Machine learning

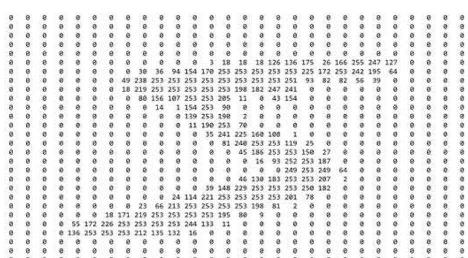
Soft-Max classifier

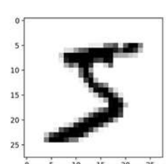
Morteza khorsand

- **Batch processing**

$$\begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \\ \vdots \\ x^{(m)} \end{bmatrix} \quad \begin{bmatrix} w^{(1)} & w^{(2)} & w^{(3)} & \cdots & w^{(n)} \\ & & \vdots & & \end{bmatrix} \quad \begin{bmatrix} s^{(1)} \\ s^{(2)} \\ s^{(3)} \\ \vdots \\ s^{(m)} \end{bmatrix}$$

$$scores = X@W + Bias$$



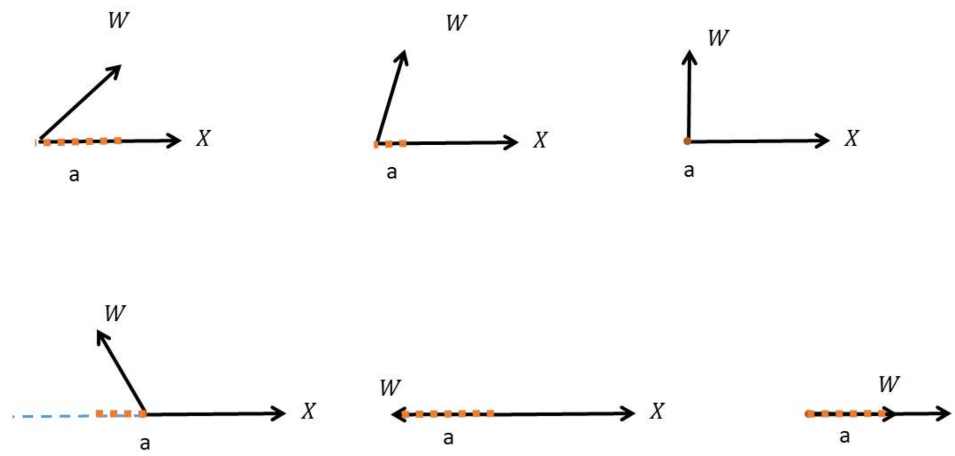


$$[0 \;\; 0 \;\; 0 \;\; \dots\dots\dots\dots\dots 30 \;\; 36 \;\; \dots\dots\dots..0]_{1*784}$$

$$\begin{bmatrix} w^{(1)} & w^{(2)} & w^{(3)} & \cdots & w^{(10)} \\ & & \vdots & & \end{bmatrix}_{784*10}$$

Scalar product - Inner product

$\boxed{x.w = \|x\| \, \|w\| \cos\theta}$ $\boxed{X.W = a \times |w|}$



In [12]:
```
import numpy as np

X= np.array([[10 , 0 ]])

w1=np.array([[0],
             [10]])

w2=np.array([[5],
             [5 ]])

w3=np.array([[-10],
             [0]])

w4=np.array([[1],
             [6]])

w5=np.array([[10],
             [0]])
```

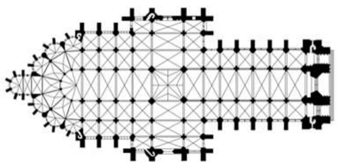
In [13]:
```
print(np.dot(X ,w1))
print(np.dot(X ,w2))
print(np.dot(X ,w3))
print(np.dot(X ,w4))
print(np.dot(X ,w5))
```

```
[[0]]
[[50]]
[[-100]]
[[10]]
[[100]]
```

Loading [MathJax]/extensions/Safe.js

- Softmax loss function – cross-entropy loss function

20*10



Amiens Cathedral floorplan. Gothic cathedrals built in France during the 13th century

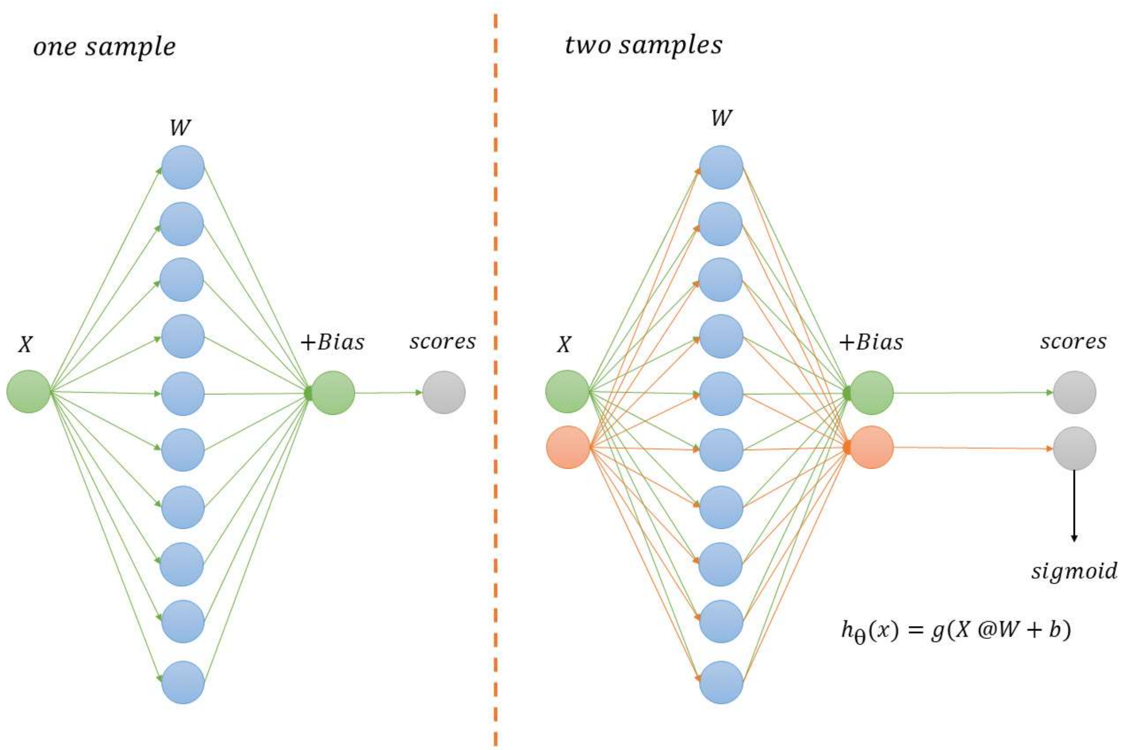
```
def softmax_loss(scores ,y):

    #softmax loss implementation
    #y in ended as a one-hot vector
    p = softmax(scores)
    return –np.sum(y*np.log(p))
```

$$l_i = -\log p(Y = y^{(i)} \mid X = x^{(i)}) = -\log \left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)$$

$$l_i = \sum_{i=1}^{c} -y_k \log p_k$$

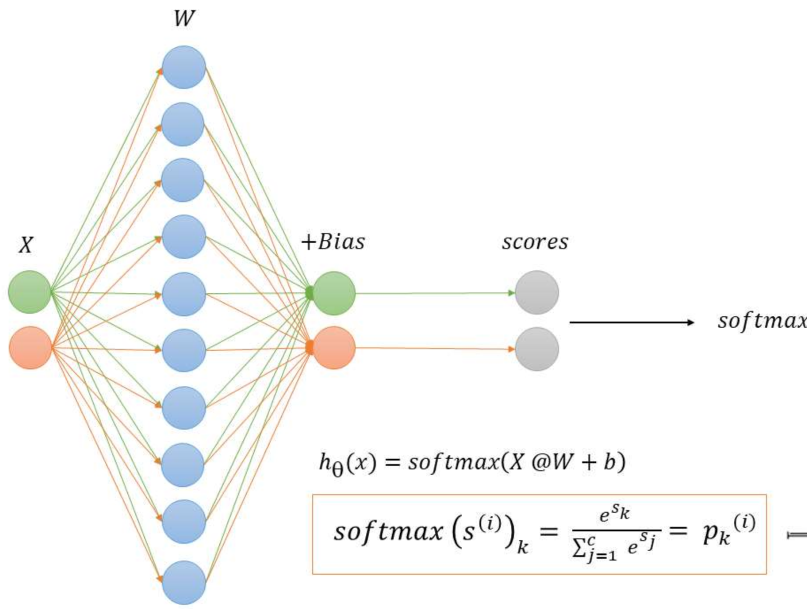
Y= $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ p= $\begin{bmatrix} p_1 \\ p_2 \\ \cdot \\ \cdot \\ p_c \end{bmatrix}$ $\longrightarrow$ H(p) = $\sum_{i=1}^{c} -p_i \log p_i$ Entropy H

style	scores	e^{s_k}	$\frac{e^{s_k}}{\sum_{j=1}^{c} e^{s_j}}$	- Log (softmax)	one-hot encode Desired y
Gothic	1.16	$e^{1.16} = 3.18$	0.05	3	1
Baroque	3.91	$e^{3.91} = 49.89$	0.90	0.1	0
Modern	-3.44	$e^{-3.44} = 0.03$	0.0005	7.60	0



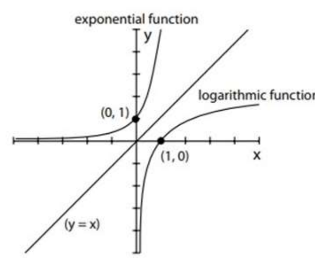
Softmax Function

normalized exponential function



$$h_\theta(x) = softmax(X \, @W + b)$$

$$\boxed{softmax\left(s^{(i)}\right)_k = \frac{e^{s_k}}{\sum_{j=1}^{c} e^{s_j}} = p_k{}^{(i)}}$$



Total Cost function

$$(w, b) = \frac{1}{m} \sum_{i=1}^{m} l_i + \lambda \, R(w)$$

$$(w, b) = \frac{1}{m} \sum_{i=1}^{m} (-\log p_y{}^{(i)}) + \lambda \, \|w\|_2{}^2$$

$$\boxed{(w, b) = \frac{1}{m} \sum_{i=1}^{m} \left(-\log \left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \, \|w\|_2{}^2}$$

$$\boxed{(w, b) = \frac{1}{m} \sum_{i=1}^{m} \left(-\log\left(\frac{e^{\left(\left(w^{y^{(i)}}\right)^T x^{(i)} + b^{(i)}\right)}}{\sum_{j=1}^{c} e^{\left(\left(w^{(j)}\right)^T x^{(i)} + b^{(j)}\right)}}\right)\right) + \lambda \, \|w\|_2{}^2}$$

- **Cost function derivative**

$$l_i = -\log p_y{}^{(i)} \qquad P_k = \left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right) \qquad S_k = (w^{(k)})^T\, x^{(i)} + b^{(k)}$$

$$\frac{\partial\, l_i}{\partial w^k} = \frac{\partial\, l_i}{\partial\, p_y{}^i} \times \frac{\partial\, p_y{}^i}{\partial\, s_k} \times \frac{\partial\, s_k}{\partial\, p_w{}^k}$$

K= $y^{(i)} \longrightarrow \frac{\partial\, l_i}{\partial w^k} = \left(-\frac{1}{p_{y^{(i)}}}\right) .\ p_k\ (1 - p_k)\ .\ x^{(i)} = \boxed{(p_k\text{-}1)\ .\ x^{(i)}}$

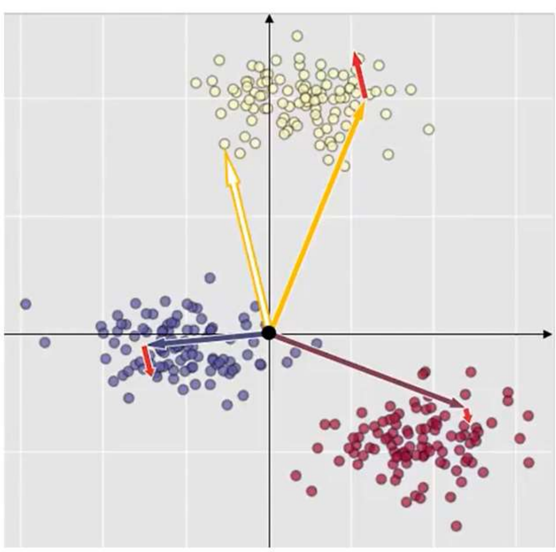
K$\neq y^{(i)} \longrightarrow \frac{\partial\, l_i}{\partial w^k} = \left(-\frac{1}{p_{y^{(i)}}}\right) .\ p_y{}^{(i)}\ (-p_k)\ .\ x^{(i)} = \boxed{(p_k)\ .\ x^{(i)}}$

K= $y^{(i)}$

K$\neq y^{(i)}$

W- derivative = $\frac{1}{m}$ (X.T @ (softmax(scores) – yencode))

bias - derivative = $\frac{1}{m}$ (sum (softmax(scores) – yencode))



K = $y^{(i)} \longrightarrow w^{(k)} = w^{(k)} - \alpha\ (p_k - 1)\ .\ x^{(i)}$

K $\neq y^{(i)} \longrightarrow w^{(k)} = w^{(k)} - \alpha\ (p_k)\ .\ x^{(i)}$


```
), 1, 2, 2])
```

```python
X = np.array([[0.1, 0.5],
              [1.1, 2.3],
              [-1.1, -2.3],
              [-1.5, -2.5]])


c= np.unique(y).shape[0]       #NUMBER OF CLASSES
m,n=X.shape                   #number of features
```

In [15]:
```python
#create functions


#scores
def scores(w , b , x):
    scores = x@w + b
    return scores


#softmax
def softmax(scores):
    return np.exp(scores) / np.sum(np.exp(scores), axis = 1, keepdims = True)


#one hot
def one_hot_encode(y):
    n_class = np.unique(y).shape[0]
    y_encode = np.zeros((y.shape[0], n_class))
    for idx, val in enumerate(y):
        y_encode[idx, val] = 1.0

    return y_encode



#croess entropy
def cross_entropy_cost(y, smax):
    return np.mean(-np.sum(y * np.log(smax), axis = 1))


loss = -np.sum(np.log(probs[range(m), y])) / m      #insted of one hot-vector
```

In [16]:
```python
def train(x, y, iterations = 100, learning_rate = 0.0001 , stopping_threshold = 1e-6):

    n=X.shape[1]
    m=X.shape[0]

    ##Initializing weight, bias, learning rate and iterations
    current_weight = 0.01 * np.random.randn(n,c)
    current_bias = np.zeros(c)

    iterations = iterations
    learning_rate = learning_rate


    costs = []
    weights = []
    previous_cost = None

    for i in range(iterations):
```

```python
        # Making predictions
        scores = (X.dot(current_weight)) + current_bias


        smax= softmax(scores)


        y_hot_vector= one_hot_encode(y)    # initial y [0,1,2,2]

        # Calculationg the current cost
        current_cost= cross_entropy_cost(y_hot_vector, smax)



        # If the change in cost is less than or equal to
        # stopping_threshold we stop the gradient descent
        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
            break


        previous_cost = current_cost

        costs.append(current_cost)
        weights.append(current_weight)

        # Calculating the gradients
        #dscores = np.copy(scores)
        #dscores-=1.0

        weight_derivative = (1/m)*np.dot(X.T, (smax-y_hot_vector))
        bias_derivative= (1/m)*np.sum(smax - y_hot_vector)




        # Updating weights and bias
        current_weight = current_weight - (learning_rate * weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)

        # Printing the parameters for each 1000th iteration
        #print(f"Iteration {i+1}: Cost {current_cost}, Weight \
        #{current_weight}, Bias {current_bias}")

    return current_weight, current_bias
```

In [17]:
```python
train(X, y, iterations = 2000, learning_rate = 0.5 , stopping_threshold = 1e-6)
```

Out[17]:
```
(array([[-8.1792237 ,  9.54689472, -1.35619975],
        [ 5.87947881, -1.09308965, -4.79068001]]),
 array([2.35786765e-16, 2.35786765e-16, 2.35786765e-16]))
```

In [18]:
```python
w=np.array([[-8.17822178,  9.54629569, -1.35273571],
            [ 5.88597534, -1.0858968 , -4.78466017]])

b= np.array([1.62053246e-15, 1.62053246e-15, 1.62053246e-15])

Loading [MathJax]/extensions/Safe.js b ,X)
```

```
def to_classlabel(z):
    return z.argmax(axis = 1)



print(" the actural label is [ 0 1 2 2] and your predicted is : ", to_classlabel(smax)
```

```
 the actural label is [ 0 1 2 2] and your predicted is :  [0 1 2 2]
```

In [19]:
```
def predict(X, w, b):

    # X --> Input.
    # w --> weights.
    # b --> bias.

    # Predicting
    z = X@w + b
    y_hat = softmax(z)

    # Returning the class with highest probability.
    return np.argmax(y_hat, axis=1)
```

In [20]:
```
def accuracy(y, y_hat):
    return np.sum(y==y_hat)/len(y)
```

In [ ]:
```
# Accuracy for training set.
train_preds = predict(X_train, w, b)
accuracy(train_y, train_preds)

# Accuracy for test set.
# Flattening and normalizing.
X_test = test_X.reshape(10000,28*28)
X_test = X_test/255
test_preds = predict(X_test, w, b)
accuracy(test_y, test_preds)
```

In [ ]:
```
fig = plt.figure(figsize=(15,10))
for i in range(40):
    ax = fig.add_subplot(5, 8, i+1)
    ax.imshow(test_X[i], cmap=plt.get_cmap('gray'))

    ax.set_title('y: {y}/ y_hat: {y_hat}'.format(y=test_y[i], y_hat=test_preds))
    plt.axis('off')
```

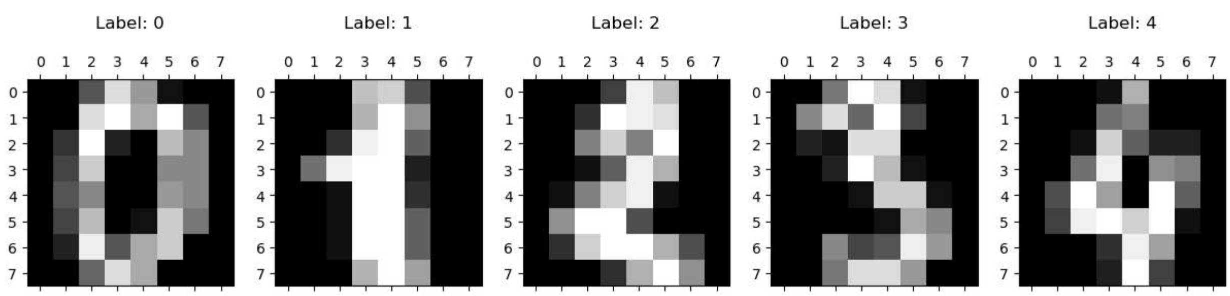
Sklearn

In [22]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import model_selection
from sklearn import linear_model
from sklearn import metrics
```

In [23]:
```python
digits_df = datasets.load_digits()
print('Digits dataset structure= ', dir(digits_df))
print('Data shape= ', digits_df.data.shape)
print('Data conatins pixel representation of each image, \n', digits_df.data)
```

```
Digits dataset structure=  ['DESCR', 'data', 'feature_names', 'frame', 'images', 'tar
get', 'target_names']
Data shape=  (1797, 64)
Data conatins pixel representation of each image,
 [[ 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.]]
```

In [24]:
```python
# Using subplot to plot the digits from 0 to 4
rows = 1
columns = 5
fig, ax =  plt.subplots(rows, columns, figsize = (15,6))

plt.gray()
for i in range(columns):
    ax[i].matshow(digits_df.images[i])
    ax[i].set_title('Label: %s\n' % digits_df.target_names[i])

plt.show()
```



In [25]:
```python
X = digits_df.data
y = digits_df.target
```

In [26]:
```python
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size= 0

print('X_train dimension= ', X_train.shape)
print('X_test dimension= ', X_test.shape)
print('y_train dimension= ', y_train.shape)
print('y_train dimension= ', y_test.shape)
```

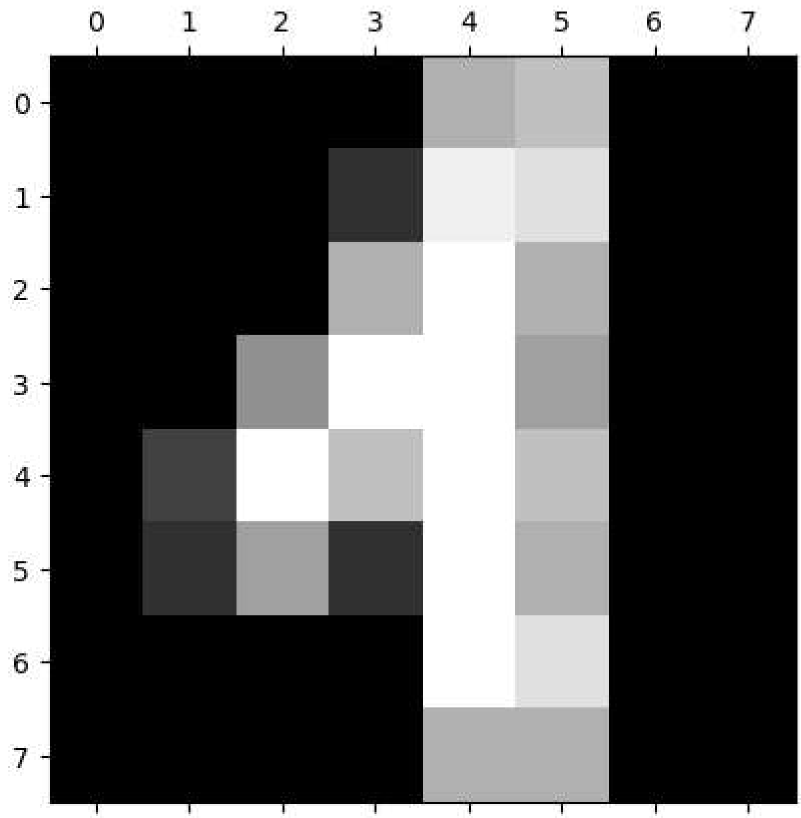
```
X_train dimension=  (1437, 64)
X_test dimension=  (360, 64)
y_train dimension=  (1437,)
y_train dimension=  (360,)
```

In [27]:
```python
lm = linear_model.LogisticRegression(multi_class='ovr', solver='liblinear')    #ovr :
lm.fit(X_train, y_train)
```

Out[27]:
```
LogisticRegression(multi_class='ovr', solver='liblinear')
```

In [28]:
```
print('Predicted value is =', lm.predict([X_test[200]]))

print('Actual value from test data is %s and corresponding image is as below' % (y_tes
plt.matshow(digits_df.images[200])
plt.show()
```

```
Predicted value is = [4]
Actual value from test data is 4 and corresponding image is as below
```



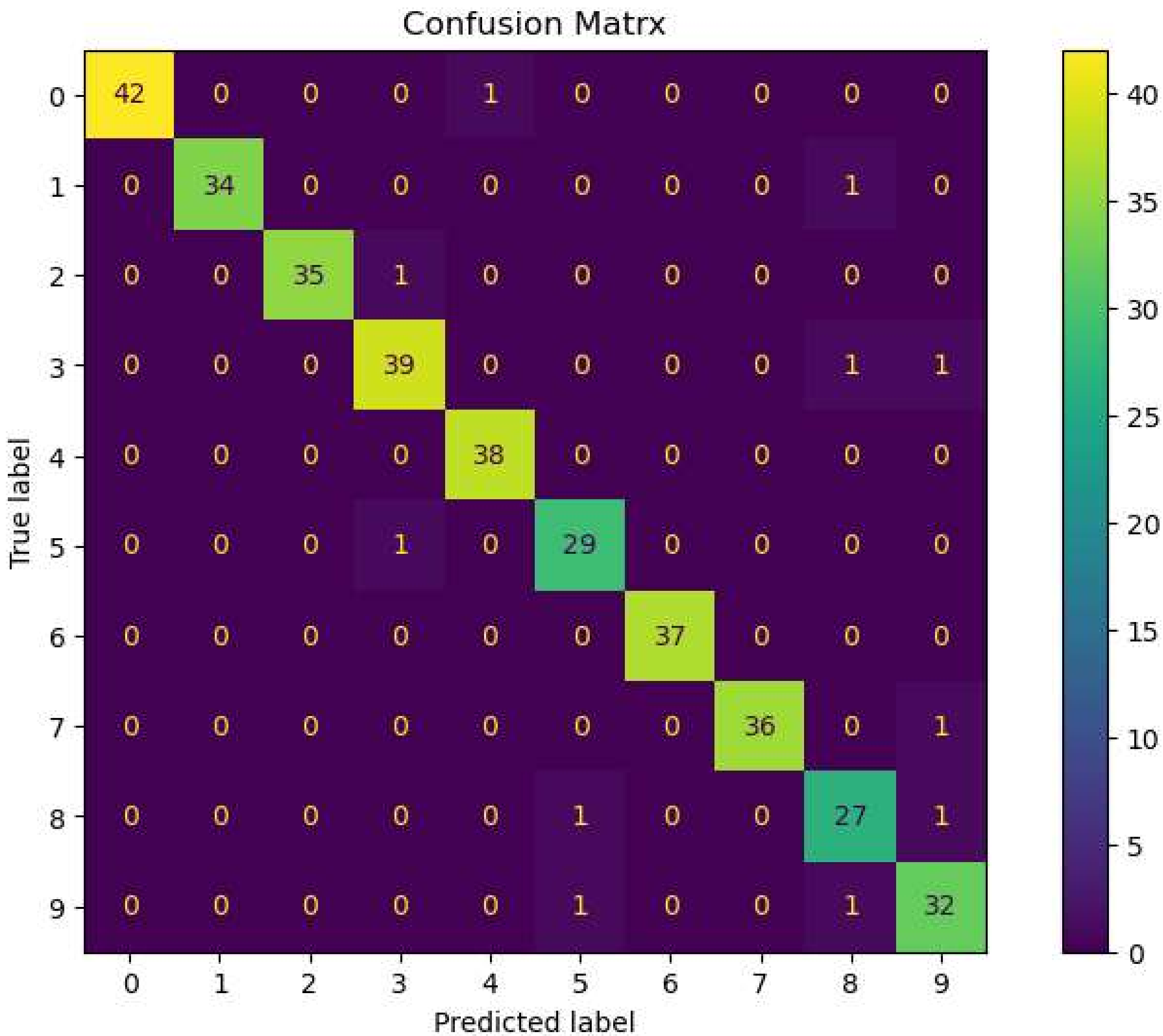
In [29]:
```
lm.score(X_test, y_test)
```

Out[29]:
```
0.9694444444444444
```

In [30]:
```
#Creating matplotlib axes object to assign figuresize and figure title
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title('Confusion Matrx')

disp =metrics.plot_confusion_matrix(lm, X_test, y_test, display_labels= digits_df.targ
disp.confusion_matrix
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWar
ning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix`
is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: Confus
ionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[30]:
```
array([[42,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0, 34,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0, 35,  1,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 39,  0,  0,  0,  0,  1,  1],
       [ 0,  0,  0,  0, 38,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0, 29,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 37,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 36,  0,  1],
       [ 0,  0,  0,  0,  0,  1,  0,  0, 27,  1],
       [ 0,  0,  0,  0,  0,  1,  0,  0,  1, 32]], dtype=int64)
```



In [31]:
```
print(metrics.classification_report(y_test, lm.predict(X_test)))
```

Loading [MathJax]/extensions/Safe.js

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        43
           1       1.00      0.97      0.99        35
           2       1.00      0.97      0.99        36
           3       0.95      0.95      0.95        41
           4       0.97      1.00      0.99        38
           5       0.94      0.97      0.95        30
           6       1.00      1.00      1.00        37
           7       1.00      0.97      0.99        37
           8       0.90      0.93      0.92        29
           9       0.91      0.94      0.93        34

    accuracy                           0.97       360
   macro avg       0.97      0.97      0.97       360
weighted avg       0.97      0.97      0.97       360
```

In [ ]:

Loading [MathJax]/extensions/Safe.js