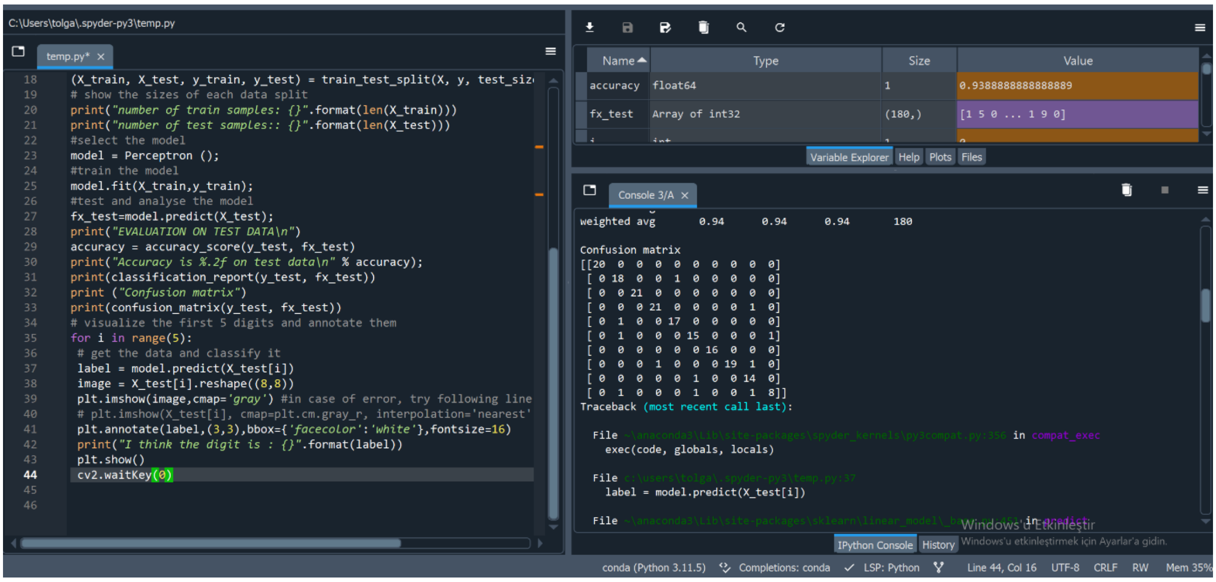
i) Running the example code results in the following confusion matrix



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

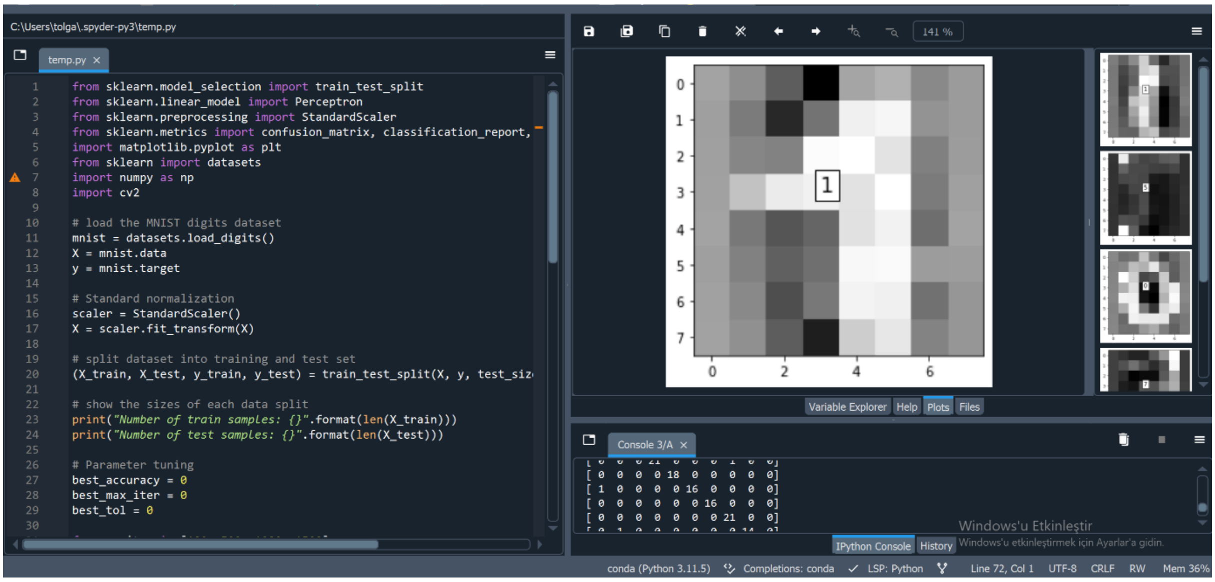
load the MNIST digits dataset

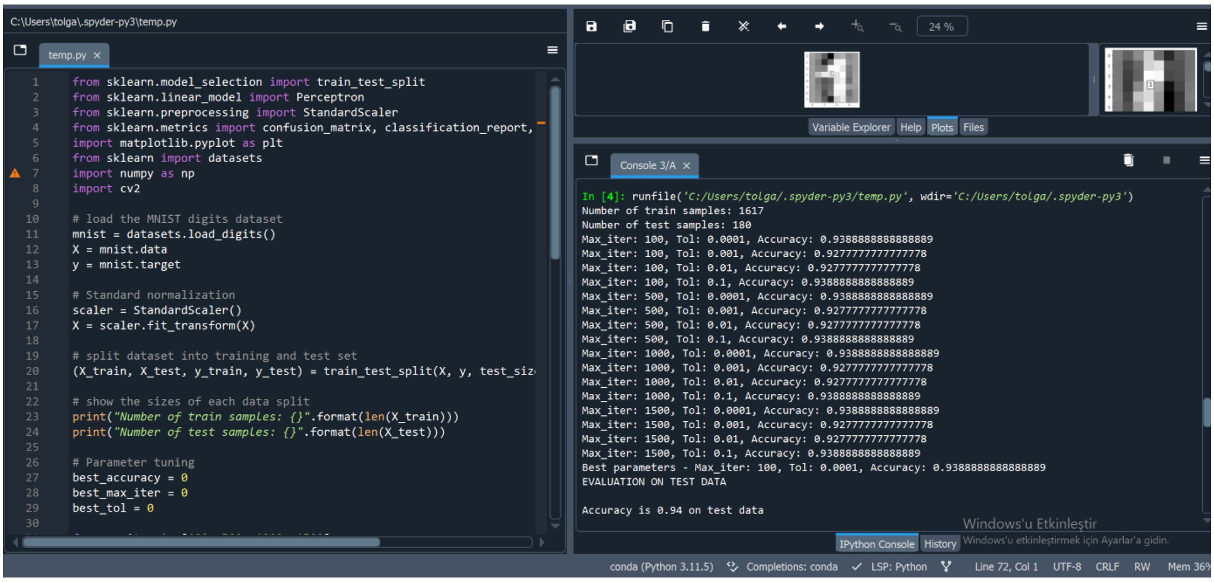
mnist = datasets.load_digits()

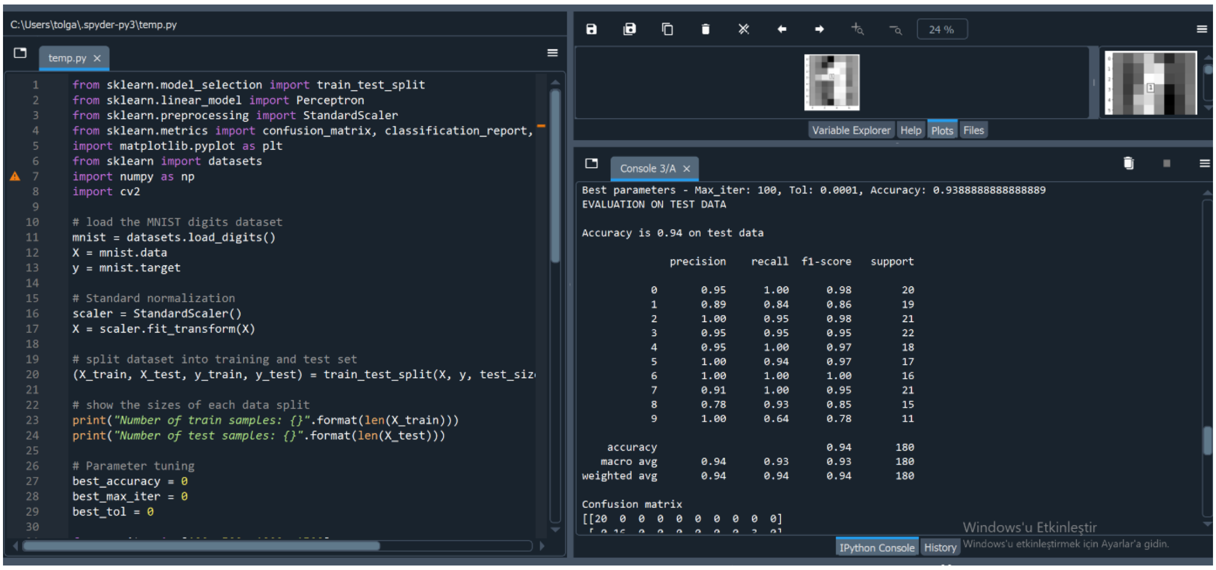
X = mnist.data

```
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)
```


C:\Users\tolga\.spyder-py3\temp.py
temp.py
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,
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_siz
# 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
141 %
1
Variable Explorer
Help
Plots
Files
Console 3/A
[0 0 0 0 18 0 0 0 0 0]
[1 0 0 0 0 16 0 0 0 0]
[0 0 0 0 0 0 16 0 0 0]
[0 0 0 0 0 0 0 21 0 0]
IPython Console
History
Windows'u Etkinleştir
Windows'u etkinleştirmek için Ayarlar'a gidin.
conda (Python 3.11.5)
Completions: conda
LSP: Python
Line 72, Col 1
UTF-8
CRLF
RW
Mem 36%

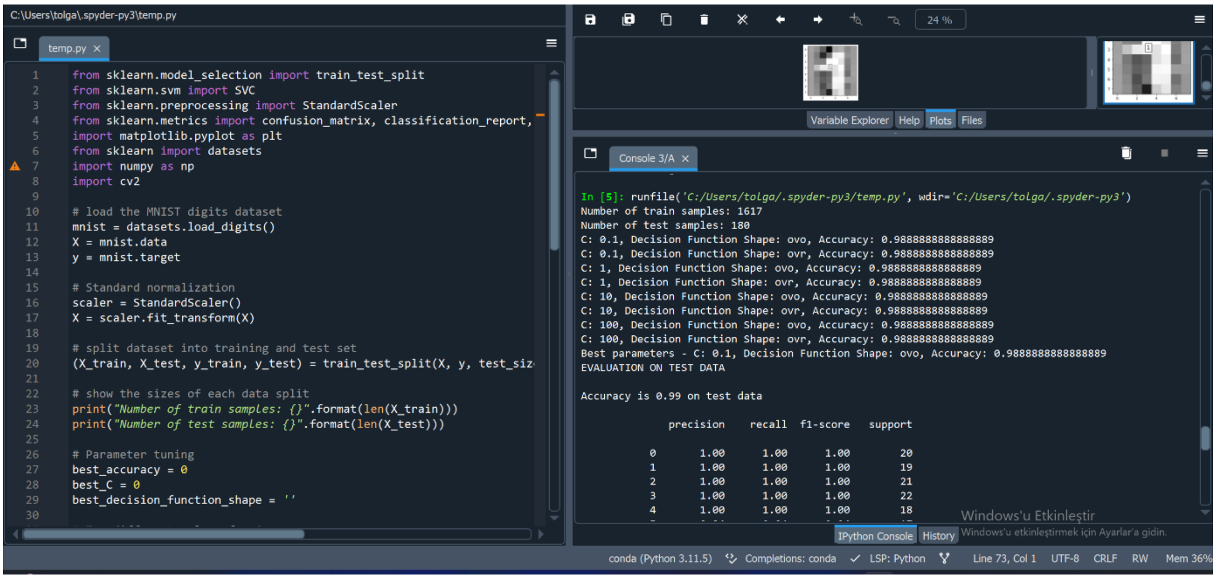

C:\Users\tolga\.spyder-py3\temp.py
temp.py
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report,
import matplotlib.pyplot as plt
from sklearn 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_siz
# 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
24 %
Variable Explorer
Help
Plots
Files
Console 3/A
In [4]: runfile('C:/Users/tolga/.spyder-py3/temp.py', wdir='C:/Users/tolga/.spyder-py3')
Number of train samples: 1617
Number of test samples: 180
Max_iter: 100, Tol: 0.0001, Accuracy: 0.9388888888888889
Max_iter: 100, Tol: 0.001, Accuracy: 0.9277777777777778
Max_iter: 100, Tol: 0.01, Accuracy: 0.9277777777777778
Max_iter: 100, Tol: 0.1, Accuracy: 0.9388888888888889
Max_iter: 500, Tol: 0.0001, Accuracy: 0.9388888888888889
Max_iter: 500, Tol: 0.001, Accuracy: 0.9277777777777778
Max_iter: 500, Tol: 0.01, Accuracy: 0.9277777777777778
Max_iter: 500, Tol: 0.1, Accuracy: 0.9388888888888889
Max_iter: 1000, Tol: 0.0001, Accuracy: 0.9388888888888889
Max_iter: 1000, Tol: 0.001, Accuracy: 0.9277777777777778
Max_iter: 1000, Tol: 0.01, Accuracy: 0.9277777777777778
Max_iter: 1000, Tol: 0.1, Accuracy: 0.9388888888888889
Max_iter: 1500, Tol: 0.0001, Accuracy: 0.9388888888888889
Max_iter: 1500, Tol: 0.001, Accuracy: 0.9277777777777778
Max_iter: 1500, Tol: 0.01, Accuracy: 0.9277777777777778
Max_iter: 1500, Tol: 0.1, Accuracy: 0.9388888888888889
Best parameters - Max_iter: 100, Tol: 0.0001, Accuracy: 0.9388888888888889
EVALUATION ON TEST DATA
Accuracy is 0.94 on test data
IPython Console
History
Windows'u Etkinleştir
Windows'u etkinleştirmek için Ayarlar'a gidin.
conda (Python 3.11.5)
Completions: conda
LSP: Python
Line 72, Col 1
UTF-8
CRLF
RW
Mem 36%

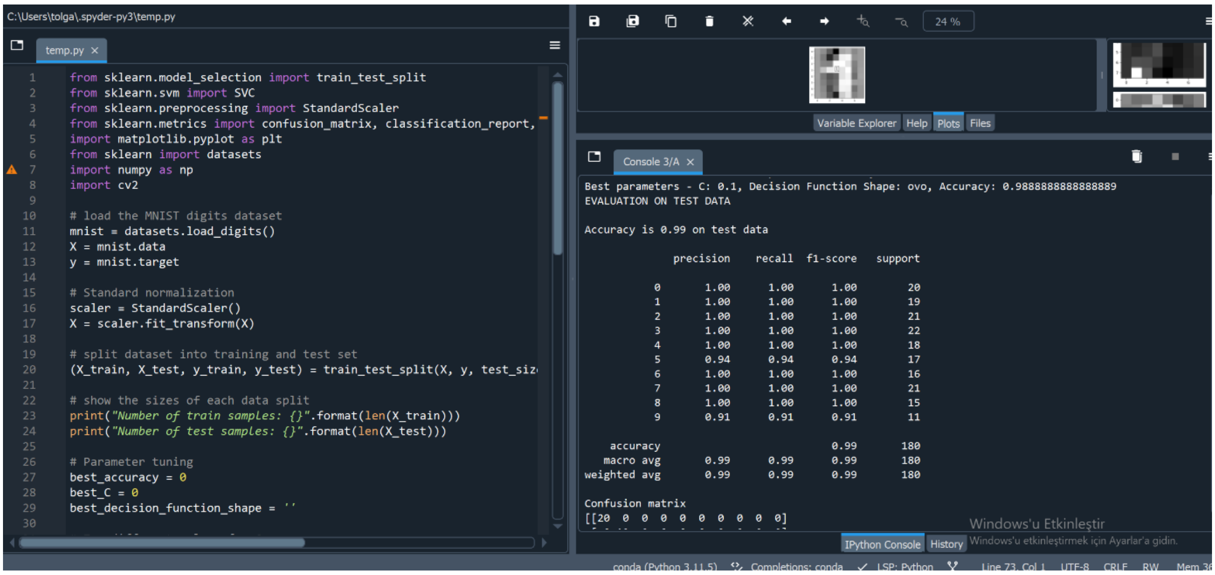


"This code includes standard normalization and performs a grid search on the max_iter and tol parameters for the Perceptron model. The goal is to determine the combination of these parameters that provides the best performance. The best selected parameters are then used to train the final model and the 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")

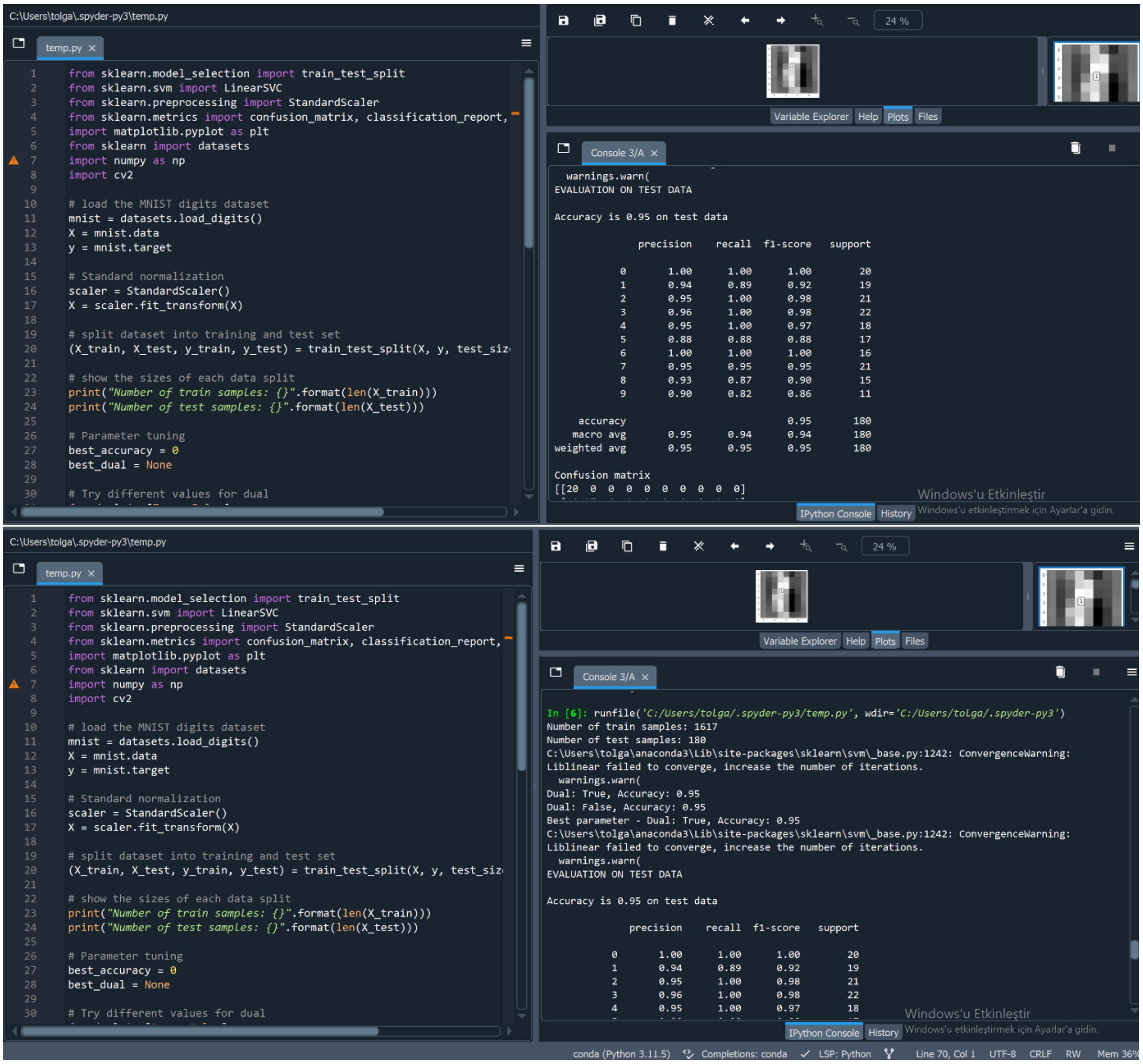




This code loops over different C and decision_function_shape values and reports the best parameters based on accuracy. The best parameters are then used to train the final SVC model and its performance is evaluated on test data. Adjustments to C and decision function shape values can have varying effects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



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)

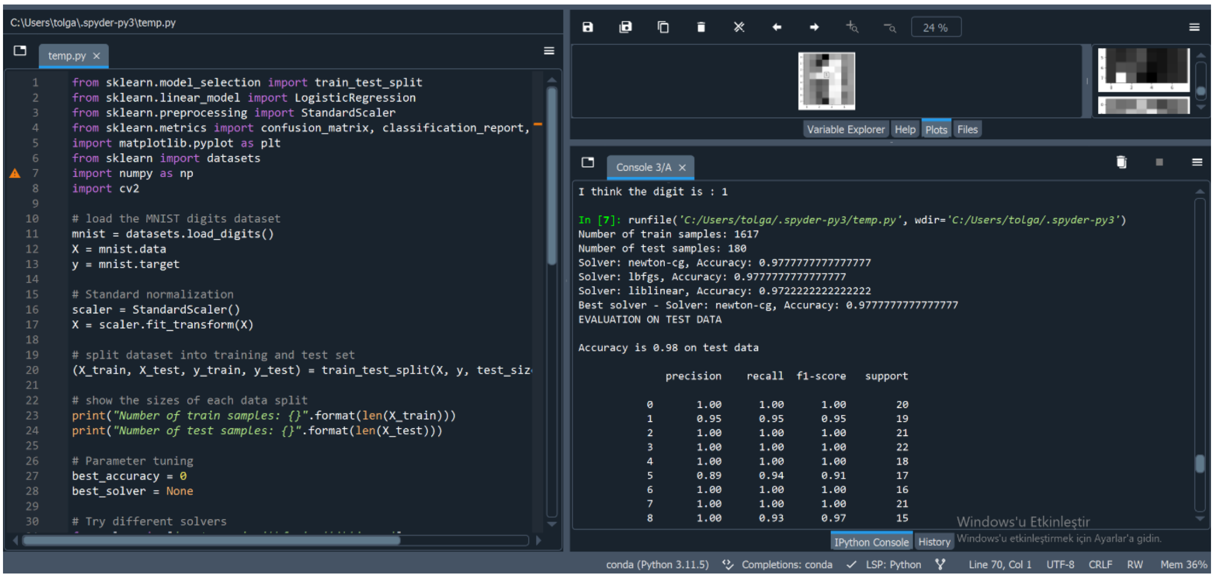
Code using Logistic Regression using different solvers ('newton-cg', 'lbfgs', 'liblinear'). The best dissolution values will be determined accordingly.

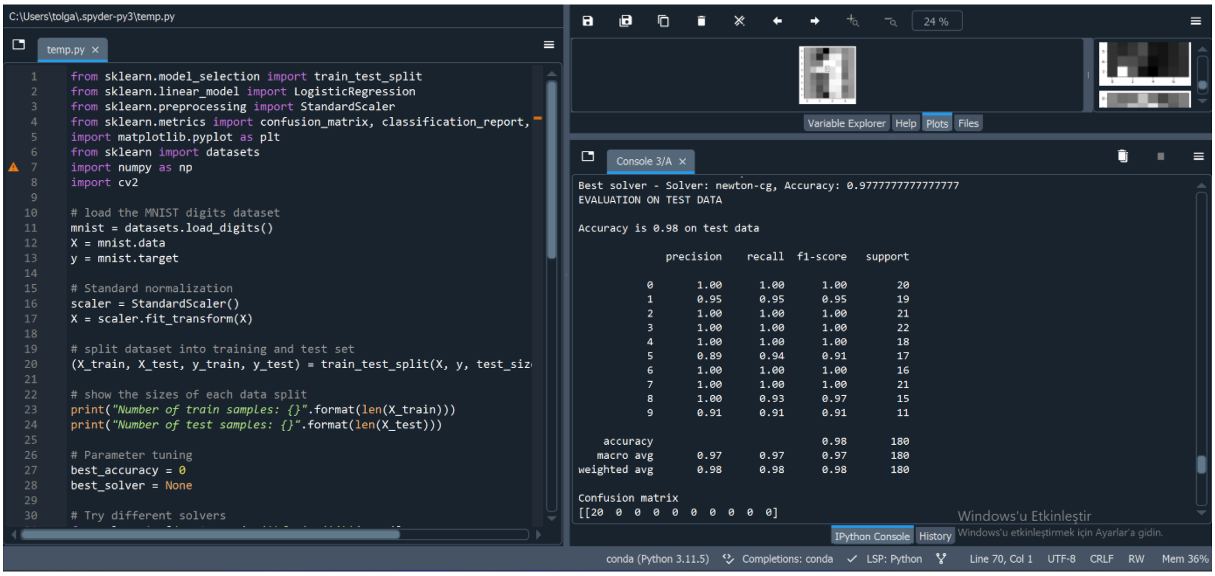
This code tests Logistic Regression with different solvers and reports accuracy, basic metric and good solver. Since Logistic Regression's default solver is 'lbfgs', the default values will be determined based on the best solver.

```
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)
```





C:\Users\tolga\.spyder-py3\temp.py
temp.py
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,
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_siz
# 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
24 %
Variable Explorer
Help
Plots
Files
Console 3/A
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%