**Assignment no 1**

**Assignment Name:-** Implement Linear Regression (Diabetes Dataset)

**Name:** Nishikant Pawar **Roll No:-**

**Class:-**Msc-1(cs) **Date:-**

**Code:-**

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets , linear\_model

diabetes = datasets.load\_diabetes()

diabetes\_X=diabetes.data[:, np.newaxis,2]

print(diabetes\_X)

diabetes\_X\_train = diabetes\_X[:-20]

diabetes\_X\_test = diabetes\_X[-20:]

diabetes\_Y\_train = diabetes.target[:-20]

diabetes\_Y\_test = diabetes.target[-20:]

regr=linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train,diabetes\_Y\_train)

print('coefficients: \n' , regr.coef\_)

print("Mean squared error : %.2f"

% np.mean((regr.predict(diabetes\_X\_test) - diabetes\_Y\_test ) \*\*2))

print('Varience score: %.2f' % regr.score(diabetes\_X\_test,diabetes\_Y\_test))

plt.scatter(diabetes\_X\_test, diabetes\_Y\_test, color='black')

plt.plot(diabetes\_X\_test, regr.predict(diabetes\_X\_test), color ='blue',linewidth=3)

plt.xticks(())

plt.yticks(())

plt.show()

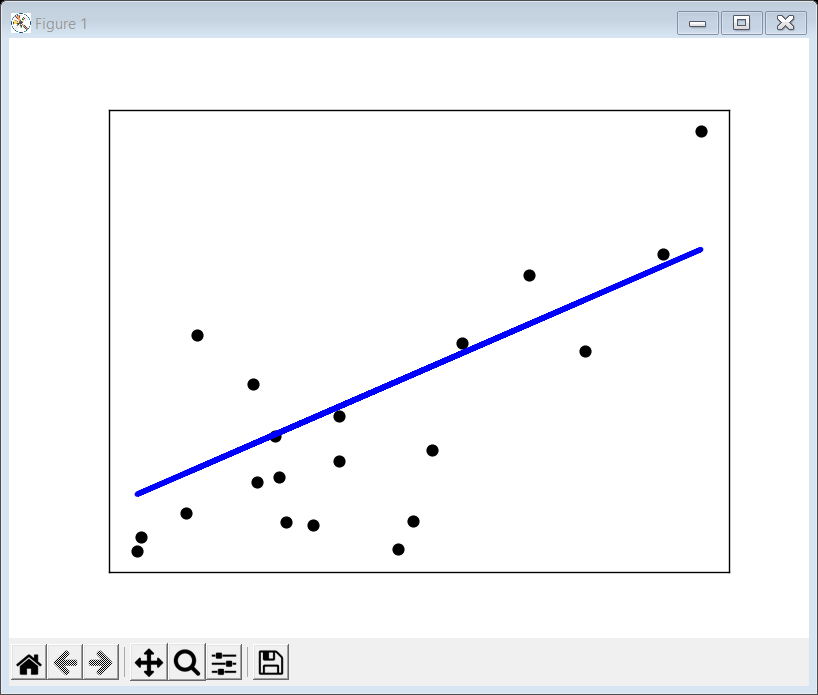
**Output:-**

coefficients:

[938.23786125]

Mean squared error : 2548.07

Varience score: 0.47



**Assignment no 2**

**Assignment Name:-** Implement Logistic Regression (Iris Dataset)

**Name:** nishikanrt Pawar **Roll No:-**

**Class:-** Msc-1(cs) **Date:-**

**Code:-**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn import datasets

# import some data to play with

iris = datasets.load\_iris()

X = iris.data[:, :2] # we only take the first two features.

Y = iris.target

# Create an instance of Logistic Regression Classifier and fit the data.

logreg = LogisticRegression(C=1e5)

logreg.fit(X, Y)

# Plot the decision boundary. For that, we will assign a color to each

# point in the mesh [x\_min, x\_max]x[y\_min, y\_max].

x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

h = 0.02 # step size in the mesh

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

Z = logreg.predict(np.c\_[xx.ravel(), yy.ravel()])

# Put the result into a color plot

Z = Z.reshape(xx.shape)

plt.figure(1, figsize=(4, 3))

plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points

plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors="k", cmap=plt.cm.Paired)

plt.xlabel("Sepal length")

plt.ylabel("Sepal width")

plt.xlim(xx.min(), xx.max())

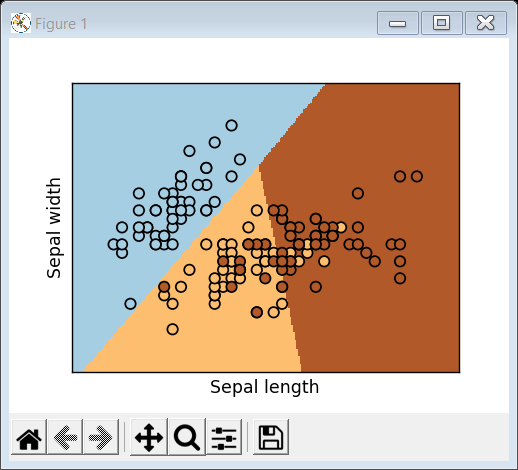
plt.ylim(yy.min(), yy.max())

plt.xticks(())

plt.yticks(())

plt.show()

**Output:-**

****

**Assignment no 3**

**Assignment Name:-** Implements Multinomial Logistic Regression (IrisDataset)

**Name:** nishikanrt Pawar **Roll No:-**

**Class:-** Msc-1(cs) **Date:-**

**Code:-**

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.cm as cm

import random

import seaborn

seaborn.set(style='whitegrid'); seaborn.set\_context('talk')

#matplotlib inline

#config InlineBackend.figure\_format = 'retina'

from sklearn.datasets import load\_iris

iris\_data = load\_iris()

print(iris\_data['DESCR'])

n\_samples, n\_features = iris\_data.data.shape

def Show\_Diagram(x\_label,y\_label,title):

plt.figure(figsize=(10,4))

plt.scatter(iris\_data.data[:,x\_label], iris\_data.data[:,y\_label], c=iris\_data.target, cmap=cm.viridis)

plt.xlabel(iris\_data.feature\_names[x\_label]);

plt.ylabel(iris\_data.feature\_names[y\_label]); plt.title(title)

plt.colorbar(ticks=([0, 1, 2]));

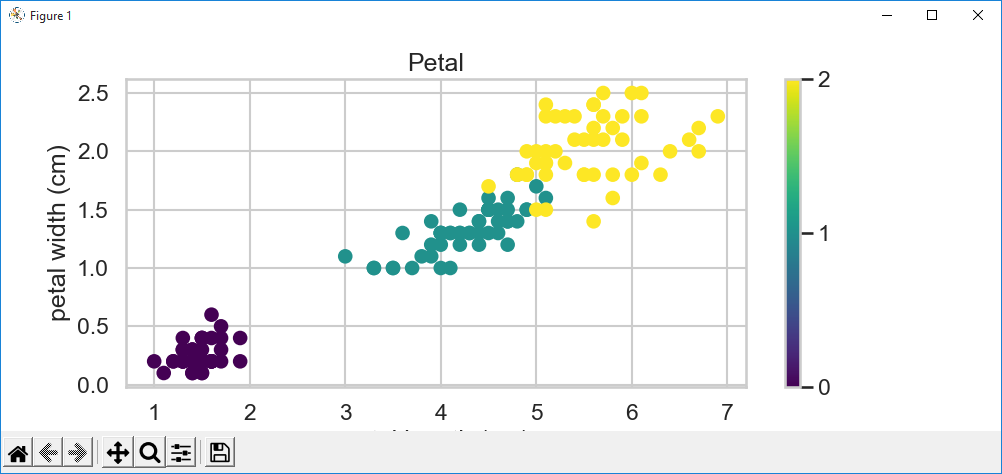
plt.show();x\_label = 2;y\_label=3;

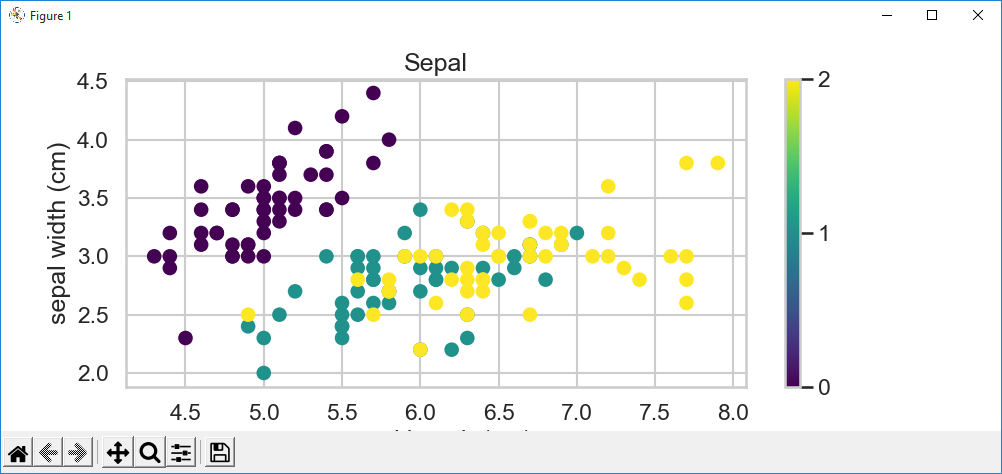
title='Petal'

Show\_Diagram(0,1,'Sepal')

Show\_Diagram(2,3,'Petal')

**Output:-**





**Assignment no 4**

**Assignment Name:-** Implements SVM Classifier (Iris Dataset)

**Name:** nishikanrt Pawar **Roll No:-**

**Class:-** Msc-1(cs) **Date:-**

**Code:-**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

def make\_meshgrid(x, y, h=0.02):

"""Create a mesh of points to plot in

Parameters

----------

x: data to base x-axis meshgrid on

y: data to base y-axis meshgrid on

h: stepsize for meshgrid, optional

Returns

-------

xx, yy : ndarray

"""

x\_min, x\_max = x.min() - 1, x.max() + 1

y\_min, y\_max = y.min() - 1, y.max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

return xx, yy

def plot\_contours(ax, clf, xx, yy, \*\*params):

"""Plot the decision boundaries for a classifier.

Parameters

----------

ax: matplotlib axes object

clf: a classifier

xx: meshgrid ndarray

yy: meshgrid ndarray

params: dictionary of params to pass to contourf, optional

"""

Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

out = ax.contourf(xx, yy, Z, \*\*params)

return out

# import some data to play with

iris = datasets.load\_iris()

# Take the first two features. We could avoid this by using a two-dim dataset

X = iris.data[:, :2]

y = iris.target

# we create an instance of SVM and fit out data. We do not scale our

# data since we want to plot the support vectors

C = 1.0 # SVM regularization parameter

models=(

svm.SVC(kernel="linear", C=C),

svm.LinearSVC(C=C, max\_iter=10000),

svm.SVC(kernel="rbf", gamma=0.7, C=C),

svm.SVC(kernel="poly", degree=3, gamma="auto", C=C),

)

models = (clf.fit(X, y) for clf in models)

# Set-up 2x2 grid for plotting.

fig, sub = plt.subplots(2, 2)

plt.subplots\_adjust(wspace=0.4, hspace=0.4)

X0, X1 = X[:, 0], X[:, 1]

title = ["SVC with linear kernel", "LinearSVC(linear kernel)", "SVC with RBF kernel","SVC with polynomial(degree 3)kernel"] # Define titles for your models

for clf, title, ax in zip(models, title, sub.flatten()):

plot\_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)

ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")

ax.set\_xlim(xx.min(), xx.max())

ax.set\_ylim(yy.min(), yy.max())

ax.set\_xlabel("Sepal length")

ax.set\_ylabel("Sepal width")

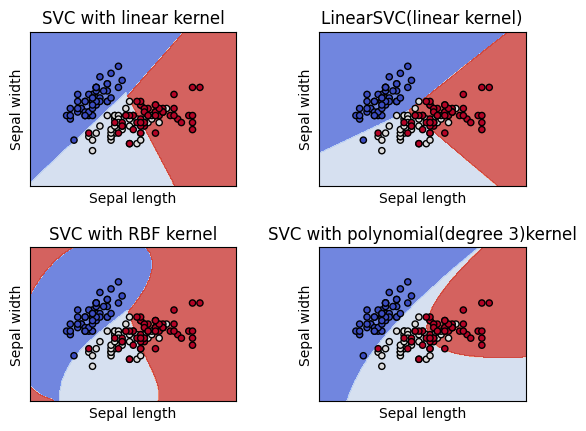
ax.set\_xticks(())

ax.set\_yticks(())

ax.set\_title(title)

plt.show()

OUTPUT:



**Assignment no 5**

**Assignment Name:-** Implements Train and fine-tune a Decision Tree For The Moons Dataset

**Name:** nishikanrt Pawar **Roll No:-**

**Class:-** Msc-1(cs) **Date:-**

**Code:-**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score, recall\_score

from sklearn.metrics import f1\_score

# This function will help in visualization of our dataset.

def plot\_dataset(X, y, axes):

plt.figure(figsize=(10,6))

plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs",alpha = 0.5)

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^",alpha = 0.2)

plt.axis(axes)

plt.grid(True, which='both')

plt.xlabel(r"$x\_1$", fontsize=20)

plt.ylabel(r"$x\_2$", fontsize=20, rotation=0)

X, y = make\_moons(n\_samples=10000, noise=0.4, random\_state=21)

plot\_dataset(X, y, [-3, 5, -3, 3])

plt.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size = 0.2)

tree\_clf = DecisionTreeClassifier()

parameter = {

'criterion' : ["gini", "entropy"],

'max\_leaf\_nodes': list(range(2, 50)),

'min\_samples\_split': [2, 3, 4]

}

clf = GridSearchCV(tree\_clf, parameter, cv = 5,scoring = "accuracy",return\_train\_score=True,n\_jobs=-1)

clf.fit(X\_train, y\_train)

clf.best\_params\_

cvres = clf.cv\_results\_

for mean\_score, params in zip(cvres["mean\_train\_score"], cvres["params"]):

print(mean\_score, params)

clf.score(X\_train, y\_train)

pred = clf.predict(X\_train)

confusion\_matrix(y\_train,pred)

pre = precision\_score(y\_train, pred)

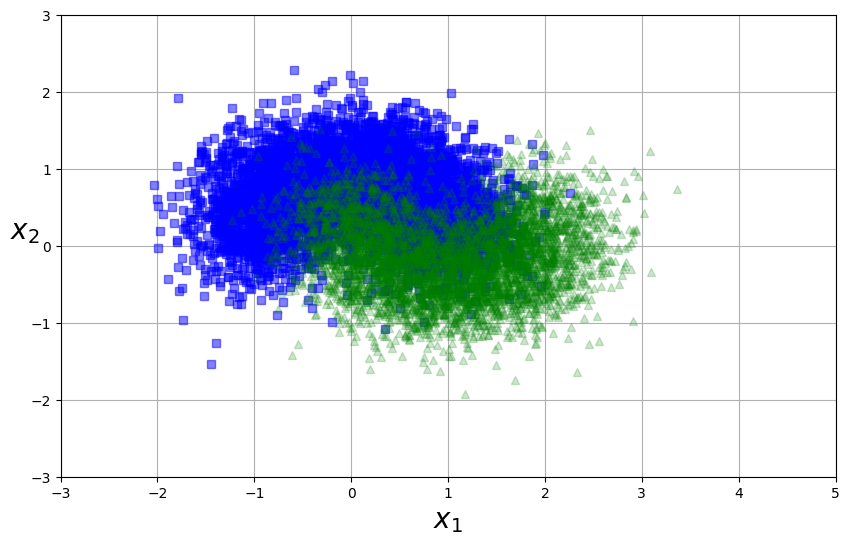
re = recall\_score(y\_train, pred)

print(f"Precision: {pre} Recall:{re}")

f1\_score(y\_train, pred)

clf.score(X\_test, y\_test)

OUTPUT:



0.7791875 {'criterion': 'gini', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 2}

0.7791875 {'criterion': 'gini', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 3}

0.7791875 {'criterion': 'gini', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 4}

0.82553125 {'criterion': 'gini', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 2}

0.82553125 {'criterion': 'gini', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 3}

0.82553125 {'criterion': 'gini', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 4}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 2}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 3}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 4}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 5, 'min\_samples\_split': 2}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 5, 'min\_samples\_split': 3}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 5, 'min\_samples\_split': 4}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 6, 'min\_samples\_split': 2}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 6, 'min\_samples\_split': 3}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 6, 'min\_samples\_split': 4}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 7, 'min\_samples\_split': 2}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 7, 'min\_samples\_split': 3}

0.86096875 {'criterion': 'gini', 'max\_leaf\_nodes': 7, 'min\_samples\_split': 4}

0.86234375 {'criterion': 'gini', 'max\_leaf\_nodes': 8, 'min\_samples\_split': 2}

Precision: 0.881307929969104 Recall:0.859402460456942

**Assignment no 6**

**Assignment Name:-** Implements Batch Grdaient Descent with early stopping for Softmax Regressions

**Name:** nishikanrt Pawar **Roll No:-**

**Class:-** Msc-1(cs) **Date:-**

**Code:-**

import numpy as np

#import scipy as sp

#import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

iris=load\_iris()

X=iris['data']

y=iris['target']

X\_with\_bias = np.c\_[np.ones([len(X), 1]), X]

np.random.seed(1234)

test\_ratio = 0.2

validation\_ratio = 0.2

total\_size = len(X\_with\_bias)

test\_size = int(total\_size \* test\_ratio)

validation\_size = int(total\_size \* validation\_ratio)

train\_size = total\_size - test\_size - validation\_size

rnd\_indices = np.random.permutation(total\_size)

X\_train = X\_with\_bias[rnd\_indices[:train\_size]]

y\_train = y[rnd\_indices[:train\_size]]

X\_valid = X\_with\_bias[rnd\_indices[train\_size:-test\_size]]

y\_valid = y[rnd\_indices[train\_size:-test\_size]]

X\_test = X\_with\_bias[rnd\_indices[-test\_size:]]

y\_test = y[rnd\_indices[-test\_size:]]

def one\_hot(Y):

nclasses=Y.max()+1

m = len(Y)

Y\_one\_hot=np.zeros((m,nclasses))

Y\_one\_hot[np.arange(m),Y]=1

return Y\_one\_hot

y\_valid[:10]

print(y\_valid)

print(one\_hot(y\_valid))

y\_train\_prob = one\_hot(y\_train)

y\_valid\_prob = one\_hot(y\_valid)

y\_test\_prob = one\_hot(y\_test)

def softmax(sk\_X):

top = np.exp(sk\_X)

bottom = np.sum(top,axis=1,keepdim=True)

return top/bottom

n\_inputs = X\_train.shape[1]

n\_outputs = len(np.unique(y\_train))

print (n\_inputs, n\_outputs)

OUTPUT:

[1 0 1 2 1 1 1 0 0 0 1 1 0 2 1 2 2 1 0 1 2 0 0 2 2 1 1 2 0 1]

[[0. 1. 0.]

[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]

[0. 1. 0.]

[0. 1. 0.]

[0. 1. 0.]

[1. 0. 0.]

[1. 0. 0.]

[1. 0. 0.]

[0. 1. 0.]

[0. 1. 0.]

[1. 0. 0.]

[0. 0. 1.]

[0. 1. 0.]

[0. 0. 1.]

[0. 0. 1.]

[0. 1. 0.]

[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]

[1. 0. 0.]

[1. 0. 0.]

[0. 0. 1.]

[0. 0. 1.]

[0. 1. 0.]

[0. 1. 0.]

[0. 0. 1.]

[1. 0. 0.]

[0. 1. 0.]]

5 3

**Assignment no 7**

**Assignment Name:-** Implement MLP for classification of handwritten digits (MNIST Dataset)

**Name:** nishikanrt Pawar **Roll No:-**

**Class:-** Msc-1(cs) **Date:-**

**Code:-**

**In [1]:** import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

from sklearn.neural\_network import MLPClassifier

import numpy as np

​

​

**In [2]:** # 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

​

print(X.shape)

​

C:\Users\Mayuri\AppData\Roaming\Python\Python311\site-packages\sklearn\datasets\\_openml.py:932: FutureWarning: The default value of `parser` will change from `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `ImportError` will be raised from 1.4 if the dataset is dense and pandas is not installed. Note that the pandas parser may return different data types. See the Notes Section in fetch\_openml's API doc for details.

warn(

**Out [2]:** (70000, 784

**In [3]:** #Model Training

# Split the data into train/test sets

X\_train, X\_test = X[:60000], X[60000:]

y\_train, y\_test = y[:60000], y[60000:]

​

**In [4]:** 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,

)

​

**In [5]:** # fit the model on the training data

classifier.fit(X\_train, y\_train)

​

​

**Out [5]:** Iteration 1, loss = 0.42635367

Iteration 2, loss = 0.15133481

Iteration 3, loss = 0.11926082

Iteration 4, loss = 0.10128421

Iteration 5, loss = 0.08698448

Iteration 6, loss = 0.08018627

Iteration 7, loss = 0.07544472

Iteration 8, loss = 0.06650726

Iteration 9, loss = 0.06502276

Iteration 10, loss = 0.05670472

Iteration 11, loss = 0.05228727

Iteration 12, loss = 0.05194876

Iteration 13, loss = 0.04580530

Iteration 14, loss = 0.04507070

Iteration 15, loss = 0.04141424

Iteration 16, loss = 0.03988480

Iteration 17, loss = 0.03980626

Iteration 18, loss = 0.03593785

Iteration 19, loss = 0.03619045

Iteration 20, loss = 0.03170852

Iteration 21, loss = 0.03625168

Iteration 22, loss = 0.03089528

Iteration 23, loss = 0.02811976

Iteration 24, loss = 0.02744623

Iteration 25, loss = 0.02897251

Iteration 26, loss = 0.02917402

Iteration 27, loss = 0.02641567

Iteration 28, loss = 0.02823747

Iteration 29, loss = 0.02250106

Iteration 30, loss = 0.02400615

Iteration 31, loss = 0.02222493

Iteration 32, loss = 0.02331528

Iteration 33, loss = 0.02260558

Iteration 34, loss = 0.02344456

Iteration 35, loss = 0.02700298

Iteration 36, loss = 0.02201304

Iteration 37, loss = 0.01941030

Iteration 38, loss = 0.01950871

Iteration 39, loss = 0.02297387

Iteration 40, loss = 0.01865801

Iteration 41, loss = 0.01820111

Iteration 42, loss = 0.01884833

Iteration 43, loss = 0.01828604

Iteration 44, loss = 0.01976615

Iteration 45, loss = 0.01893485

Iteration 46, loss = 0.01474573

Iteration 47, loss = 0.01497717

Iteration 48, loss = 0.01962730

Iteration 49, loss = 0.02032470

Iteration 50, loss = 0.02027927

Iteration 51, loss = 0.01397859

Iteration 52, loss = 0.01611249

Iteration 53, loss = 0.01580507

Iteration 54, loss = 0.01449746

Iteration 55, loss = 0.01333669

Iteration 56, loss = 0.01726942

Iteration 57, loss = 0.01569697

Iteration 58, loss = 0.01700670

Iteration 59, loss = 0.01674167

Iteration 60, loss = 0.01958126

Iteration 61, loss = 0.01542470

Iteration 62, loss = 0.01663296

Iteration 63, loss = 0.00958946

Iteration 64, loss = 0.01540829

Iteration 65, loss = 0.01580617

Iteration 66, loss = 0.01417699

Iteration 67, loss = 0.02034291

Iteration 68, loss = 0.01089459

Iteration 69, loss = 0.00722730

Iteration 70, loss = 0.01463658

Iteration 71, loss = 0.02067891

Iteration 72, loss = 0.01635221

Iteration 73, loss = 0.01127840

Iteration 74, loss = 0.00846482

Iteration 75, loss = 0.01601730

Iteration 76, loss = 0.01544564

Iteration 77, loss = 0.01312495

Iteration 78, loss = 0.01609130

Iteration 79, loss = 0.00880175

Iteration 80, loss = 0.00698280

Iteration 81, loss = 0.00413808

Iteration 82, loss = 0.00812941

Iteration 83, loss = 0.00645263

Iteration 84, loss = 0.01887495

Iteration 85, loss = 0.02256672

Iteration 86, loss = 0.01192657

Iteration 87, loss = 0.00936322

Iteration 88, loss = 0.00618735

Iteration 89, loss = 0.00947515

Iteration 90, loss = 0.01071651

Iteration 91, loss = 0.01715785

Iteration 92, loss = 0.00871255

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)

**In [6]:** #Model Evaluation

​

print("Training set score: %f" % classifier.score(X\_train, y\_train))

print("Test set score: %f" % classifier.score(X\_test, y\_test))

​

​

**Out [6]:** Training set score: 0.997250

Test set score: 0.971500

**In [7]:** #cost function evolution

fig, axes = plt.subplots(1, 1)

axes.plot(classifier.loss\_curve\_, 'o-')

axes.set\_xlabel("number of iteration")

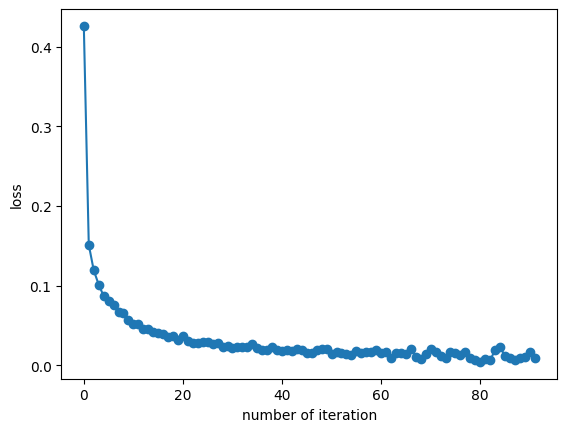
axes.set\_ylabel("loss")

plt.show()

​

​

**Out [7]:**



**In [8]:** len(classifier.intercepts\_) == len(classifier.coefs\_) == 4

**Out [8]:** True

**In [9]:** #Visualizing the learnt weights of the input layer

​

target\_layer = 0 #0 is input, 1 is 1st hidden etc

fig, axes = plt.subplots(1, 1, figsize=(15,6))

axes.imshow(np.transpose(classifier.coefs\_[target\_layer]), cmap=plt.get\_cmap("gray"), aspect="auto")

axes.set\_xlabel(f"number of neurons in {target\_layer}")

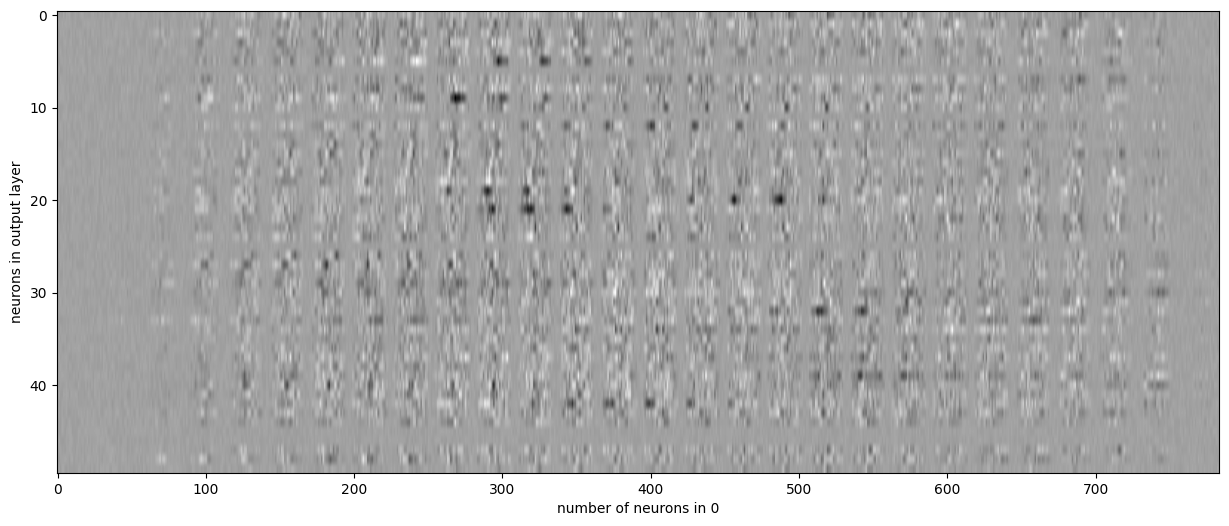
axes.set\_ylabel("neurons in output layer")

plt.show()

​

**Out [9]:**

​



**In [10]:** #For reshape 2d image

# choose layer to plot

target\_layer = 0 #0 is input, 1 is 1st hidden etc

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

vmin, vmax = classifier.coefs\_[0].min(), classifier.coefs\_[target\_layer].max()

for coef, ax in zip(classifier.coefs\_[0].T, axes.ravel()):

ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=0.5 \* vmin, vmax=0.5 \* vmax)

ax.set\_xticks(())

ax.set\_yticks(())

plt.show()

​

**Out [10]:**

