EX. NO: 3 — STUDY OF THE CLASSIFIERS WITH RESPECT TO STATISTICAL PARAMETERS

07.08.25

AIM

To study and compare the performance of different classification algorithms using statistical parameters.

OBJECTIVE

To implement and train classifiers (Decision tree), on the digits dataset.

To evaluate and compare the performance of the classification using statistical metrics.

To understand how different algorithms behave in terms of classification accuracy and error distribution.

Algorithm used

1. Decision Tree classifier
2. SVM
3. Logistic regression

pseudocode

```
1. Decision Tree classifier
    → Load the digits dataset.
    → Split the dataset into training and testing sets.
    → Initialize the decision tree classifier
    → Fit the classifier on training data.
    → predict labels for the test data
    → Evaluate the model using accuracy, confusion matrix and classification report.
```

```
Accuracy: 0.872

confusion matrix

array([[29, 0, 1, 0, 2, 1, 0, 0, 0, 0],
       [0, 22, 1, 0, 1, 0, 1, 4, 1, 1],
       [0, 1, 25, 3, 0, 0, 2, 1, 1, 0],
       [0, 0, 0, 29, 0, 0, 1, 1, 3, 0],
       [0, 0, 0, 0, 42, 1, 2, 1, 0, 0],
       [0, 0, 1, 0, 1, 42, 1, 0, 1, 1],
       [0, 0, 0, 0, 1, 0, 34, 0, 0, 0],
       [0, 0, 0, 2, 2, 0, 0, 29, 1, 0],
       [0, 1, 0, 3, 2, 1, 0, 1, 20, 2],
       [0, 0, 0, 3, 2, 1, 0, 1, 0, 33]])
```

2. Support vector machine

⇒ Load the digits dataset.

⇒ Split the dataset into training and testing sets

⇒ Initialize the SVM classifier

⇒ Fit the classifier on training data.

⇒ predict labels for the test data

⇒ Evaluate the model using accuracy, confusion matrix and classification report.

3 Logistic regression

⇒ Load the digits dataset

⇒ Split the dataset into training and testing sets.

⇒ Initialize Logistic regression classifier with a suitable max_iter

⇒ Fit the classifier on training data

⇒ Evaluate the model using accuracy, confusion matrix, and classification report.

Overall

✱ SVM provides the best overall classification performance on the digits dataset.

✱ Logistic Regression is very close in performance, and significantly better than Decision Tree.

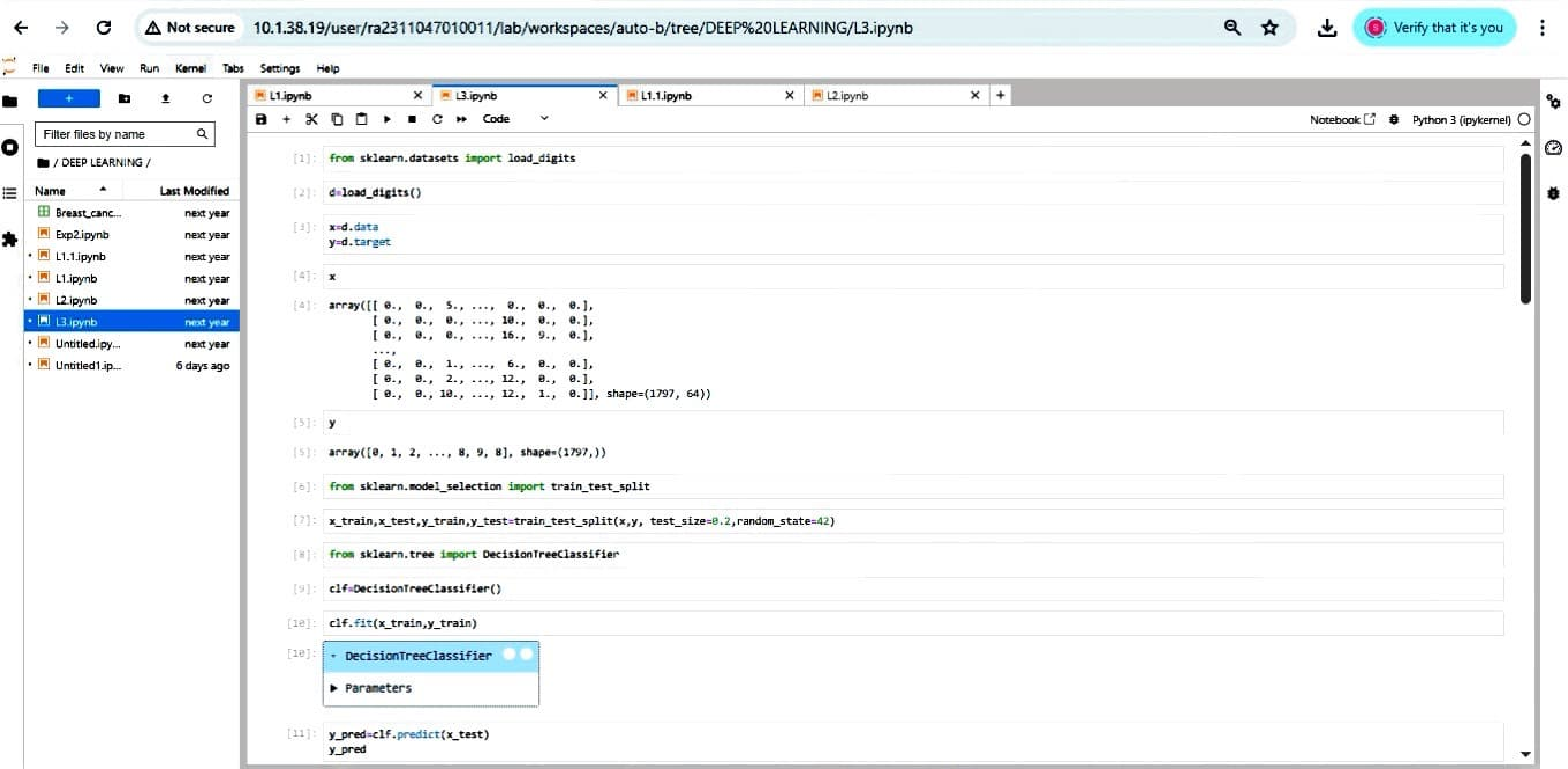
✱ Decision Tree shows some signs of overfitting or misclassification in certain digits, especially where shapes are visually similar.

Result

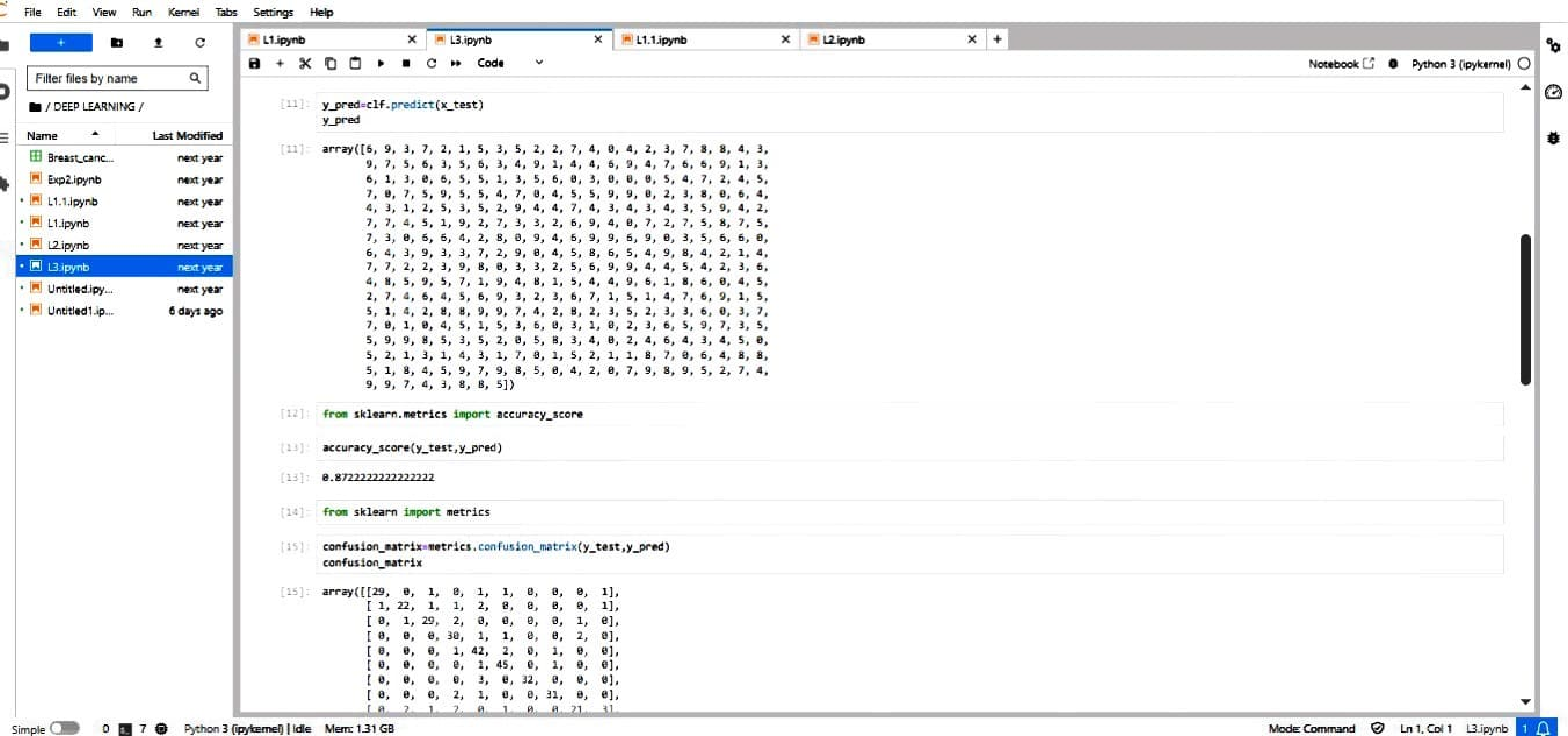
The support vector machine classifier performed the best with 98.61% accuracy, followed by Logistic regression with 97.5%, and Decision Tree with 84.72%. This shows that SVM and Logistic regression are more efficient for high dimensional database like handwritten digits compared to a basic decision tree.

OBSERVATION

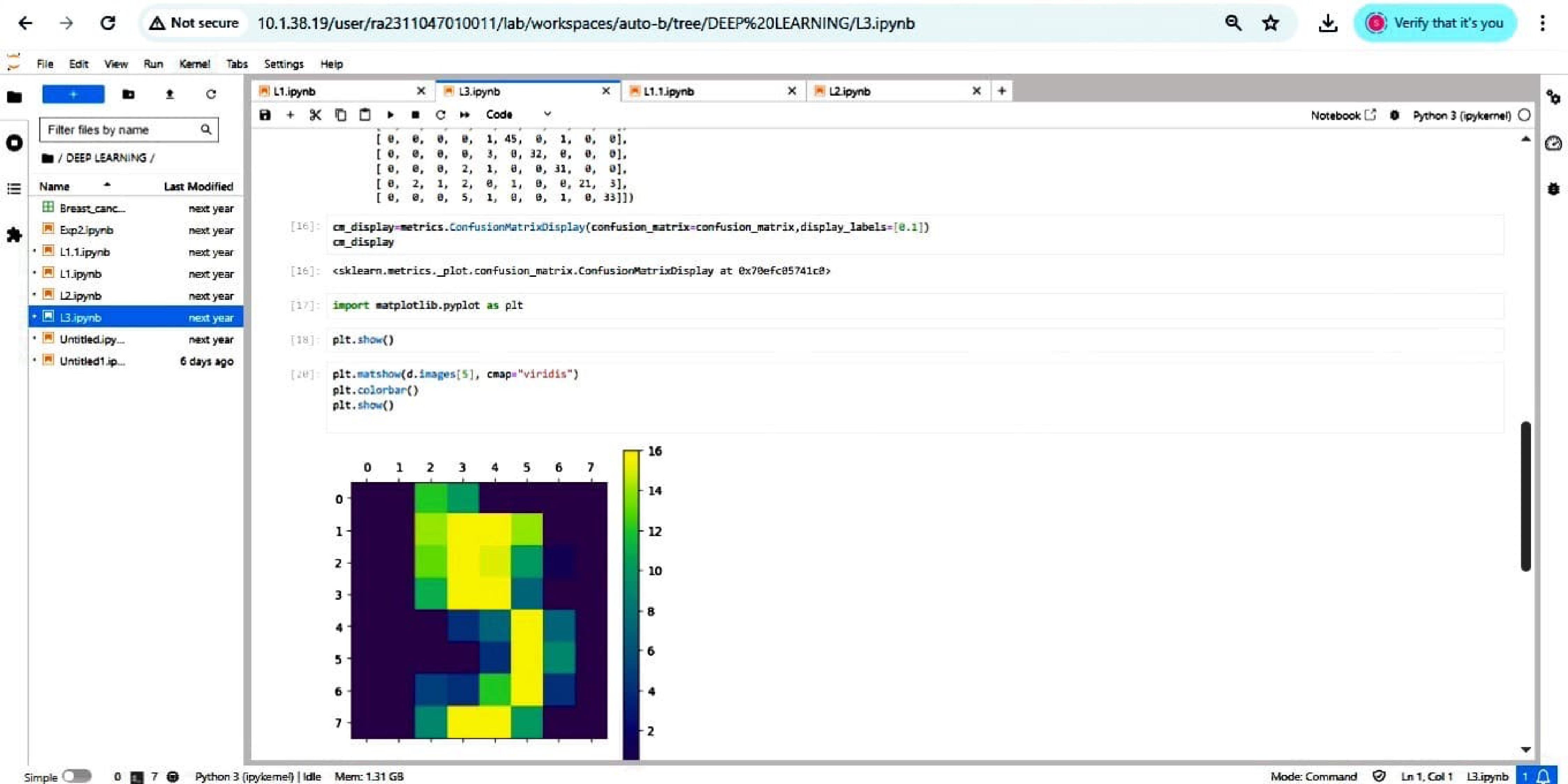
CLASSIFIER	Decision Tree	SVM	Logistic Regression
Accuracy	84.72%	98.61%	97.50%
Macro Avg precision	85.30%	98.72%	97.67%
Macro Avg Recall	83.98%	98.66%	97.65%
macro Avg F1-score	84.81%	98.66%	97.65%
Weighted Avg F1-score	84.72%	98.61	97.51%


10.1.38.19/user/ra2311047010011/lab/workspaces/auto-b/tree/DEEP%20LEARNING/L3.ipynb
File Edit View Run Kernel Tabs Settings Help
L1.ipynb L3.ipynb L1.1.ipynb L2.ipynb
/ DEEP LEARNING /
Breast_canc... Exp2.ipynb L1.1.ipynb L1.ipynb L2.ipynb L3.ipynb Untitled.ipy... Untitled1.ip...
Python 3 (ipykernel)

```
[1]: from sklearn.datasets import load_digits

[2]: d=load_digits()

[3]: x=d.data
     y=d.target

[4]: x

[4]: array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
            [ 0.,  0.,  0., ..., 10.,  0.,  0.],
            [ 0.,  0.,  0., ..., 16.,  9.,  0.],
            ...,
            [ 0.,  0.,  1., ...,  6.,  0.,  0.],
            [ 0.,  0.,  2., ..., 12.,  0.,  0.],
            [ 0.,  0., 10., ..., 12.,  1.,  0.]], shape=(1797, 64))

[5]: y

[5]: array([0, 1, 2, ..., 8, 9, 8], shape=(1797,))

[6]: from sklearn.model_selection import train_test_split

[7]: x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.2,random_state=42)

[8]: from sklearn.tree import DecisionTreeClassifier

[9]: clf=DecisionTreeClassifier()

[10]: clf.fit(x_train,y_train)

[10]: DecisionTreeClassifier
      ▶ Parameters

[11]: y_pred=clf.predict(x_test)
      y_pred
```



```
[11]: y_pred=clf.predict(x_test)
      y_pred

[11]: array([6, 9, 3, 7, 2, 1, 5, 3, 5, 2, 2, 7, 4, 0, 4, 2, 3, 7, 8, 8, 4, 3,
             9, 7, 5, 6, 3, 5, 6, 3, 4, 9, 1, 4, 4, 6, 9, 4, 7, 6, 6, 9, 1, 3,
             6, 1, 3, 0, 6, 5, 5, 1, 3, 5, 6, 0, 3, 0, 0, 0, 5, 4, 7, 2, 4, 5,
             7, 0, 7, 5, 9, 5, 5, 4, 7, 0, 4, 5, 5, 9, 9, 0, 2, 3, 8, 0, 6, 4,
             4, 3, 1, 2, 5, 3, 5, 2, 9, 4, 4, 7, 4, 3, 4, 3, 4, 3, 5, 9, 4, 2,
             7, 7, 4, 5, 1, 9, 2, 7, 3, 3, 2, 6, 9, 4, 0, 7, 2, 7, 5, 8, 7, 5,
             7, 3, 0, 6, 6, 4, 2, 8, 0, 9, 4, 6, 9, 9, 6, 9, 0, 3, 5, 6, 6, 0,
             6, 4, 3, 9, 3, 3, 7, 2, 9, 0, 4, 5, 8, 6, 5, 4, 9, 8, 4, 2, 1, 4,
             7, 7, 2, 2, 3, 9, 8, 0, 3, 3, 2, 5, 6, 9, 9, 4, 4, 5, 4, 2, 3, 6,
             4, 8, 5, 9, 5, 7, 1, 9, 4, 8, 1, 5, 4, 4, 9, 6, 1, 8, 6, 0, 4, 5,
             2, 7, 4, 6, 4, 5, 6, 9, 3, 2, 3, 6, 7, 1, 5, 1, 4, 7, 6, 9, 1, 5,
             5, 1, 4, 2, 8, 8, 9, 9, 7, 4, 2, 8, 2, 3, 5, 2, 3, 3, 6, 0, 3, 7,
             7, 0, 1, 0, 4, 5, 1, 5, 3, 6, 0, 3, 1, 0, 2, 3, 6, 5, 9, 7, 3, 5,
             5, 9, 9, 8, 5, 3, 5, 2, 0, 5, 8, 3, 4, 0, 2, 4, 6, 4, 3, 4, 5, 0,
             5, 2, 1, 3, 1, 4, 3, 1, 7, 0, 1, 5, 2, 1, 1, 8, 7, 0, 6, 4, 8, 8,
             5, 1, 8, 4, 5, 9, 7, 9, 8, 5, 0, 4, 2, 0, 7, 9, 8, 9, 5, 2, 7, 4,
             9, 9, 7, 4, 3, 8, 8, 5])

[12]: from sklearn.metrics import accuracy_score

[13]: accuracy_score(y_test,y_pred)

[13]: 0.8722222222222222

[14]: from sklearn import metrics

[15]: confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
      confusion_matrix

[15]: array([[29,  0,  1,  0,  1,  1,  0,  0,  0,  1],
             [ 1, 22,  1,  1,  2,  0,  0,  0,  0,  1],
             [ 0,  1, 29,  2,  0,  0,  0,  0,  1,  0],
             [ 0,  0,  0, 30,  1,  1,  0,  0,  2,  0],
             [ 0,  0,  0,  1, 42,  2,  0,  1,  0,  0],
             [ 0,  0,  0,  0,  1, 45,  0,  1,  0,  0],
             [ 0,  0,  0,  0,  3,  0, 32,  0,  0,  0],
             [ 0,  0,  0,  2,  1,  0,  0, 31,  0,  0],
             [ 0,  2,  1,  2,  0,  1,  0,  0, 21,  3],
```



```
       [ 0,  0,  0,  0,  1, 45,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  3,  0, 32,  0,  0,  0],
       [ 0,  0,  0,  2,  1,  0,  0, 31,  0,  0],
       [ 0,  2,  1,  2,  0,  1,  0,  0, 21,  3],
       [ 0,  0,  0,  5,  1,  0,  0,  1,  0, 33]])
```

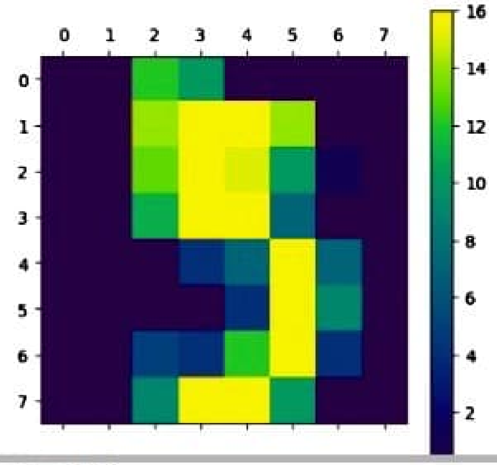
```
[16]: cm_display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0.1])
      cm_display
```

```
[16]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x70efc05741c0>
```

```
[17]: import matplotlib.pyplot as plt
```

```
[18]: plt.show()
```

```
[20]: plt.matshow(d.images[5], cmap="viridis")
      plt.colorbar()
      plt.show()
```


0
1
2
3
4
5
6
7
16
14
12
10
8
6
4
2