**CS6301 MACHINE LEARNING LAB WEEK – 6 SVM**

**SRIHARI. S – 2018103601**

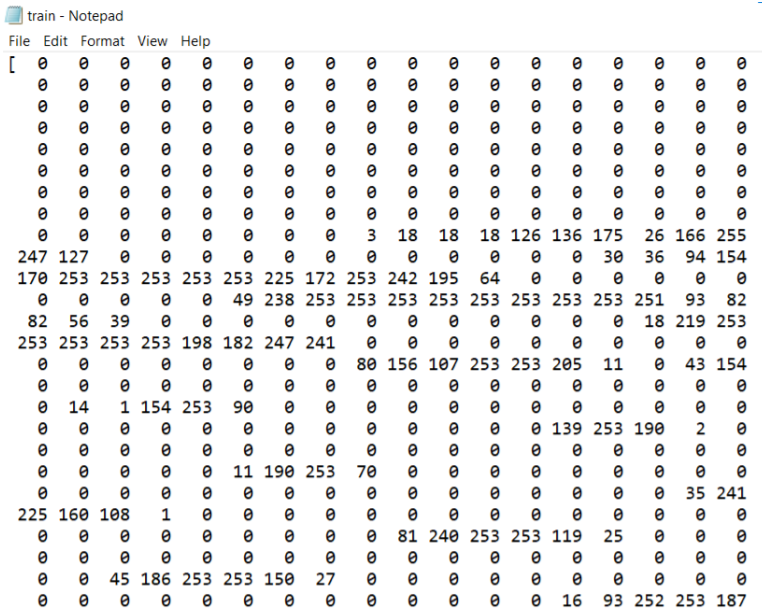
**Date**: 29-03-2021 Monday

**Aim**: To implement Support Vector Machine with different datasets and measure the performance metrics.

**Dataset-1:** MNIST Dataset

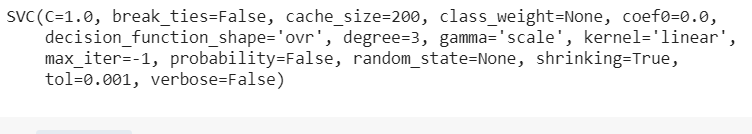
The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems which contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

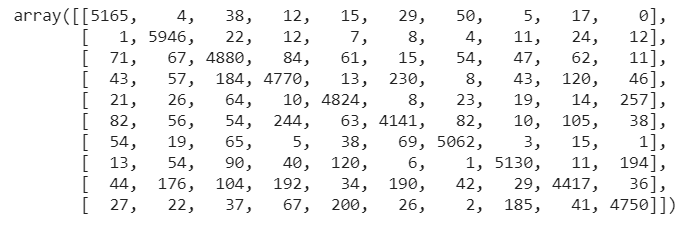
**Input:**



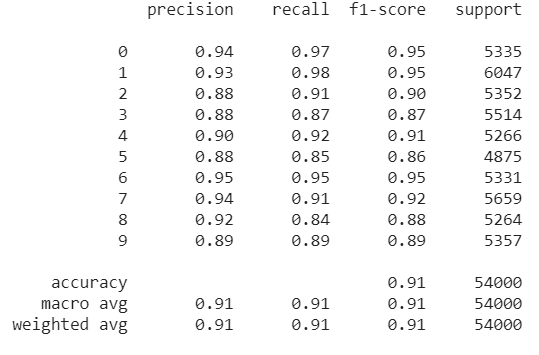
**Output:**

**LINEAR KERNEL**

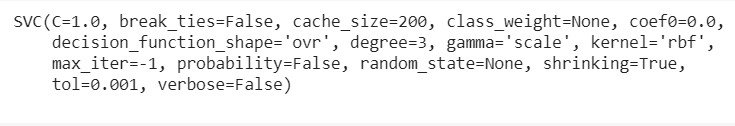




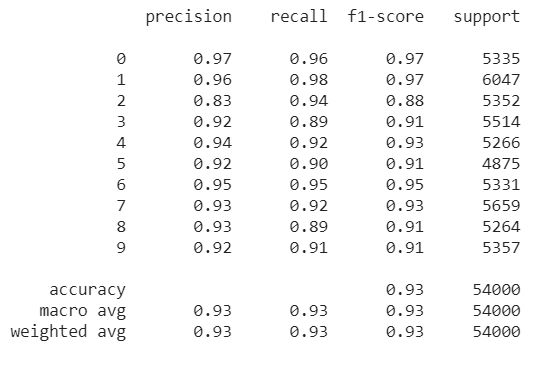
ACCURACY = 

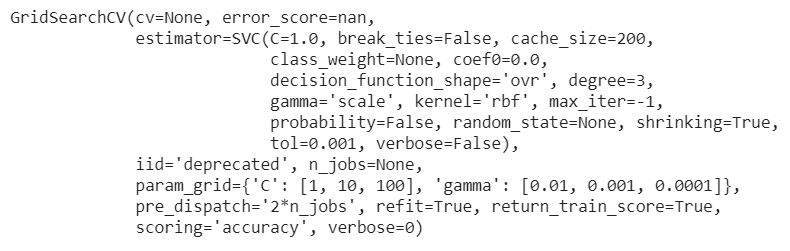


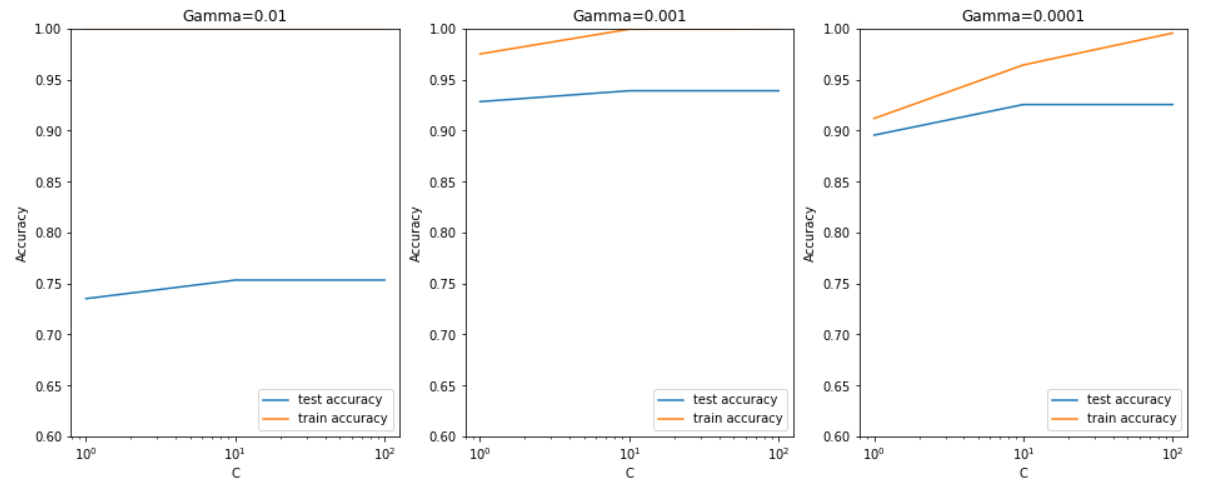
**NON-LINEAR KERNEL**

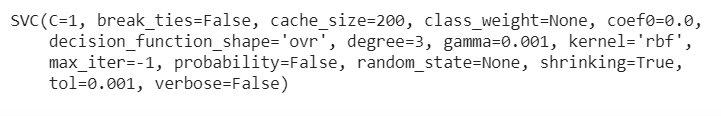


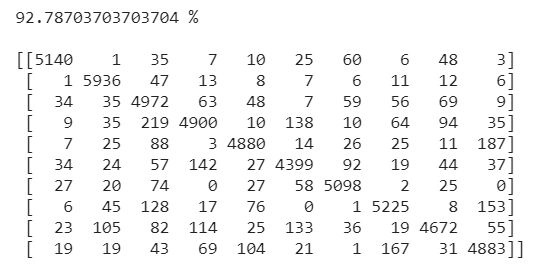
**ACCURACY =** 











**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import linear\_model

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

import gc

import cv2

# read the dataset

digits = pd.read\_csv("/content/drive/MyDrive/mnist\_train.csv")

digits.info()

four = digits.iloc[3, 1:]

four.shape

four = four.values.reshape(28, 28)

plt.imshow(four, cmap='gray')

# Summarise the counts of 'label' to see how many labels of each digit are present

digits.label.value\_counts()

# Summarise count in terms of percentage

100\*(round(digits.label.astype('category').value\_counts()/len(digits.index), 4))

# missing values - there are none

digits.isnull().sum()

description = digits.describe()

# Creating training and test sets

# Splitting the data into train and test

X = digits.iloc[:, 1:]

Y = digits.iloc[:, 0]

# Rescaling the features

from sklearn.preprocessing import scale

X = scale(X)

# train test split with train\_size=10% and test size=90%

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, train\_size=0.10, random\_state=101)

print(x\_train.shape)

print(x\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

from sklearn import svm

from sklearn import metrics

**#LINEAR KERNEL**

svm\_linear = svm.SVC(kernel='linear')

# fit

svm\_linear.fit(x\_train, y\_train)

# predict

predictions = svm\_linear.predict(x\_test)

predictions[:10]

# evaluation: accuracy

# C(i, j) represents the number of points known to be in class i

# but predicted to be in class j

confusion = metrics.confusion\_matrix(y\_true = y\_test, y\_pred = predictions)

confusion

# measure accuracy

metrics.accuracy\_score(y\_true=y\_test, y\_pred=predictions)

# class-wise accuracy

class\_wise = metrics.classification\_report(y\_true=y\_test, y\_pred=predictions)

print(class\_wise)

# run gc.collect() (garbage collect) to free up memory

# else, since the dataset is large and SVM is computationally heavy,

# it'll throw a memory error while training

gc.collect()

**#NON-LINEAR KERNEL**

# rbf kernel with other hyperparameters kept to default

svm\_rbf = svm.SVC(kernel='rbf')

svm\_rbf.fit(x\_train, y\_train)

predictions = svm\_rbf.predict(x\_test)

# accuracy

print(metrics.accuracy\_score(y\_true=y\_test, y\_pred=predictions))

# conduct (grid search) cross-validation to find the optimal values

# of cost C and the choice of kernel

from sklearn.model\_selection import GridSearchCV

parameters = {'C':[1, 10, 100], 'gamma': [1e-2, 1e-3, 1e-4]}

svc\_grid\_search = svm.SVC(kernel="rbf")

# create a classifier to perform grid search

clf = GridSearchCV(svc\_grid\_search, param\_grid=parameters,return\_train\_score=True, scoring='accuracy')

clf.fit(x\_train, y\_train)

cv\_results = pd.DataFrame(clf.cv\_results\_)

# converting C to numeric type for plotting on x-axis

cv\_results['param\_C'] = cv\_results['param\_C'].astype('int')

plt.figure(figsize=(16,6))

plt.subplot(131)

gamma\_01 = cv\_results[cv\_results['param\_gamma']==0.01]

plt.plot(gamma\_01["param\_C"], gamma\_01["mean\_test\_score"])

plt.plot(gamma\_01["param\_C"], gamma\_01["mean\_train\_score"])

plt.xlabel('C')

plt.ylabel('Accuracy')

plt.title("Gamma=0.01")

plt.ylim([0.60, 1])

plt.legend(['test accuracy', 'train accuracy'], loc='lower right')

plt.xscale('log')

plt.subplot(132)

gamma\_001 = cv\_results[cv\_results['param\_gamma']==0.001]

plt.plot(gamma\_001["param\_C"], gamma\_001["mean\_test\_score"])

plt.plot(gamma\_001["param\_C"], gamma\_001["mean\_train\_score"])

plt.xlabel('C')

plt.ylabel('Accuracy')

plt.title("Gamma=0.001")

plt.ylim([0.60, 1])

plt.legend(['test accuracy', 'train accuracy'], loc='lower right')

plt.xscale('log')

plt.subplot(133)

gamma\_0001 = cv\_results[cv\_results['param\_gamma']==0.0001]

plt.plot(gamma\_0001["param\_C"], gamma\_0001["mean\_test\_score"])

plt.plot(gamma\_0001["param\_C"], gamma\_0001["mean\_train\_score"])

plt.xlabel('C')

plt.ylabel('Accuracy')

plt.title("Gamma=0.0001")

plt.ylim([0.60, 1])

plt.legend(['test accuracy', 'train accuracy'], loc='lower right')

plt.xscale('log')

plt.show()

best\_C = 1

best\_gamma = 0.001

svm\_final = svm.SVC(kernel='rbf', C=best\_C, gamma=best\_gamma)

svm\_final.fit(x\_train, y\_train)

predictions = svm\_final.predict(x\_test)

confusion = metrics.confusion\_matrix(y\_true = y\_test, y\_pred = predictions)

test\_accuracy = metrics.accuracy\_score(y\_true=y\_test, y\_pred=predictions)

print(test\_accuracy\*100, "%\n")

print(confusion)

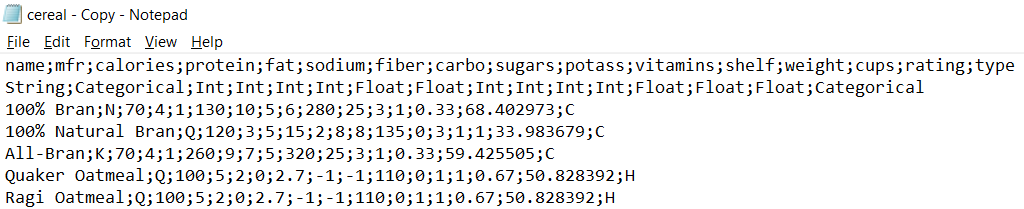
**DATASET 2: 2020/W36: Calories and Sugar in Cereals; dataset in makeover Monday**

**Link :** [2020/W36: Calories and Sugar in Cereals - dataset by makeovermonday | data.world](https://data.world/makeovermonday/2020w36/)

We determine the type of cereal (Hot/ Cold) based on the amount of proteins, fats, sodium, fiber, carbohydrates, sugar, potassium, vitamins, weight, cups and rating. Both hot and cold cereals have nutritional benefits, however the type of hot or cold cereal you choose may make a difference. Often times cold cereals are coated with sugar and have lost many of their nutrients through the milling process. Choosing a low-sugar, high-fiber cereal will help to improve nutritional values. Hot cereal, such as oatmeal, contains large amounts of fiber. The high fiber content of hot cereal aids in keeping you full longer and increases the amount of time until your next meal, which may also aid in weight loss.

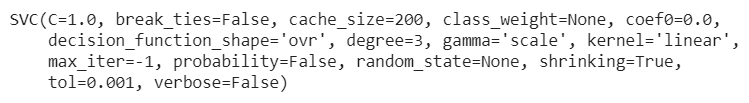
Hence by adopting linear and non-linear SVM we identify the type of cereal as hot or cold.

**INPUT**

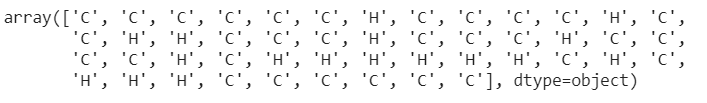


**OUTPUT**

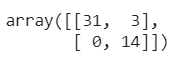
**LINEAR KERNEL – OUTPUT**



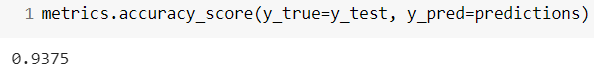
**PREDICTIONS:**

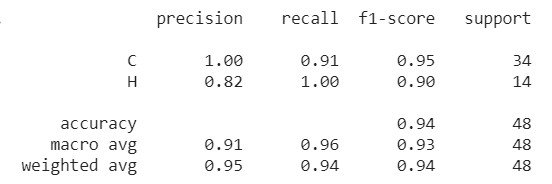


**CONFUSION MATRIX:**

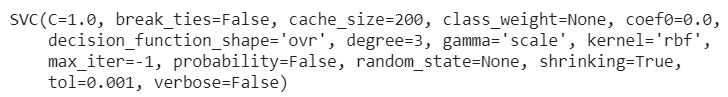


**ACCURACY:**

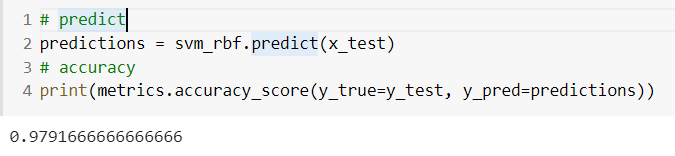


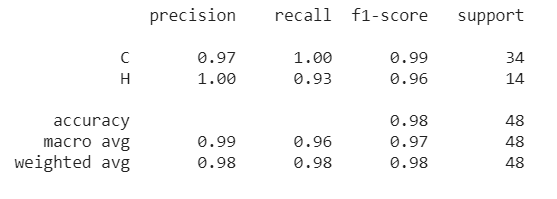


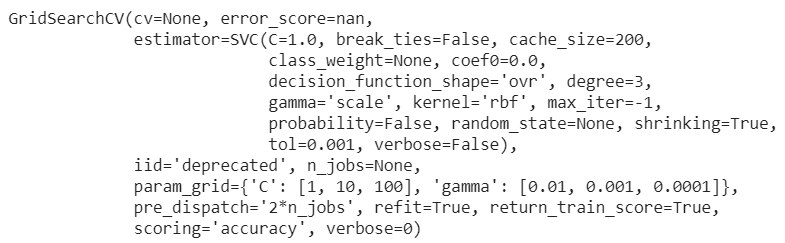
**NON LINEAR KERNEL**

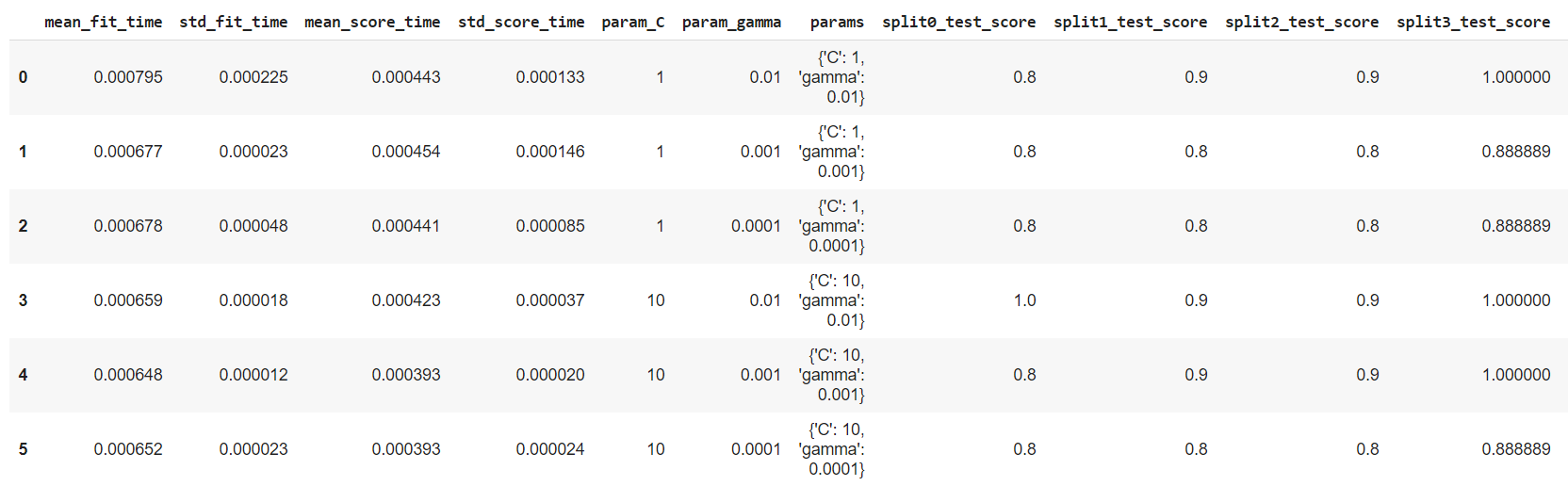


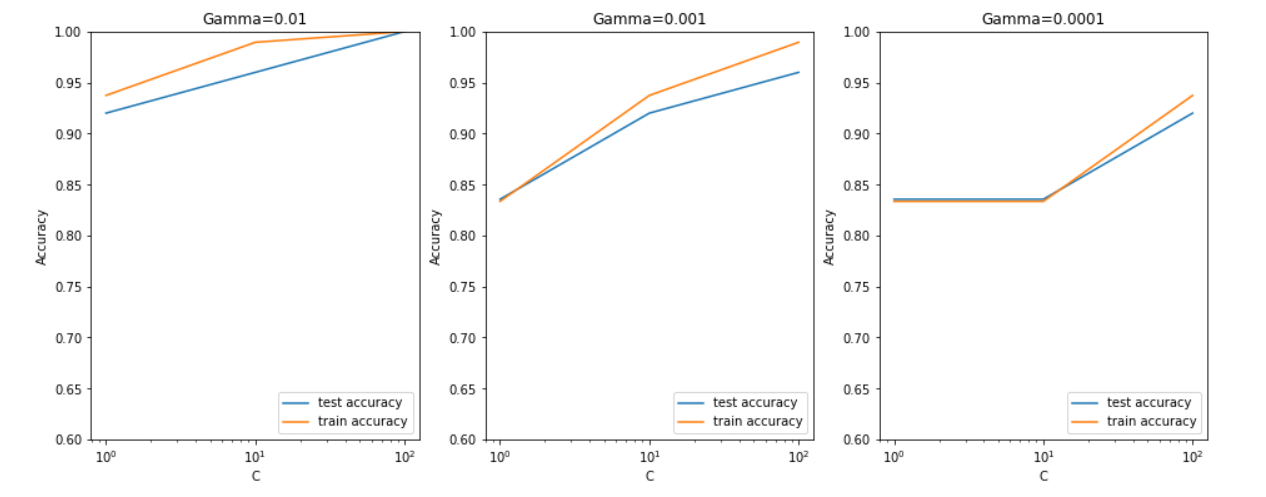
**ACCURACY:**

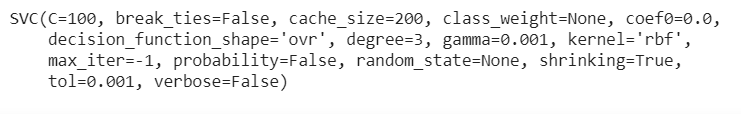




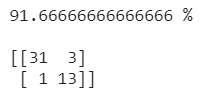








We see that the best values for C and Gamma is 100 and 0.001 respectively.



**CODE**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import linear\_model

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

import gc

import cv2

# read the dataset

digits = pd.read\_csv('/content/cereal - Copy.csv',sep=";")

digits.head()

X = digits.iloc[1:, 3:]

Y = digits.iloc[1:, 2]

# Rescaling the features

from sklearn.preprocessing import scale

X = scale(X)

# train test split with train\_size=10% and test size=90%

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, train\_size=0.5, random\_state=101)

print(x\_train.shape)

print(x\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

from sklearn import svm

from sklearn import metrics

svm\_linear = svm.SVC(kernel='linear')

svm\_linear.fit(x\_train, y\_train)

predictions = svm\_linear.predict(x\_test)

predictions

confusion = metrics.confusion\_matrix(y\_true = y\_test, y\_pred = predictions)

confusion

metrics.accuracy\_score(y\_true=y\_test, y\_pred=predictions)

class\_wise = metrics.classification\_report(y\_true=y\_test, y\_pred=predictions)

print(class\_wise)

# NON LINEAR KERNEL

# rbf kernel with other hyperparameters kept to default

svm\_rbf = svm.SVC(kernel='rbf')

svm\_rbf.fit(x\_train, y\_train)

# predict

predictions = svm\_rbf.predict(x\_test)

# accuracy

print(metrics.accuracy\_score(y\_true=y\_test, y\_pred=predictions))

class\_wise = metrics.classification\_report(y\_true=y\_test, y\_pred=predictions)

print(class\_wise)

from sklearn.model\_selection import GridSearchCV

parameters = {'C':[1, 10, 100],

             'gamma': [1e-2, 1e-3, 1e-4]}

# instantiate a model

svc\_grid\_search = svm.SVC(kernel="rbf")

# create a classifier to perform grid search

clf = GridSearchCV(svc\_grid\_search, param\_grid=parameters,return\_train\_score=True, scoring='accuracy')

# fit

clf.fit(x\_train, y\_train)

cv\_results = pd.DataFrame(clf.cv\_results\_)

cv\_results

# converting C to numeric type for plotting on x-axis

cv\_results['param\_C'] = cv\_results['param\_C'].astype('int')

# # plotting

plt.figure(figsize=(16,6))

# subplot 1/3

plt.subplot(131)

gamma\_01 = cv\_results[cv\_results['param\_gamma']==0.01]

plt.plot(gamma\_01["param\_C"], gamma\_01["mean\_test\_score"])

plt.plot(gamma\_01["param\_C"], gamma\_01["mean\_train\_score"])

plt.xlabel('C')

plt.ylabel('Accuracy')

plt.title("Gamma=0.01")

plt.ylim([0.60, 1])

plt.legend(['test accuracy', 'train accuracy'], loc='lower right')

plt.xscale('log')

# subplot 2/3

plt.subplot(132)

gamma\_001 = cv\_results[cv\_results['param\_gamma']==0.001]

plt.plot(gamma\_001["param\_C"], gamma\_001["mean\_test\_score"])

plt.plot(gamma\_001["param\_C"], gamma\_001["mean\_train\_score"])

plt.xlabel('C')

plt.ylabel('Accuracy')

plt.title("Gamma=0.001")

plt.ylim([0.60, 1])

plt.legend(['test accuracy', 'train accuracy'], loc='lower right')

plt.xscale('log')

# subplot 3/3

plt.subplot(133)

gamma\_0001 = cv\_results[cv\_results['param\_gamma']==0.0001]

plt.plot(gamma\_0001["param\_C"], gamma\_0001["mean\_test\_score"])

plt.plot(gamma\_0001["param\_C"], gamma\_0001["mean\_train\_score"])

plt.xlabel('C')

plt.ylabel('Accuracy')

plt.title("Gamma=0.0001")

plt.ylim([0.60, 1])

plt.legend(['test accuracy', 'train accuracy'], loc='lower right')

plt.xscale('log')

plt.show()

# optimal hyperparameters

best\_C = 100

best\_gamma = 0.001

# model

svm\_final = svm.SVC(kernel='rbf', C=best\_C, gamma=best\_gamma)

# fit

svm\_final.fit(x\_train, y\_train)

# predict

predictions = svm\_final.predict(x\_test)

# evaluation: CM

confusion = metrics.confusion\_matrix(y\_true = y\_test, y\_pred = predictions)

# measure accuracy

test\_accuracy = metrics.accuracy\_score(y\_true=y\_test, y\_pred=predictions)

print(test\_accuracy\*100, "%\n")

print(confusion)

**TABULAR INFERENCE**

|  |  |  |
| --- | --- | --- |
| **DATASET** | **MNIST** | **CEREAL** |
| **LINEAR KERNEL** |  |  |
| PRECISION | 0.91 | 0.91 |
| RECALL | 0.91 | 0.96 |
| F1-SCORE | 0.91 | 0.91 |
| ACCURACY | 90.9 % | 90.75 % |
| **NON-LINEAR KERNEL** |  |  |
| PRECISION | 0.92 | 0.97 |
| RECALL | 0.91 | 0.93 |
| F1-SCORE | 0.91 | 0.96 |
| ACCURACY | 92.8 % | 91.66 % |

**DATASET 1 – MNIST**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KERNEL** | **ACCURACY (%)** | **PRECISION** | **RECALL** | **F1-SCORE** |
| **Linear** | 90.9 | 0.91 | 0.91 | 0.91 |
| **Non-Linear** | 92.8 | 0.92 | 0.91 | 0.91 |

**ACCURACY IN NON – LINEAR KERNEL HYPERPARAMETER TUNING WHILE TRAINING**

|  |  |  |  |
| --- | --- | --- | --- |
| **Gamma\C** | **1** | **10** | **100** |
| **0.01** | 100% accuracy | 100% accuracy | 100% accuracy |
| **0.001** | 97% accuracy | 100% accuracy | 100% accuracy |
| **0.0001** | 92% accuracy | 96% accuracy | 99% accuracy |

**NON – LINEAR KERNEL HYPERPARAMETER TUNING WHILE TESTING**

|  |  |  |  |
| --- | --- | --- | --- |
| **Gamma\C** | **1** | **10** | **100** |
| **0.01** | 73% accuracy | 75% accuracy | 75% accuracy |
| **0.001** | 93% accuracy | 94% accuracy | 94% accuracy |
| **0.0001** | 89% accuracy | 93% accuracy | 93% accuracy |

**DATASET 2 – CEREAL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KERNEL** | **ACCURACY (%)** | **PRECISION** | **RECALL** | **F1-SCORE** |
| **Linear** | 90.75 | 0.91 | 0.96 | 0.91 |
| **Non-Linear** | 91.66 | 0.97 | 0.93 | 0.96 |

**ACCURACY IN NON – LINEAR KERNEL HYPERPARAMETER TUNING WHILE TRAINING**

|  |  |  |  |
| --- | --- | --- | --- |
| **Gamma\C** | **1** | **10** | **100** |
| **0.01** | 94% accuracy | 98% accuracy | 100% accuracy |
| **0.001** | 83% accuracy | 94% accuracy | 99% accuracy |
| **0.0001** | 83.5% accuracy | 83% accuracy | 94% accuracy |

**NON – LINEAR KERNEL HYPERPARAMETER TUNING WHILE TESTING**

|  |  |  |  |
| --- | --- | --- | --- |
| **Gamma\C** | **1** | **10** | **100** |
| **0.01** | 92% accuracy | 95% accuracy | 100% accuracy |
| **0.001** | 84% accuracy | 91.5% accuracy | 95.5% accuracy |
| **0.0001** | 83% accuracy | 83% accuracy | 92% accuracy |

**Inference: Thus, Support Vector Machine is implemented on the 2 datasets using linear and non-linear kernels and the results obtained are tabulated. Hyperparameters (C and gamma) are tuned to obtain a good accuracy.**