93	0.93	180
weighted avg	0.94	0.94	0.94	180

Confusion matrix

```

[[20  0  0  0  0  0  0  0  0]
 [ 0 19  0  0  0  0  0  0  0]
 [ 0  0 21  0  0  0  0  0  0]
 [ 0  0  0 22  0  0  0  0  0]
 [ 0  0  0  0 18  0  0  0  0]
 [ 0  0  0  0  0 17  0  0  0]
 [ 0  0  0  0  0  0 16  0  0]
 [ 0  0  0  0  0  0  0 15  0]
 [ 0  0  0  0  0  0  0  0 11]

```

"This code includes standard normalization and performs a grid search on the max_iter and tol parameters for the Perceptron model. The goal is to determine the combination of these parameters that provides the best performance. The best selected parameters are then used to train the final model and the effectiveness of the model is evaluated on test data "The effect of normalization may vary depending on the characteristics of the algorithm and the dataset. Therefore, it is always important to conduct experiments to observe and analyze the results."

iii) Modified Code Fragment

model = svm.SVC(kernel = "linear")

```

1 from sklearn.model_selection import train_test_split
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.metrics import confusion_matrix, classification_report,
5 import matplotlib.pyplot as plt
6 from sklearn import datasets
7 import numpy as np
8 import cv2
9
10 # load the MNIST digits dataset
11 mnist = datasets.load_digits()
12 X = mnist.data
13 y = mnist.target
14
15 # Standard normalization
16 scaler = StandardScaler()
17 X = scaler.fit_transform(X)
18
19 # split dataset into training and test set
20 (X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.2)
21
22 # show the sizes of each data split
23 print("Number of train samples: {}".format(len(X_train)))
24 print("Number of test samples: {}".format(len(X_test)))
25
26 # Parameter tuning
27 best_accuracy = 0
28 best_C = 0
29 best_decision_function_shape = ''
30

```

In [5]: runfile('C:/Users/tolga/.spyder-py3/temp.py', wdir='C:/Users/tolga/.spyder-py3')

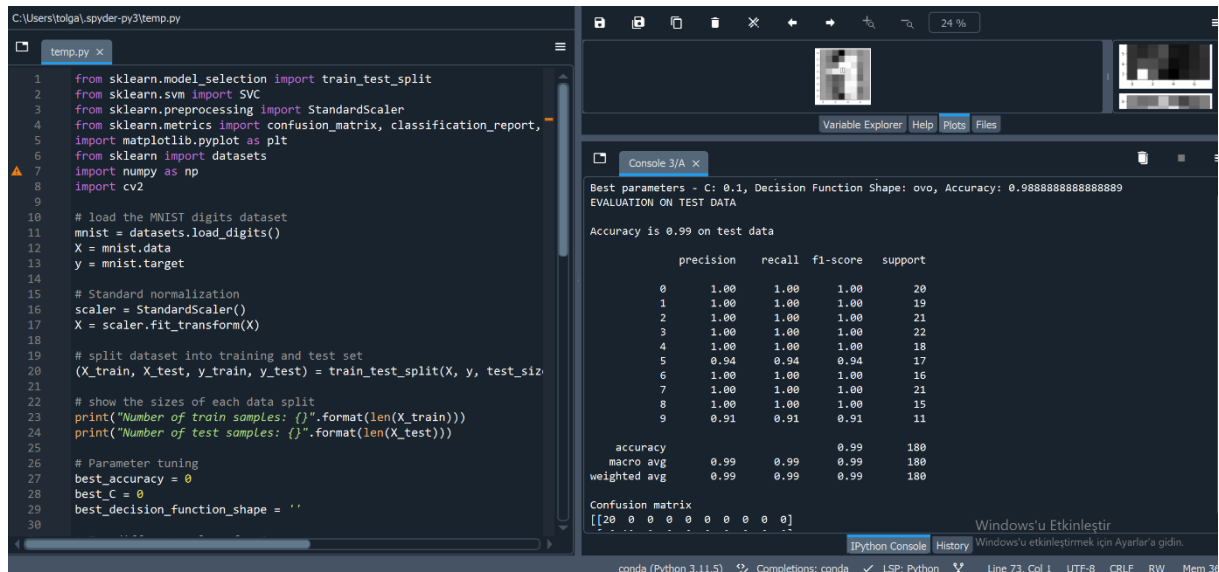
Number of train samples: 167
Number of test samples: 180

C: 0.1, Decision Function Shape: ovo, Accuracy: 0.9888888888888889
C: 0.1, Decision Function Shape: ovr, Accuracy: 0.9888888888888889
C: 1, Decision Function Shape: ovo, Accuracy: 0.9888888888888889
C: 1, Decision Function Shape: ovr, Accuracy: 0.9888888888888889
C: 10, Decision Function Shape: ovo, Accuracy: 0.9888888888888889
C: 10, Decision Function Shape: ovr, Accuracy: 0.9888888888888889
C: 100, Decision Function Shape: ovo, Accuracy: 0.9888888888888889
C: 100, Decision Function Shape: ovr, Accuracy: 0.9888888888888889
Best parameters - C: 0.1, Decision Function Shape: ovo, Accuracy: 0.9888888888888889

EVALUATION ON TEST DATA

Accuracy is 0.99 on test data

	precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	1.00	1.00	1.00	19
2	1.00	1.00	1.00	21
3	1.00	1.00	1.00	22
4	1.00	1.00	1.00	18



This code loops over different C and decision_function_shape values and reports the best parameters based on accuracy. The best parameters are then used to train the final SVC model and its performance is evaluated on test data. Adjustments to C and decision function shape values can have varying effects on the performance of the model, so it is important to observe and analyze the results.

iv)

model = LinearSVC (C=1.0, dual=False) # yes to lagrange dual solution

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.svm import LinearSVC
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.metrics import confusion_matrix, classification_report,
5 import matplotlib.pyplot as plt
6 from sklearn import datasets
7 import numpy as np
8 import cv2
9
10 # load the MNIST digits dataset
11 mnist = datasets.load_digits()
12 X = mnist.data
13 y = mnist.target
14
15 # Standard normalization
16 scaler = StandardScaler()
17 X = scaler.fit_transform(X)
18
19 # split dataset into training and test set
20 (X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.2)
21
22 # show the sizes of each data split
23 print("Number of train samples: {}".format(len(X_train)))
24 print("Number of test samples: {}".format(len(X_test)))
25
26 # Parameter tuning
27 best_accuracy = 0
28 best_dual = None
29
30 # Try different values for dual
```

Console 3/A x

```
warnings.warn(
EVALUATION ON TEST DATA
Accuracy is 0.95 on test data
precision recall f1-score support
0 1.00 1.00 1.00 20
1 0.94 0.89 0.92 19
2 0.95 1.00 0.98 21
3 0.96 1.00 0.98 22
4 0.95 1.00 0.97 18
5 0.88 0.88 0.88 17
6 1.00 1.00 1.00 16
7 0.95 0.95 0.95 21
8 0.93 0.87 0.90 15
9 0.90 0.82 0.86 11
accuracy 0.95 180
macro avg 0.95 0.94 0.94 180
weighted avg 0.95 0.95 0.95 180
Confusion matrix
[[20 0 0 0 0 0 0 0 0]
[ 0 19 0 0 0 0 0 0 0]
[ 0 0 21 0 0 0 0 0 0]
[ 0 0 0 22 0 0 0 0 0]
[ 0 0 0 0 18 0 0 0 0]
[ 0 0 0 0 0 17 0 0 0]
[ 0 0 0 0 0 0 16 0 0]
[ 0 0 0 0 0 0 0 21 0]
[ 0 0 0 0 0 0 0 0 15]
[ 0 0 0 0 0 0 0 0 11]]
```

Windows'u Etkinleştir
Windows'u etkinleştirmek için Ayarlar'a gidin.

conda (Python 3.11.5) Completions: conda LSP: Python Line 70, Col 1 UTF-8 CRLF RW Mem 36%

The LinearSVC class in the scikit-learn library has a parameter called dual, and setting this parameter to False uses the primal optimization problem, while setting it to True uses the dual optimization problem. Generally, the choice between dual=False and dual=True depends on the relationship between the number of samples (n_samples) and the number of features (n_features). If n_samples is much larger than n_features, it will usually be more efficient to set dual=False.

This code tests LinearSVC with different dual values and reports the best parameter based on accuracy. In practice, the performance comparison between dual=False and dual=True depends on the characteristics and characteristics of a particular dataset. Generally, for datasets with a large number of samples and a small number of features, setting dual=False can lead to better performance.

v)

Code using Logistic Regression using different solvers ('newton-cg', 'lbfgs', 'liblinear'). The best dissolution values will be determined accordingly.

This code tests Logistic Regression with different solvers and reports accuracy, basic metric and good solver. Since Logistic Regression's default solver is 'lbfgs', the default values will be determined based on the best solver.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import matplotlib.pyplot as plt
from sklearn import datasets
import numpy as np
import cv2

# load the MNIST digits dataset
mnist = datasets.load_digits()
X = mnist.data
y = mnist.target

# Standard normalization
scaler = StandardScaler()
X = scaler.fit_transform(X)

# split dataset into training and test set
(X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.10, random_state=1)

# show the sizes of each data split
print("Number of train samples: {}".format(len(X_train)))
print("Number of test samples: {}".format(len(X_test)))

# Parameter tuning
```



```

best_accuracy = 0
best_solver = None

# Try different solvers
for solver in ['newton-cg', 'lbfgs', 'liblinear']:
    model = LogisticRegression(solver=solver)
    model.fit(X_train, y_train)

    fx_test = model.predict(X_test)
    accuracy = accuracy_score(y_test, fx_test)

    print(f"Solver: {solver}, Accuracy: {accuracy}")

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_solver = solver

# Report the best parameter
print(f"Best solver - Solver: {best_solver}, Accuracy: {best_accuracy}")

# train the model with the best parameter
model = LogisticRegression(solver=best_solver)
model.fit(X_train, y_train)

# test and analyze the model
fx_test = model.predict(X_test)
print("EVALUATION ON TEST DATA\n")
accuracy = accuracy_score(y_test, fx_test)
print("Accuracy is %.2f on test data\n" % accuracy)
print(classification_report(y_test, fx_test))
print("Confusion matrix")

```

```
print(confusion_matrix(y_test, fx_test))
```

```
# visualize the first 5 digits and annotate them
```

```
for i in range(5):
```

```
    # get the data and classify it
```

```
    label = model.predict(X_test[i:i+1])
```

```
    image = X_test[i].reshape((8, 8))
```

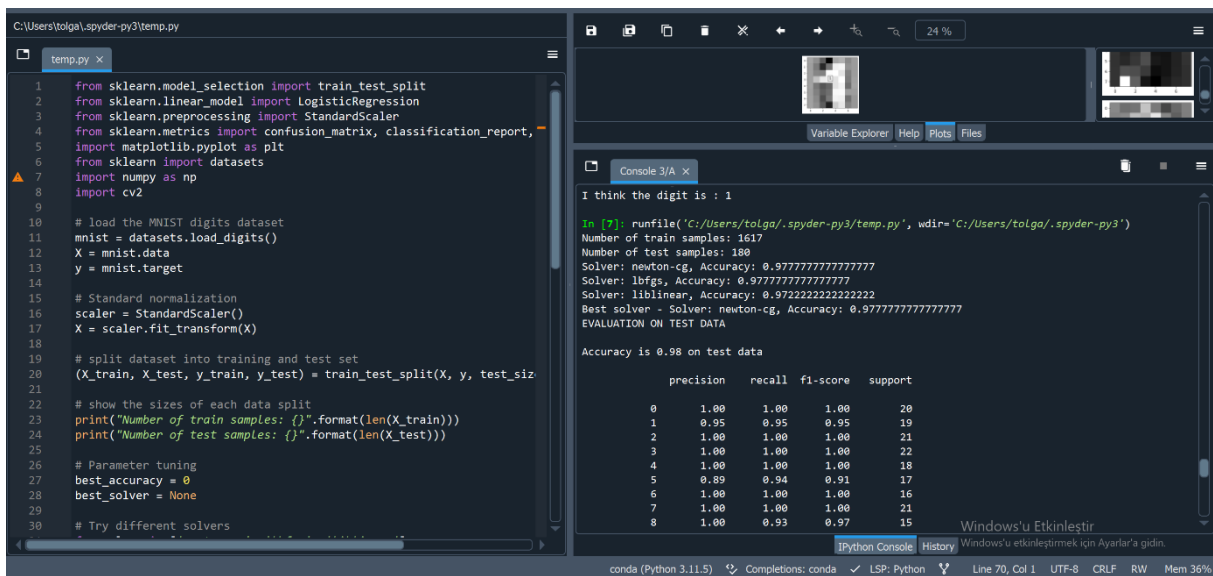
```
    plt.imshow(image, cmap='gray')
```

```
    plt.annotate(label[0], (3, 3), bbox={'facecolor': 'white'}, fontsize=16)
```

```
    print("I think the digit is : {}".format(label[0]))
```

```
    plt.show()
```

```
    cv2.waitKey(0)
```



The screenshot displays the Spyder Python IDE interface. The left pane shows a Python script named 'temp.py' with the following code:

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.metrics import confusion_matrix, classification_report,
5 import matplotlib.pyplot as plt
6 from sklearn import datasets
7 import numpy as np
8 import cv2
9
10 # load the MNIST digits dataset
11 mnist = datasets.load_digits()
12 X = mnist.data
13 y = mnist.target
14
15 # Standard normalization
16 scaler = StandardScaler()
17 X = scaler.fit_transform(X)
18
19 # split dataset into training and test set
20 (X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.2)
21
22 # show the sizes of each data split
23 print("Number of train samples: {}".format(len(X_train)))
24 print("Number of test samples: {}".format(len(X_test)))
25
26 # Parameter tuning
27 best_accuracy = 0
28 best_solver = None
29
30 # Try different solvers
```

The right pane shows the IPython console output. It displays the results of running the script, including the number of train and test samples, the accuracy of different solvers, and a table of confusion matrix results for digits 0 through 8.

```
In [7]: runfile('C:/Users/tolga/.spyder-py3/temp.py', wdir='C:/Users/tolga/.spyder-py3')
Number of train samples: 1617
Number of test samples: 180
Solver: newton-cg, Accuracy: 0.9777777777777777
Solver: lbfgs, Accuracy: 0.9777777777777777
Solver: liblinear, Accuracy: 0.9722222222222222
Best solver - Solver: newton-cg, Accuracy: 0.9777777777777777
EVALUATION ON TEST DATA

Accuracy is 0.98 on test data
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	0.95	0.95	0.95	19
2	1.00	1.00	1.00	21
3	1.00	1.00	1.00	22
4	1.00	1.00	1.00	18
5	0.89	0.94	0.91	17
6	1.00	1.00	1.00	16
7	1.00	1.00	1.00	21
8	1.00	0.93	0.97	15

The bottom status bar indicates the environment is 'conda (Python 3.11.5)' and shows various settings like 'Line 70, Col 1' and 'Mem 36%'.

C:\Users\tolga\spyder-py3\temp.py

temp.py x

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.metrics import confusion_matrix, classification_report,
5 import matplotlib.pyplot as plt
6 from sklearn import datasets
7 import numpy as np
8 import cv2
9
10 # load the MNIST digits dataset
11 mnist = datasets.load_digits()
12 X = mnist.data
13 y = mnist.target
14
15 # Standard normalization
16 scaler = StandardScaler()
17 X = scaler.fit_transform(X)
18
19 # split dataset into training and test set
20 (X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.2)
21
22 # show the sizes of each data split
23 print("Number of train samples: {}".format(len(X_train)))
24 print("Number of test samples: {}".format(len(X_test)))
25
26 # Parameter tuning
27 best_accuracy = 0
28 best_solver = None
29
30 # Try different solvers
```

Variable Explorer | Help | Plots | Files

Console 3/A x

Best solver - Solver: newton-cg, Accuracy: 0.9777777777777777
EVALUATION ON TEST DATA

Accuracy is 0.98 on test data

	precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	0.95	0.95	0.95	19
2	1.00	1.00	1.00	21
3	1.00	1.00	1.00	22
4	1.00	1.00	1.00	18
5	0.89	0.94	0.91	17
6	1.00	1.00	1.00	16
7	1.00	1.00	1.00	21
8	1.00	0.93	0.97	15
9	0.91	0.91	0.91	11
accuracy			0.98	180
macro avg	0.97	0.97	0.97	180
weighted avg	0.98	0.98	0.98	180

Confusion matrix
[[20 0 0 0 0 0 0 0 0 0]]

Windows'u Etkinleştir
Windows'u etkinleştirmek için Ayarlar'a gidin.

IPython Console | History

conda (Python 3.11.5) Completions: conda LSP: Python Line 70, Col 1 UTF-8 CRLF RW Mem 36%