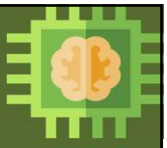
Elective Course
Course Code: CS4103
Autumn 2025-26



Lecture #41

Artificial Intelligence for Data Science

Week-12:

MACHINE LEARNING (Part IX)

Exploring Multi-Layer Perceptron using Python (Scikit-learn)

Course Instructor:

Dr. Monidipa Das

Assistant Professor

Department of Computational and Data Sciences

Indian Institute of Science Education and Research Kolkata, India 741246

MLPClassifier from Scikit-learn

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,),
activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5,
max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False,
warm_start=False, momentum=0.9, nesterovs_momentum=True,
early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

```
from sklearn.neural_network import MLPClassifier
```

MLP for Iris Classification: Dataset Loading


```
import pandas as pd

#Load the dataset
dataset =
pd.read_csv('F:/CS4103/code/iris.csv')

#View dataset
print("First 6 rows of the dataset:")
print(dataset.head(6))

#Dataset column names
print("\nName of the columns in the dataset:")
print(list(dataset.columns))

input_features = list(dataset.columns[:-1])
class_col=dataset.columns[-1]
print("Name of the input feature columns:",input_features)
print("Name of the class/label column:",class_col)
```

```
First 6 rows of the dataset:
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
5           5.4          3.9           1.7          0.4  Setosa
```

```
Name of the columns in the dataset:
['sepal.length', 'sepal.width',
'petal.length', 'petal.width', 'variety']
Name of the input feature columns:
['sepal.length', 'sepal.width',
'petal.length', 'petal.width']
Name of the class/label column: variety
```


MLP for Iris Classification: Exploring Dataset Details


```
# Number of rows or instances and number of columns features
print("\nTotal number of data points/examples/instances:", dataset.shape[0])
print("Total number of input features:", dataset.shape[1]-1)

#Dataset description
print("\nDescription of the dataset:")
print(dataset.describe())

#Print number of classes
c=dataset[class_col].nunique()
print("\nNumber of classes(discrete labels):",c)

#Number of instances per class
print("\nSample count per class:")
print(dataset[class_col].value_counts())

#Check missing values in variables
dataset.isnull().sum()
```

```
Total number of data points/examples/instances: 150
Total number of input features: 4
```

```
Description of the dataset:
       sepal.length  sepal.width  petal.length  petal.width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

```
Number of classes (discrete labels): 3

Sample count per class:
Setosa        50
Versicolor    50
Virginica     50
Name: variety, dtype: int64
```

```
sepal.length    0
sepal.width     0
petal.length    0
petal.width     0
variety         0
dtype: int64
```

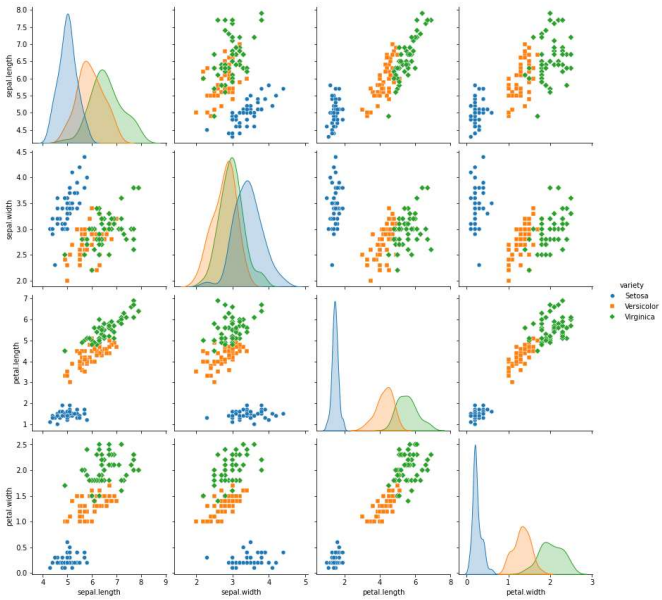
MLP for Iris Classification: Data Split

```
X = dataset[input_features].values
y = dataset[class_col].values

#split dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)

#check the shape of X_train and X_test
print("Traning set shape:",X_train.shape,"\nTest set shape:",X_test.shape)
```

```
Traning set shape: (120, 4)
Test set shape: (30, 4)
```


MLP for Iris Classification: Data Visualization

```
#visualize the distribution
of features and their
relationship with others

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure()
sns.pairplot(dataset, hue =
class_col, size=3, markers=["o",
"s", "D"])
plt.show()
```



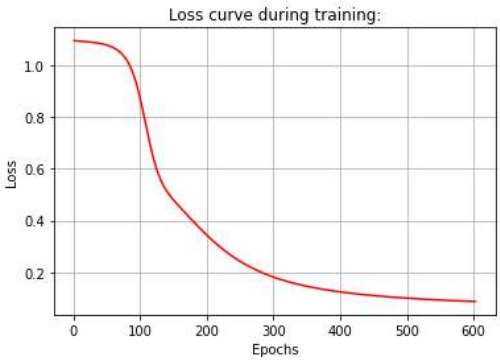
MLP for Iris Classification: Building Classifier

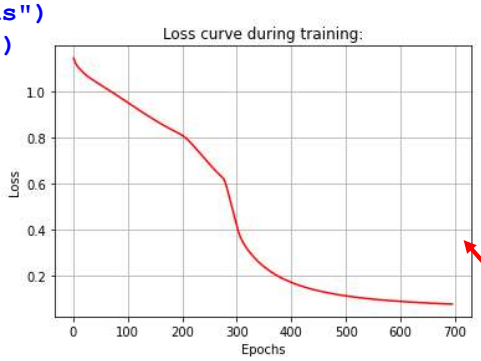
```
#import MLPClassifier
from sklearn.neural_network import MLPClassifier

#instantiate the MLPClassifier model
mlp =
MLPClassifier(hidden_layer_sizes=(10, 5),
activation='logistic',
solver='sgd',
learning_rate_init=0.05,
max_iter=1000,
verbose=True,
random_state=0)

#fit the model
mlp.fit(X_train, y_train)
```

```
Iteration 581, loss = 0.08967096
Iteration 582, loss = 0.08956557
Iteration 583, loss = 0.08946072
Iteration 584, loss = 0.08935640
Iteration 585, loss = 0.08925261
Iteration 586, loss = 0.08914935
Iteration 587, loss = 0.08904661
Iteration 588, loss = 0.08894438
Iteration 589, loss = 0.08884267
Iteration 590, loss = 0.08874147
Iteration 591, loss = 0.08864077
Iteration 592, loss = 0.08854058
Iteration 593, loss = 0.08844089
Iteration 594, loss = 0.08834169
Iteration 595, loss = 0.08824298
Iteration 596, loss = 0.08814475
Iteration 597, loss = 0.08804702
Iteration 598, loss = 0.08794976
Iteration 599, loss = 0.08785298
Iteration 600, loss = 0.08775667
Iteration 601, loss = 0.08766084
Iteration 602, loss = 0.08756547
Iteration 603, loss = 0.08747056
Training loss did not improve more
than tol=0.000100 for 10 consecutive
epochs. Stopping.
```


MLP for Iris Classification: Loss Curve During Training

```
#Plot loss curve during training
plt.figure()
plt.plot(range(1,len(mlp.loss_curve_)+1),
mlp.loss_curve_,color='green')
plt.grid(True)
plt.title("Loss curve during training:")
plt.xlabel("Epochs")
plt.ylabel("Loss")
```






```
mlp =
MLPClassifier(hidden_layer_s
izes=(10, 5), max_iter=1000,
verbose=True,
random_state=0)

...
```

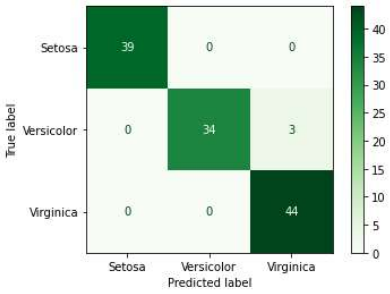
MLP for Iris Classification: Evaluating Classifier


```
#Evaluating predictions
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

predictions_train = mlp.predict(X_train)
print('Training set score:')
print(accuracy_score(y_train,predictions_train))

print('Test set score:')
predictions_test = mlp.predict(X_test)
print(accuracy_score(y_test,predictions_test))
```

```
Training set score:
0.975
Test set score:
1.0
```


MLP for Iris Classification: Evaluating Classifier

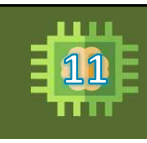

```
#Evaluating training performance
cm_train=confusion_matrix(y_train, predictions_train)
cm_pred=confusion_matrix(y_test, predictions_test)
plt.figure()
disp = ConfusionMatrixDisplay(
confusion_matrix=cm_train,display_labels=
dataset[class_col].unique())
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```



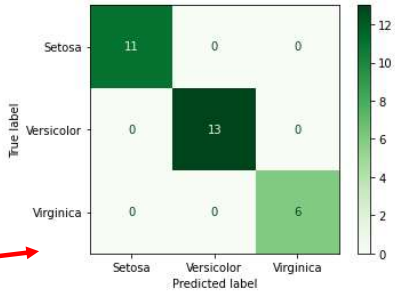

```
from sklearn.metrics import
classification_report
print("Classification Report for
Training:")
print(classification_report(y_train,
predictions_train))
```

```
Classification Report for Training:
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        39
  Versicolor       1.00      0.92      0.96        37
   Virginica       0.94      1.00      0.97        44

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120
```

MLP for Iris Classification: Evaluating Classifier



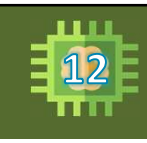

```
#Evaluating test performance
cm_train=confusion_matrix(y_train, predictions_train)
cm_pred=confusion_matrix(y_test, predictions_test)

plt.figure()
disp = ConfusionMatrixDisplay(
confusion_matrix=cm_pred,display_labels=
dataset[class_col].unique())
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```




```
from sklearn.metrics import
classification_report
print("Classification Report for
Testing:")
print(classification_report(y_test,
predictions_test))
```

```
Classification Report for Testing:
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        11
  Versicolor       1.00      1.00      1.00        13
   Virginica       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```


MLP for Iris Classification: Building Classifier with Validation Set

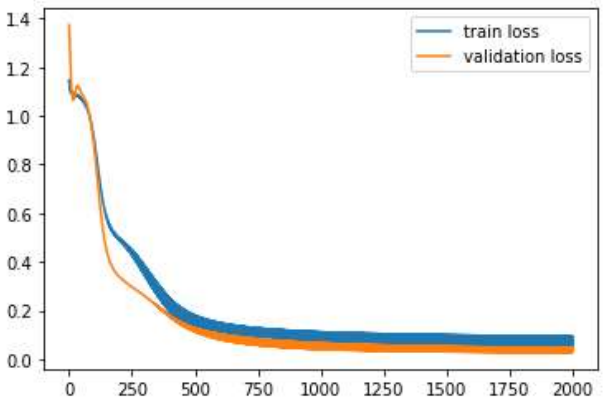



```
# Training the Model
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import log_loss

#instantiate the MLPClassifier model
X_train, X_hold, y_train, y_hold =
train_test_split(X, y, train_size=.7,random_state=42)
X_valid, X_test, y_valid, y_test =
train_test_split(X_hold, y_hold, train_size=.5,random_state=42)

clf = MLPClassifier(hidden_layer_sizes=(10, 5),
activation='logistic', solver='sgd',
learning_rate_init=0.04,max_iter=1000,verbose=True,random_state=42)

batch_size, train_loss_, valid_loss_ = 40, [], []
```

MLP for Iris Classification: Training loss vs. Validation loss

13

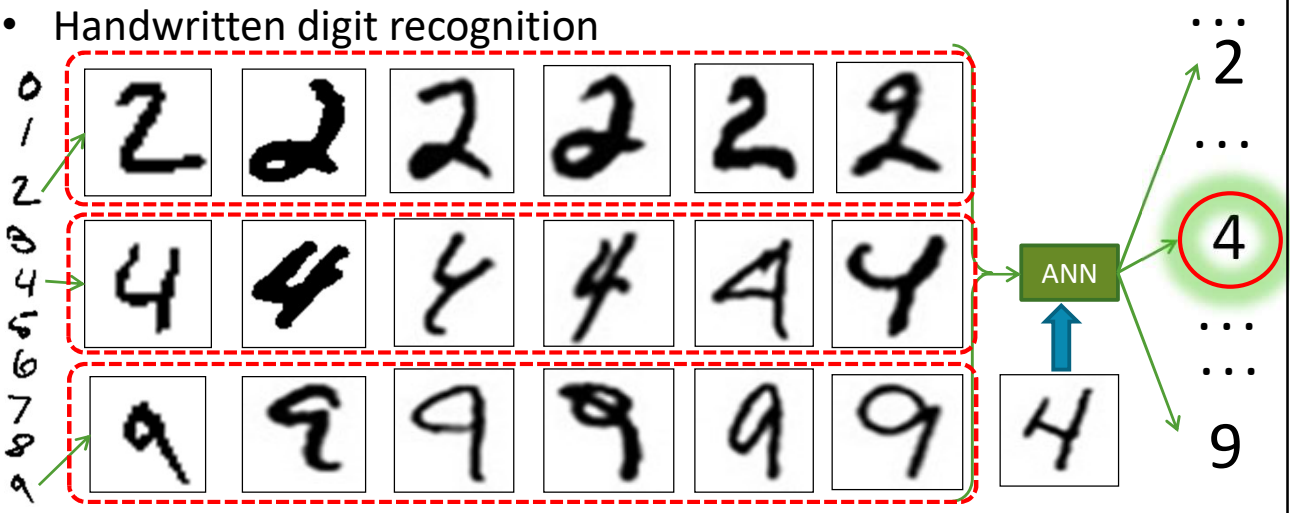
```
# Training Loop
for _ in range(1000):
    for b in range(batch_size,
len(y_train), batch_size):
        X_batch, y_batch = X_train[b-
batch_size:b], y_train[b-batch_size:b]
        clf.partial_fit(X_batch, y_batch,
classes=[0, 1, 2])

        train_loss_.append(log_loss(y_ba
tch, clf.predict_proba(X_batch)))

        valid_loss_.append(log_loss(y_va
lid, clf.predict_proba(X_valid)))

plt.plot(range(len(train_loss_)), train_loss_, label="train loss")
plt.plot(range(len(train_loss_)), valid_loss_, label="validation loss")
plt.legend()
```



Neural Network on Image Data

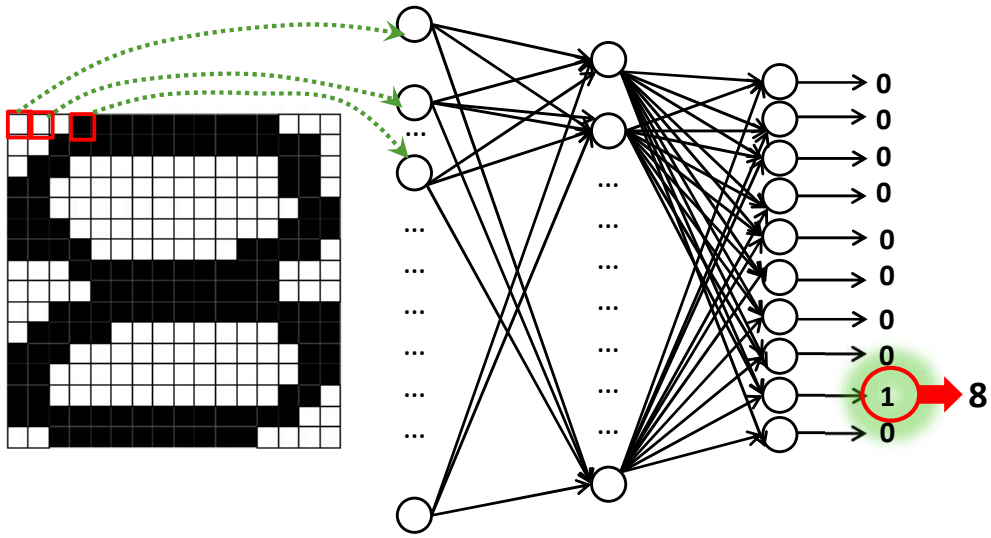
14

- Handwritten digit recognition



Neural Network on Image Data

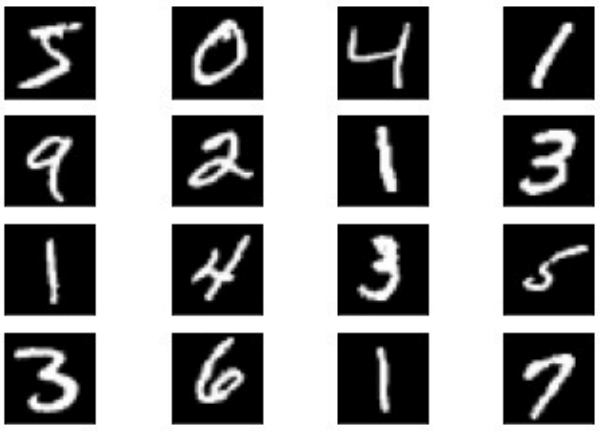
- Handwritten digit recognition



MLP for Handwritten-digit Classification: Dataset Loading


```
from sklearn.neural_network import MLPClassifier
import pickle
import matplotlib.pyplot as plt

with open("F:/CS4103/code/mnist.pkl","rb") as fh:
    train_set, validation_set, test_set = pickle.load(fh,encoding='latin1')


train_imgs,train_labels = train_set[0], train_set[1]
valid_imgs, valid_labels = validation_set[0], validation_set[1]
test_imgs, test_labels = test_set[0], test_set[1]


image_size = 28
no_of_different_labels = 10
image_pixels = image_size * image_size
```

MLP for Handwritten-digit Classification: Data Visualization



```
fig, axes = plt.subplots(4, 4)

# Flatten the axes array for easy
iteration
axes = axes.flatten()

#Plot a few training sample images
for i, ax in enumerate(axes):
    ax.imshow(train_imgs[i].reshape(
         image_size, image_size),
         cmap=plt.cm.gray)
    ax.set_xticks(())
    ax.set_yticks(())
```


MLP for Handwritten-digit Classification: Building Classifier and Evaluation

```
mlp = MLPClassifier(
         hidden_layer_sizes=(100,16),
          max_iter=480, alpha=1e-4,
          solver='sgd', verbose=2,
          tol=1e-4, random_state=1,
          learning_rate_init=.1)

train_labels =
train_labels.reshape(train_labels.shape[0],)
print(train_imgs.shape, train_labels.shape)

mlp.fit(train_imgs, train_labels)
print("Training set score: %f" %
mlp.score(train_imgs, train_labels))
print("Test set score: %f" %
mlp.score(test_imgs, test_labels))
```

```
(50000, 784) (50000,)
Iteration 1, loss = 0.38085539
Iteration 2, loss = 0.12810752
Iteration 3, loss = 0.09071036
Iteration 4, loss = 0.07330909
Iteration 5, loss = 0.05679172
Iteration 6, loss = 0.04689361
Iteration 7, loss = 0.03973453
Iteration 8, loss = 0.03317428
Iteration 9, loss = 0.02923015
Iteration 10, loss = 0.02402320
Iteration 11, loss = 0.02408135
Iteration 12, loss = 0.01974540
Iteration 13, loss = 0.01579598
Iteration 14, loss = 0.01272360
Iteration 15, loss = 0.01099683
Iteration 16, loss = 0.01341322
Iteration 17, loss = 0.01550250
Iteration 18, loss = 0.01278556
Iteration 19, loss = 0.00966535
Iteration 20, loss = 0.01254099
Iteration 21, loss = 0.01206898
Iteration 22, loss = 0.01353172
Iteration 23, loss = 0.01204964
Iteration 24, loss = 0.01129362
Iteration 25, loss = 0.01330346
Iteration 26, loss = 0.01325458
Iteration 27, loss = 0.01147159
Iteration 28, loss = 0.01516204
Iteration 29, loss = 0.01484799
Iteration 30, loss = 0.01083113
Training loss did not improve more
than tol=0.000100 for 10 consecutive
epochs. Stopping.
```

```
Training set score: 0.997420
Test set score: 0.976200
```

MLP for Handwritten-digit Classification: Classifier Evaluation


```
from sklearn.metrics import
classification_report

#Classification report for training
predictions_train = mlp.predict(train_imgs)
print("Classification Report for
Training:")
print(classification_report(train_labels,
predictions_train))
```

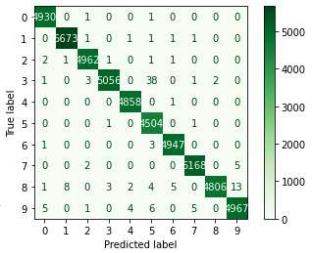
```
Classification Report for Training:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4932
           1       1.00      1.00      1.00      5678
           2       1.00      1.00      1.00      4968
           3       1.00      0.99      1.00      5101
           4       1.00      1.00      1.00      4859
           5       0.99      1.00      0.99      4506
           6       1.00      1.00      1.00      4951
           7       1.00      1.00      1.00      5175
           8       1.00      0.99      1.00      4842
           9       1.00      1.00      1.00      4988

    accuracy                           1.00     50000
   macro avg       1.00      1.00      1.00     50000
weighted avg       1.00      1.00      1.00     50000
```

```
#Classification report for testing
predictions_test = mlp.predict(test_imgs)
print("Classification Report for Testing:")
print(classification_report(test_labels,
predictions_test))
```

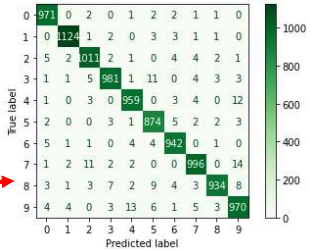
```
Classification Report for Testing:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.98      0.98      1032
           3       0.98      0.97      0.98      1010
           4       0.97      0.98      0.98       982
           5       0.96      0.98      0.97       892
           6       0.98      0.98      0.98       958
           7       0.98      0.97      0.97      1028
           8       0.99      0.96      0.97       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```


MLP for Handwritten-digit Classification: Classifier Evaluation


```
import numpy as np
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

cm_train=confusion_matrix(train_labels, predictions_train)
cm_pred=confusion_matrix(test_labels, predictions_test)
plt.figure()
disp = ConfusionMatrixDisplay(
confusion_matrix=
cm_train,display_labels=np.unique(train_labels))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```

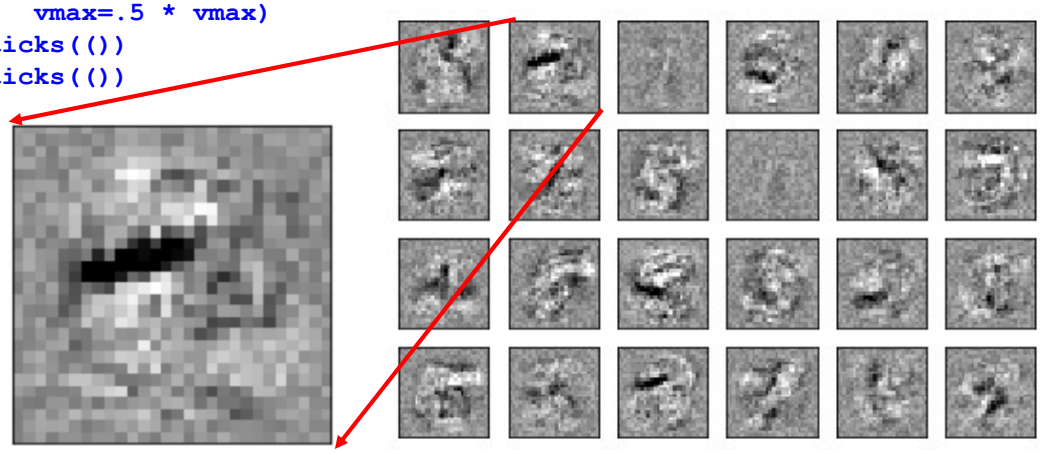


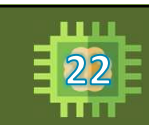

```
plt.figure()
disp = ConfusionMatrixDisplay(
confusion_matrix=
cm_pred,display_labels=np.unique(train_labels))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```



MLP for Handwritten-digit Classification: Visualizing Parameters


```
fig, axes = plt.subplots(4, 6)
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin,
               vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```





Questions?