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:

train - Notepad

File Edit Format View Help

```
[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
  247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
  170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
    0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
   82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
  253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
  225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
```

Output:

LINEAR KERNEL

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
array([[5165,    4,   38,   12,   15,   29,   50,    5,   17,    0],
       [   1, 5946,   22,   12,    7,    8,    4,   11,   24,   12],
       [  71,   67, 4880,   84,   61,   15,   54,   47,   62,   11],
       [  43,   57,  184, 4770,   13,  230,    8,   43,  120,   46],
       [  21,   26,   64,   10, 4824,    8,   23,   19,   14,  257],
       [  82,   56,   54,  244,   63, 4141,   82,   10,  105,   38],
       [  54,   19,   65,    5,   38,   69, 5062,    3,   15,    1],
       [  13,   54,   90,   40,  120,    6,    1, 5130,   11,  194],
       [  44,  176,  104,  192,   34,  190,   42,   29, 4417,   36],
       [  27,   22,   37,   67,  200,   26,    2,  185,   41, 4750]])
```

ACCURACY = 0.9089814814814815

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      5335
           1       0.93      0.98      0.95      6047
           2       0.88      0.91      0.90      5352
           3       0.88      0.87      0.87      5514
           4       0.90      0.92      0.91      5266
           5       0.88      0.85      0.86      4875
           6       0.95      0.95      0.95      5331
           7       0.94      0.91      0.92      5659
           8       0.92      0.84      0.88      5264
           9       0.89      0.89      0.89      5357

    accuracy                           0.91     54000
   macro avg       0.91      0.91      0.91     54000
weighted avg       0.91      0.91      0.91     54000
```

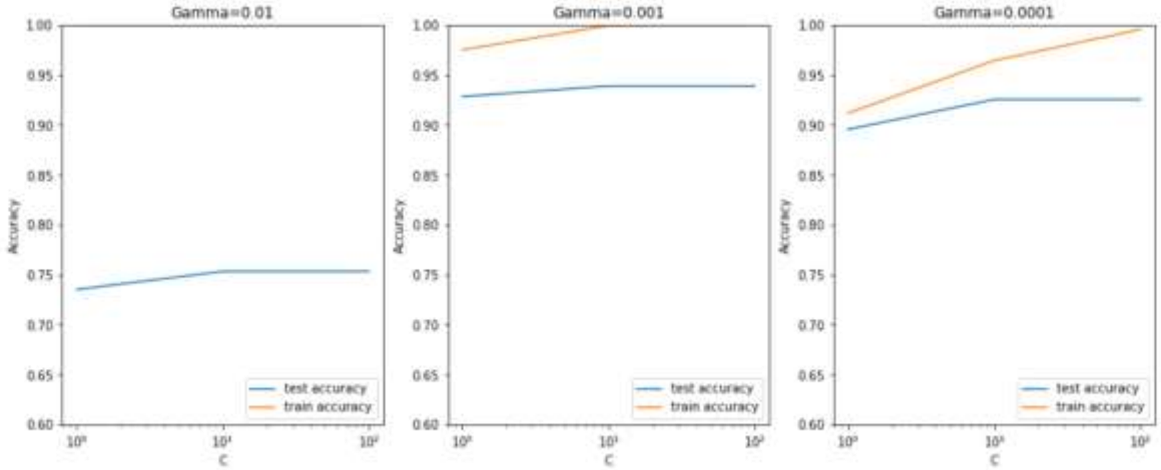
NON-LINEAR KERNEL

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

ACCURACY = 0.927962962962963

	precision	recall	f1-score	support
0	0.97	0.96	0.97	5335
1	0.96	0.98	0.97	6047
2	0.83	0.94	0.88	5352
3	0.92	0.89	0.91	5514
4	0.94	0.92	0.93	5266
5	0.92	0.90	0.91	4875
6	0.95	0.95	0.95	5331
7	0.93	0.92	0.93	5659
8	0.93	0.89	0.91	5264
9	0.92	0.91	0.91	5357
accuracy			0.93	54000
macro avg	0.93	0.93	0.93	54000
weighted avg	0.93	0.93	0.93	54000

```
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [1, 10, 100], 'gamma': [0.01, 0.001, 0.0001]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=0)
```




```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
92.78703703703704 %

[[5140    1   35    7   10   25   60    6   48    3]
 [   1 5936   47   13    8    7    6   11   12    6]
 [  34   35 4972   63   48    7   59   56   69    9]
 [   9   35  219 4900   10  138   10   64   94   35]
 [   7   25   88    3 4880   14   26   25   11  187]
 [  34   24   57  142   27 4399   92   19   44   37]
 [  27   20   74    0   27   58 5098    2   25    0]
 [   6   45  128   17   76    0    1 5225    8  153]
 [  23  105   82  114   25  133   36   19 4672   55]
 [  19   19   43   69  104   21    1  167   31 4883]]
```

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, scori
ng='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]()

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

cereal - Copy - Notepad

File Edit Format View Help

```
name;mfr;calories;protein;fat;sodium;fiber;carbo;sugars;potass;vitamins;shelf;weight;cups;rating;type
String;Categorical;Int;Int;Int;Int;Float;Float;Int;Int;Int;Int;Float;Float;Float;Categorical
100% Bran;N;70;4;1;130;10;5;6;280;25;3;1;0.33;68.402973;C
100% Natural Bran;Q;120;3;5;15;2;8;8;135;0;3;1;1;33.983679;C
All-Bran;K;70;4;1;260;9;7;5;320;25;3;1;0.33;59.425505;C
Quaker Oatmeal;Q;100;5;2;0;2.7;-1;-1;110;0;1;1;0.67;50.828392;H
Ragi Oatmeal;Q;100;5;2;0;2.7;-1;-1;110;0;1;1;0.67;50.828392;H
```

OUTPUT

LINEAR KERNEL – OUTPUT

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

PREDICTIONS:

```
array(['C', 'C', 'C', 'C', 'C', 'C', 'H', 'C', 'C', 'C', 'C', 'H', 'C',
       'C', 'H', 'H', 'C', 'C', 'C', 'H', 'C', 'C', 'C', 'H', 'C', 'C',
       'C', 'C', 'H', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'C', 'H', 'C',
       'H', 'H', 'H', 'C', 'C', 'C', 'C', 'C', 'C'], dtype=object)
```

CONFUSION MATRIX:

```
array([[31,  3],
       [ 0, 14]])
```

ACCURACY:

```
1 metrics.accuracy_score(y_true=y_test, y_pred=predictions)
```

```
0.9375
```

```
              precision    recall  f1-score   support

           C       1.00      0.91      0.95        34
           H       0.82      1.00      0.90        14

    accuracy                           0.94        48
   macro avg       0.91      0.96      0.93        48
weighted avg       0.95      0.94      0.94        48
```

NON LINEAR KERNEL

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

ACCURACY:

```
# predict
predictions = svm_rbf.predict(x_test)
# accuracy
print(metrics.accuracy_score(y_true=y_test, y_pred=predictions))
```

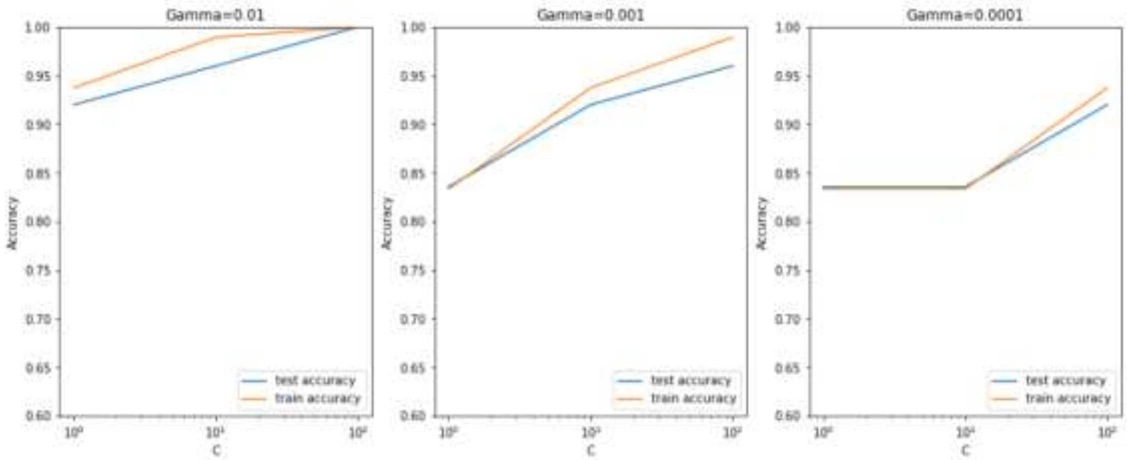
```
0.9791666666666666
              precision    recall  f1-score   support

           C       0.97      1.00      0.99        34
           H       1.00      0.93      0.96        14

    accuracy                           0.98        48
   macro avg       0.99      0.96      0.97        48
weighted avg       0.98      0.98      0.98        48
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [1, 10, 100], 'gamma': [0.01, 0.001, 0.0001]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=0)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score
0	0.000796	0.000225	0.000443	0.000133	1	0.01	{'C': 1, 'gamma': 0.01}	0.8	0.9	0.9	1.000000
1	0.000677	0.000023	0.000454	0.000146	1	0.001	{'C': 1, 'gamma': 0.001}	0.8	0.8	0.8	0.888889
2	0.000678	0.000048	0.000441	0.000085	1	0.0001	{'C': 1, 'gamma': 0.0001}	0.8	0.8	0.8	0.888889
3	0.000659	0.000018	0.000423	0.000037	10	0.01	{'C': 10, 'gamma': 0.01}	1.0	0.9	0.9	1.000000
4	0.000648	0.000012	0.000383	0.000020	10	0.001	{'C': 10, 'gamma': 0.001}	0.8	0.9	0.9	1.000000
5	0.000652	0.000023	0.000393	0.000024	10	0.0001	{'C': 10, 'gamma': 0.0001}	0.8	0.8	0.8	0.888889




```
SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

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

```
91.66666666666666 %

[[31  3]
 [ 1 13]]
```

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.